SMPTE REGISTERED DISCLOSURE DOCUMENT

TICO Lightweight Codec Used in IP Networked or in SDI Infrastructures

The attached document is a Registered Disclosure Document prepared by the sponsor identified below. It has been examined by the appropriate SMPTE Technology Committee and is believed to contain adequate information to satisfy the objectives defined in the Scope, and to be technically consistent.

This document is NOT a Standard, Recommended Practice or Engineering Guideline, and does NOT imply a finding or representation of the Society.

Errors in this document should be reported to the proponent identified below, with a copy to eng@smpte.org.

All other inquiries in respect of this document, including inquiries as to intellectual property requirements that may be attached to use of the disclosed technology, should be addressed to the proponent identified below.

Proponent contact information:

Jean-Baptiste Lorent
intoPIX SA
Rue Emile Francqui, 9
1435 Mont-Saint-Guibert
BELGIUM

Email: jb.lorent@intopix.com

Approved
March 24, 2016

Table of Contents

Introduction

TICO is a video compression scheme developed by intoPIX SA, which stands for Tiny Codec. TICO answers the need for a low latency and light weight visually lossless mezzanine compression to manage and carry UHDTV format across 3G-SDI and IP networks in production workflows. TICO can compress image data up to 10240 H x 10240 W x 12 bit per components.

This SMPTE RDD describes the TICO bitstream, the decoding process, the provisions for mapping bitstreams onto a single 3G-SDI link and the provisions for mapping bitstreams onto an IP network.

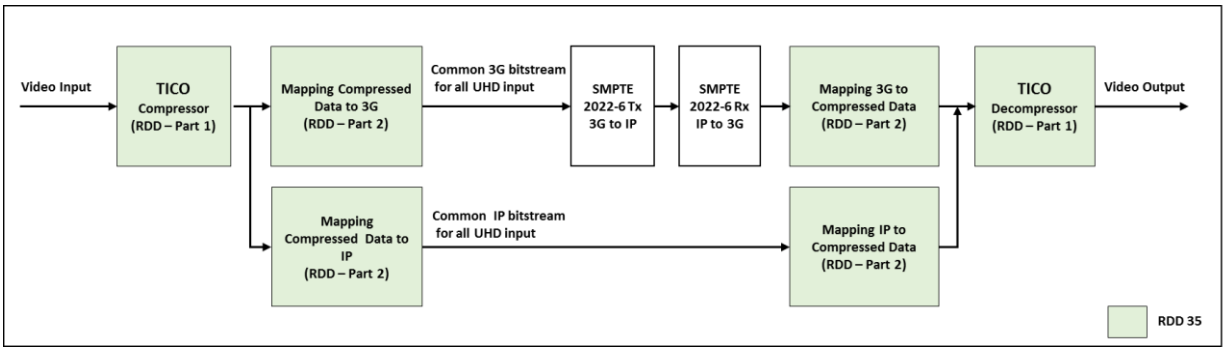

Figure 1 – TICO SMPTE RDD 35 scope

This SMPTE RDD contains sufficient information for knowledgeable individuals to build an implementation. An important feature of this mapping process is that it can be implemented in a practical form using currently available firmware and hardware. In the course of time, it could be implemented within new chip designs to ensure the minimum impact on hardware power requirements and physical size.

Note: “TICO” is trademark of intoPIX SA.

1 Scope

This Registered Disclosure Document (RDD):

- provides specification on TICO compression;
- specifies a TICO bitstream syntax containing information for interpreting the compressed image data;
- specifies TICO decoding processes for converting compressed image data to reconstructed image data based on TICO profile 2;
- specifies TICO bitstream mapping and detection in 3G-SDI environments (including protecting the EAV/SAV codes);
- specifies TICO bitstream mapping in IP networks with RTP mapping.

2 Normative References

Note: All references in this document to other SMPTE documents use the current numbering style (e.g. SMPTE ST 274:2008) although, during a transitional phase, the document as published (printed or PDF) may bear an older designation (such as SMPTE 274M-2008). Documents with the same root number (e.g. 274) and publication year (e.g. 2008) are functionally identical.

The following standards contain provisions which, through reference in this text, constitute provisions of this Registered Disclosure Document (RDD). At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this RDD are encouraged to investigate the possibility of applying the most recent editions of the standards listed below.

SMPTE ST 12-2:2014, Transmission of Time Code in the Ancillary Data Space

SMPTE ST 274:2008, Television — 1920 x 1080 Image Sample Structure, Digital Representation and Digital Timing Reference Sequences for Multiple Picture Rates

SMPTE ST 291-1:2011, Ancillary Data Packet and Space Formatting

SMPTE ST 292-1:2012,1.5 Gb/s Signal/Data Serial Interface

SMPTE ST 299-1:2009, 24-Bit Digital Audio Format for SMPTE 292 Bit-Serial Interface

SMPTE ST 299-2:2010, Extension of the 24-Bit Digital Audio Format to 32 Channels for 3 Gb/s Bit-Serial Interfaces

SMPTE ST 352:2013, Payload Identification Codes for Serial Digital Interfaces

SMPTE ST 424:2012, 3 Gb/s Signal/Data Serial Interface

SMPTE ST 425-1:2014, Source Image Format and Ancillary Data Mapping for the 3 Gb/s Serial Interface

SMPTE ST 425-3:2015, Image Format and Ancillary Data Mapping for the Dual Link 3 Gb/s Serial Interface

SMPTE ST 425-5:2015, Image Format and Ancillary Data Mapping for the Quad Link 3 Gb/s Serial Interface

SMPTE ST 2022-6:2012, Transport of High Bit Rate Media Signals Over IP Networks (HBRMT)

SMPTE ST 2059-1:2015, Generation and Alignment of Interface Signals to the SMPTE Epoch

SMPTE ST 2081-10:2015, 2160-Line and 1080-Line Source Image and Ancillary Data Mapping for Single Link 6G-SDI

SMPTE ST 2082-10:2015, 2160-Line Source Image and Ancillary Data Mapping for 12G-SDI

SMPTE ST 2084:2014, High Dynamic Range Electro-Optical Transfer Function of Mastering Reference Displays.

Recommendation ITU-R BT.601-7 (03/2011), Studio Encoding Parameters of Digital Television for Standard 4:3 and Wide-Screen 16:9 Aspect Ratios.

Recommendation ITU-R BT.709-6 (06/2015), Parameter Values for the HDTV Standards for Production and International Programme Exchange.

Recommendation ITU-R BT.2020-2 (10/2015), Parameter Values for Ultra-High Definition Television Systems for Production and International Programme Exchange.

3 TICO Compression

3.1 Notation

3.1.1 Mathematical Operators

3.1.1.1 Arithmetic Operators

+	Addition
–	Subtraction (as a binary operator) or negation (as a unary prefix operator)
*	Multiplication
÷	Division (used in mathematical equations where no truncation or rounding is intended)
/	Integer division with truncation of the result toward negative infinity: x / y = floor(x ÷ y)
n mod m	Modulo operator with modulus m. Defined only for integers n and m with m > 0. Result is remainder r after integer division of n by m, r = n − floor(n ÷ m) * m; 0 ≤ r ≤ m − 1.
<<	Left shift
>>	Right shift
=	Assignment
++	Increment by one
<	Smaller than
≤	Equal to or smaller than
==	Equal to
$\lceil x \rceil$	Ceiling operator
$\lfloor x \rfloor$	Floor operator

3.1.1.2 Mathematical Functions

Ceil(x)	Ceiling of x. Returns the smallest integer that is greater than or equal to x, x being a real number greater or equal than 0.
Floor(x)	Floor of x. Returns the highest integer that is less or equal to x, x being a real number greater or equal than 0.
Log2(x)	Logarithm of x to the base 2
Clamp(x,low,up)	Produces x when x in interval [low,up] else produces low (if x<low) or up (if x>up).
Max(x)	Maximum of x. Returns the maximum value

3.1.2 Terms and Definitions

3.1.2.1 Bitstream

Compressed image data using TICO compression. (For details, see section 3.3 Picture Structure).

3.1.2.2 Bit-plane

A one dimensional array of bits. A bit-plane refers to all the bits of the same position in a group of 4 consecutive binary-coded coefficients with a sign magnitude representation.

3.1.2.3 Coefficient

The values that are result of a wavelet transformation.

3.1.2.4 Component (comp)

A two-dimensional array of samples having the same designation in the output or display device. An image typically consists of several components, e.g. red, green and blue.

3.1.2.5 Decomposition level

A collection of wavelet sub-bands where each coefficient has the same spatial impact or span with respect to the source component samples.

3.1.2.6 Greatest Coded Line Index (GCLI)

Index of the most significant non-null bit-plane

3.1.2.7 Greatest Trimmed Line Index (GTLI)

Number of trimmed bits for a sub-band at a certain iteration

3.1.2.8 Picture

A segment of bitstream for one frame or field.

3.1.2.9 Precinct

A 2-pixel lines region of a transformed component, used as compression unit for limiting the size of packets.

3.1.2.10 Precision

Number of bits allocated to a particular sample, coefficient, or other binary numerical representation.

3.1.2.11 Quantization

A method of reducing the precision of the individual coefficients to reduce the number of bits used to entropy code them. This is equivalent to division while compressing and multiplication while decompressing. Quantization can be achieved by an explicit operation with a given quantization value.

3.1.2.12 Sign-magnitude

Binary representation of an integer value, starting with 1 sign bit (0 for positive integer, 1 for negative integer). The remaining bits in the number indicate the magnitude (or absolute value).

3.1.2.13 Sub-band (sb)

A group of transform coefficients from the same component resulting from the same sequence of low-pass and high-pass filtering operations, both vertically and horizontally.

3.2 TICO Characteristics Overview

3.2.1 Lightweight, High Quality and Low Latency in Software and Hardware

TICO algorithm offers simultaneously low complexity (in hardware and software), low latency and visually lossless quality at low compression ratio (typically 4:1).

- *Image quality* :

  The compression scheme provides visually lossless compression and is robust to multiple encoding generations. The compression can be mathematically lossless at low compression rate due to the use of the 5/3 wavelet for lossless (reversible) transform.

  The rate control mechanisms are guaranteeing an optimal quality within each precinct using contrast sensitivity function (CSF) for an optimal visual weighting and optimal transfer between precincts. TICO outputs at a constant bitrate (CBR) for an optimal quality.

- *Low latency:*

  Due to its line-based wavelet transforms and entropy coding, a hardware implementation can achieve a latency of few lines of pixels for the encoding or decoding process. The latency is also fixed. With parallel processing, a software (CPU) implementation can achieve a single frame of latency for the encoding or decoding process.

- *Lightweight and low power in hardware:*

  TICO should not require any external memory when implemented in a FPGA (& ASIC) since the codec needs only to buffer few lines of pixels. The structure of the algorithm, due to the use of light transforms, the entropy coding based on GCLIs, and the rate allocation does not require a large usage of logic. As a consequence, the codec does not consume a lot of power in a hardware circuit.

- *Lightweight and fast in software:*

  TICO is highly parallelizable within pictures, slices and precincts which makes it very efficient and fast in CPU.

3.2.2 Encoding / Decoding Block Diagram

The encoder consists of an optional color transform (CT), a vertical discrete wavelet transform (DWT), a horizontal discrete wavelet transform, an integer converter and quantizer (Q), a light entropy (GCLI) coder and a packer with rate allocation.

It is illustrated in the Figure 2.

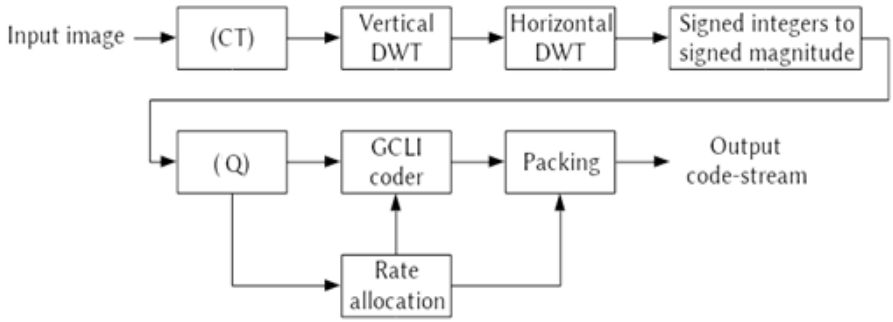

Figure 2 – TICO Encoding Block Diagram

The decoder consists of a bitstream parser, a light entropy (GCLI) decoder, an unpacker, an integer converter and dequantizer (DQ), a horizontal inverse discrete wavelet transform (DWT), a vertical inverse discrete wavelet transform and an optional inverse color transform (CT).

It is illustrated in the Figure 3.

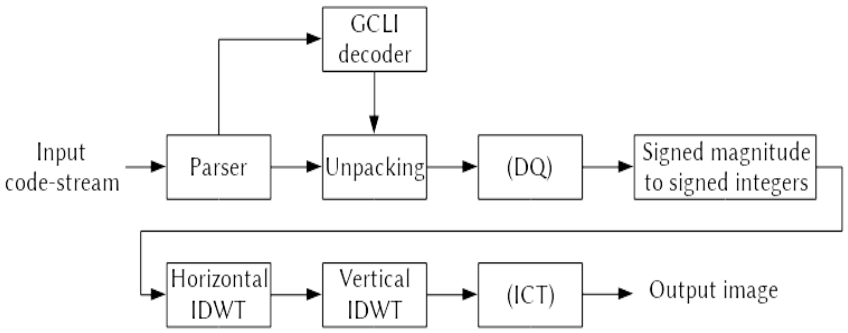

Figure 3 – TICO Decoding Block Diagram

3.3 Picture Structure

TICO encodes progressive pictures. Interlaced frames shall be processed by progressively encoding each field in sequence.

Each picture shall be divided horizontally into precincts. Precincts shall have 2 pixels height except on the last precinct if the image pixel height is not an even number.

Finally, the precincts of a picture shall be grouped into slices. Each slice shall have the same number of precincts except the last slice if the number of precincts contained in the picture is not an integer multiple of the slice height. Each slice may be encoded and decoded independently, which facilitates parallel encoding and decoding.

As an example, consider a progressive frame 3840 pixels wide by 2160 pixels high, with a desired slice size of 32 precincts (or 64 lines). There shall be 34 slices: each slice shall be 32 precincts except the last slice that shall only have 24 precincts. This is illustrated in Figure 4.

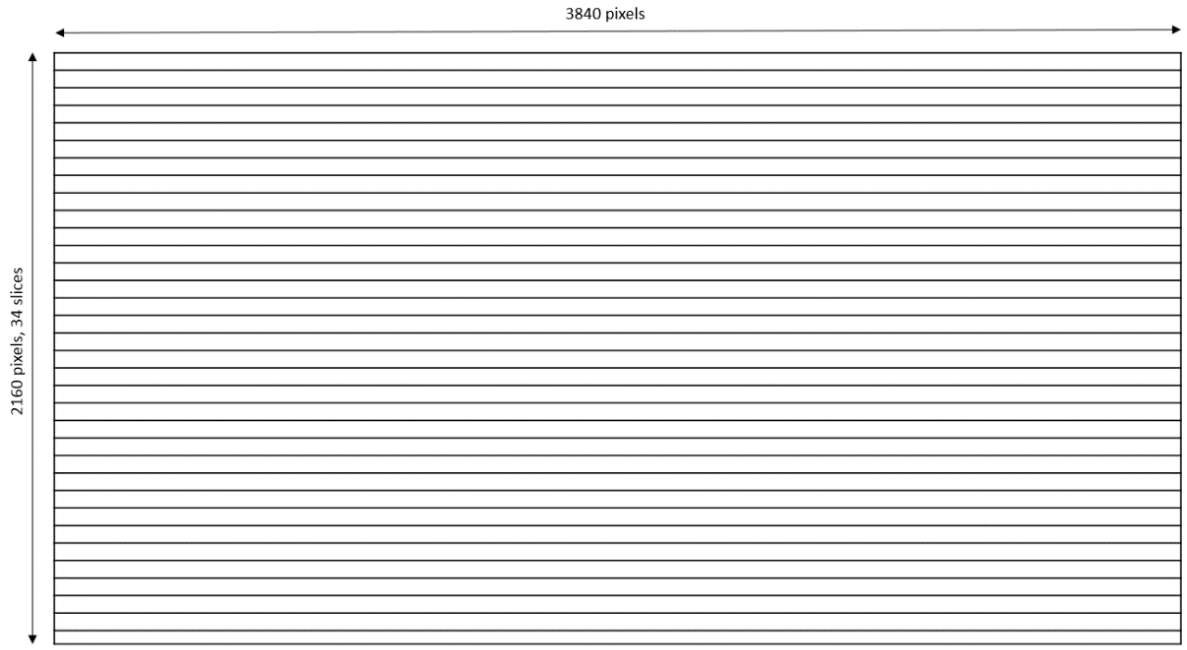

Figure 4 – Example of TICO picture slice arrangement

3.4 Bitstream Syntax

3.4.1 Overview

The formal description of TICO bitstream syntax is provided by the tables in this section.

The syntax description involves several special constructs. Syntax elements are the fundamental parameters that describe the compressed image, or direct the exact nature of the decoding process.

Syntax structures are collections of syntax elements. They are denoted by functions whose names use all lowercase letters with underscore characters separating words. The functions can take arguments, which identify a specific instance of the syntax structure or provide information relevant to the parsing or decoding process of the structure. The sizes of some syntax structures are specified in TICO bitstreams as described subsequently. Decoders can use these sizes to determine the start of the immediately following syntax structure.

Finally, syntax variables and syntax functions are used within syntax structures as part of the description of those structures. Syntax variables are defined implicitly by their use in the syntax tables.

TICO bitstream syntax elements belong to one of three categories: fixed-length bit strings, fixed-length numerical values, or variable-length codes. In the syntax tables, the “Descriptor” column indicates the category to which each bitstream syntax element belongs. Bit strings appear in the bitstream left bit first; numerical values appear most-significant bit first.

Table 1 – Categories of syntax elements

Categories of syntax elements	Notation
Fixed-length bit strings where n is the number of bits in the string	f(n)
Fixed-length numerical values. unsigned integers where n is the number of bits used to represent the value	u(n)
Variable-length bit strings. V is a variable name used to refer to that length (expressed in bits) in the rest of this section.	vl(V)

The syntax element descriptions tables have three columns. The first column shows the name of the syntax element. The second column indicates the category of the syntax element as defined in Table 1. The third column may contain an abbreviation used to refer to the value of the syntax element in the rest of this section.

Figure 5 gives an overview of the hierarchy of TICO bitstream syntax structures.

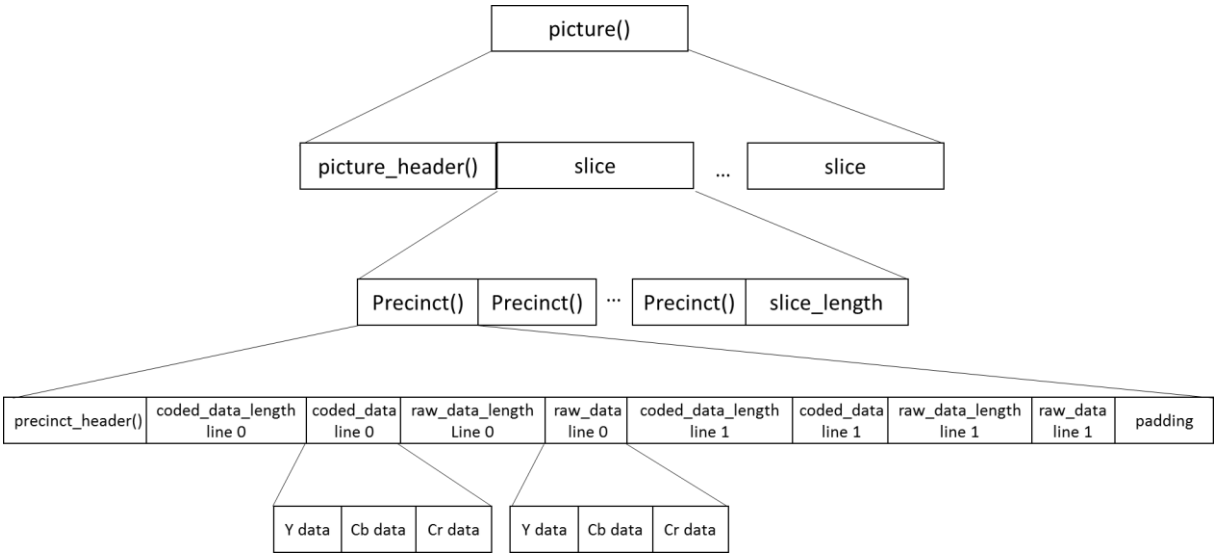

Figure 5 – TICO bitstream syntax structure hierarchy

3.4.2 Picture Syntax

Table 2 – Picture Syntax

Picture() {	vl(VL_P)	
picture_header()	vl(VL_PHD)	
k Slices() (k=ceil(PVS/SH)	vl(VL_SL)	

3.4.2.1 Picture Header Syntax

Table 3 – Picture Header Syntax

picture_header() {	vl(VL_PHD)	
picture_identifier	f(32)= ‘TICO’	
header_length	u(8)	PHDL
reserved	u(24)	
horizontal_size	u(16)	PHS
vertical_size	u(16)	PVS
color_difference _subsampling	u(4)	
bit_depth	u(4)	
reserved	u(24)	
encoder_identifier	u(32)	
bitstream_length	u(32)	BSL
bitstream_profile	u(4)	
Cbr	u(1)	
color_transform	u(3)	
hor_decomposition_number	u(4)	
ver_decompostion_number	u(4)	
dq_type	u(1)	
reserved	u(15)	
slice_height	u(16)	SLH
reserved	u(16)	
lvl_weights	u(16*36)	
reserved	u(192)	
reserved_optional	vl(VL_PHRO)	

3.4.3 Slice Syntax

Table 4 – Slice Syntax

Slice() {	vl(VL_S)	
Multiple Precincts	vl(VL_MP)	
slice_length	u(24)	SL

3.4.4 Precinct Syntax

Table 5 – Precinct Syntax

Precinct() {	vl(VL_P)	
precinct_header	u(40)	
coded_data_length_marker (line 0)	u(16)	CDL0
coded_data_packet (line 0) Components are scanned one after the other: comp0, comp1, comp2 Inside components, the horizontal decomposition levels are packed one after the other: Lvl0, Lvl1, Lvl2, Lvl3, Lvl4, Lvl5	vl(VL_CD0)	
coded_data padding	vl(VL_CDP0)	
raw_data_length_marker (line 0)	u(20)	RDL0
raw_data_packet (line 0) Components are packed one after the other: comp0, comp1, comp2 Inside components, the horizontal decomposition levels are packed one after the other: Lvl0, Lvl1, Lvl2, Lvl3, Lvl4, Lvl5	vl(VL_RD0)	
coded_data_length_marker (line 1)	u(16)	CDL1
coded_data_packet (line 1) Components are scanned one after the other: comp0, comp1, comp2 Inside components, the horizontal decomposition levels are packed one after the other: Lvl0, Lvl1, Lvl2, Lvl3, Lvl4, Lvl5	vl(VL_CD1)	
coded_data padding	vl(VL_CDP1)	
raw_data_length_marker (line 1)	u(20)	RDL1
raw_data_packet (line 1) Components are packed one after the other: comp0, comp1, comp2 Inside components, the horizontal decomposition levels are packed one after the other: Lvl0, Lvl1, Lvl2, Lvl3, Lvl4, Lvl5	vl(VL_RD1)	
precinct_padding	vl(VL_PP)	

3.4.4.1 Precinct Header Syntax

Table 6 – Precinct Header Syntax

Precinct_header()	u(40)	
precinct_length_marker	u(20)	PL
truncation_scenario	u(4)	
truncation_refinement	u(8)	
GCLI_coding_mode	u(4)	
reserved	u(4)	

3.5 Bitstream Semantics

3.5.1 Picture Header Semantics

picture_identifier

A four-character ASCII code that shall identify the bitstream as a TICO picture. This shall be ‘TICO’ value. Byte sequence is: 5449434F

header_length

An 8-bit integer that shall contain the length in bytes of the whole picture header. It is greater than or equal to 128. PHDL = (VL_PHD / 8). The length of the optional_reserved field can be calculated from this variable. VL_PHRO = 8*(PHDL – 128)

horizontal_size

horizontal_size shall be the width of the picture in luma samples.

vertical_size

vertical_size shall be the height of the picture in luma samples.

color_difference_subsampling

color_difference_subsampling shall specify the sampling format of the picture. Values and their meanings shall be as listed in Table 7.

Table 7 – Meaning of color_difference _subsampling

color_difference _subsampling	Meaning
0	4:2:2
1	4:4:4
2-15	Reserved

bit_depth

bit_depth shall specify the bit depth format of the picture. Values and their meanings shall be as listed in Table 8.

Table 8 – Meaning of bit_depth

bit_depth	Meaning
0	8
1	10
2	12
3-15	Reserved

encoder_identifier

A four-character ASCII code that shall identify the encoder vendor or product that generated the compressed frame. intoPIX maintains a registry of the codes for encoder licensees. Decoders should ignore this element.

bistream_length

A 32-bit word that shall contain the length in bytes of the bitstream. The picture header itself shall not be included in the bitstream length.

BSL = (VL_SL/8) = (VL_P – VL_PHD)/8

bitstream_profile

bitstream_profile shall be the TICO profile number of the bitstream. The profile number is incremented when a change is made to bitstream syntax or semantics that breaks compatibility with existing decoders. A decoder may abort if it encounters a bitstream with an unsupported bitstream_profile value. Values and their meanings shall be as listed in Table 9.

Table 9 – Meaning of bitstream_profile

bitstream_profile	Meaning
0	Unspecified/Unknown
1	Profile 1
2	Profile 2
3-15	Reserved

cbr

This field shall specify if the compressed bitstream was generated in CBR (Constant Bit Rate) mode, therefore using padding. Values and their meanings shall be as listed in Table 10

Table 10 – Meaning of cbr

cbr	Meaning
0	VBR (variable bit rate)
1	CBR (constant bit rate)

color_transform

This field shall specify if a color transform is applied on the image data. Values and their meanings shall be as listed in Table 11. color_transform value shall be set to 0 when color_difference _subsampling value is set to 0.

Table 11 – Meaning of color_transform

color_transform	Meaning
0	No color transform
1	Reversible color transform
2-7	Reserved

horizontal_decomposition_number

This field shall represent the number of wavelet decomposition applied in horizontal way. Values and their meanings shall be listed as in Table 12.

Table 12 – Meaning of horizontal_decompostion_number

horizontal_decomposition_number	Meaning
0-4	Reserved
5	5
6-15	Reserved

vertical_decomposition_number

This field shall represent the number of wavelet decomposition applied in vertical way. It shall be fixed to '1'.

dq_type

This field shall specify how the decoder has to dequantize the compressed data. Values and their meanings shall be listed as in Table 13.

Table 13 – Meaning of dq_type

dq_type	Meaning
0	Dequantizer with dead zone
1	Dequantizer with center rounding

slice_height

This field shall specify the number of lines in a slice. When bitstream_profile has a value of 1 or 2, slice_height shall have a value of 64.

lvl_weights

This field shall indicate the rate allocation strategies. It shall specify the different weight values per wavelet sub-bands. There shall be one weight per wavelet sub-band, they shall be provided for all horizontal and vertical wavelet sub-bands and for all components (so 6*2*3=36). Values shall be provided one after each other per component, then per vertical wavelet level, then per horizontal wavelet level. The first provided weight shall relate to the lowest horizontal level, lowest vertical level of component Y'. See Table 14 for ordering detail.

Table 14 – lvl_weights ordering

Index	0												11	12												23	24												35
Component	comp0													comp1													comp2												
Ver DWT Lvl	L	L	L	L	L	L	H	H	H	H	H	H	H	L	L	L	L	L	L	H	H	H	H	H	H	H	L	L	L	L	L	L	H	H	H	H	H	H	H
Hor DWT Lvl	L	1	2	3	4	H	L	1	2	3	4	H		L	1	2	3	4	H	L	1	2	3	4	H		L	1	2	3	4	H	L	1	2	3	4	H	

Each weight shall be 2-byte wide:

1) The msb byte shall correspond to the sub-band gain (lvl_gain). The most preserved sub-band have the highest gain. The lvl_gain shall range from 0 to 15.

2) The lsb byte shall specify the priority order of the rate allocation (lvl_priority). It gives more precision for rate allocation than using only gains. The priority values shall range from 0 to 35. The lowest value shall be for the most important sub-band to be added in the final bitstream.

3.5.2 Slice Semantics

slice_length

slice_length shall be a marker containing the length of the slice in number of bytes. SL = VL_S/8.

3.5.3 Precinct Semantics

coded_data_length_marker

coded_data_length_marker shall be a marker containing the length of the coded data (in number of 4-bit words). The value of the first marker of the precinct CDL0 shall be Ceil(VL_CD0/4). The value of the second marker of the precinct shall be Ceil(VL_CD1/4).

coded_data_padding

coded_data_padding shall be dummy bits to be discarded at the end of a coded data packet in order for the raw packet to start on a 4-bit aligned boundary. VL_CDP0 = 4*CDL0 - VL_CD0 and VL_CDP1 = 4*CDL1 – VL_CD1.

raw_data_length_marker

raw_data_length_marker shall be the length of the raw data (in number of 4-bit words). The value shall be VL_RD0/4 for first marker of the precinct and VL_RD1/4 for second marker.

precinct_padding

precinct_padding shall be dummy data inserted at the end of precinct for rate adjustment. Those bits shall be skipped by the decoder. The number of bits of this field may be computed from PL, CDL0, CDL1, RDL0 and RDL1.

VL_PP=4*PL – CDL0 – CDL1 – RDL0 – RDL1 – 82

3.5.4 Precinct Header Semantics

precinct_length_marker

precinct_length_marker shall be a marker containing the length of the precinct expressed in number of 4bit words. This length shall not include the 20 bits of the length marker itself. It shall include potential “end of precinct” padding. PL = VL_P - 20

truncation_scenario and truncation_refinement

The scenario field (4 bits) and the refinement field (8 bits) shall be the output of the TICO encoder rate allocation process. Those fields from the precinct header, together with lvl_weights from the picture header shall be used to calculate the GTLIs for the 36 sub-bands, both for coded and raw data. See Section 3.6.2.3 (Truncation level calculation).

GCLI_coding_mode

This field shall specify how the GCLI are encoded for the precinct. Different techniques are used for the encoding of GCLI. Values and their meanings shall be listed as in Table 15.

Table 15 – Meaning of GCLI_coding_mode

GCLI_coding_mode	Meaning
0	RAW
1	reserved
2	HOR (horizontal prediction)
3	VER (vertical prediction)

3.6 Decoding Process

This section describes the process that a decoder shall follow to reconstruct a picture from a TICO profile 2 bitstream.

3.6.1 Decoding Summary

3.6.1.1 Decoding Sequence Schematic

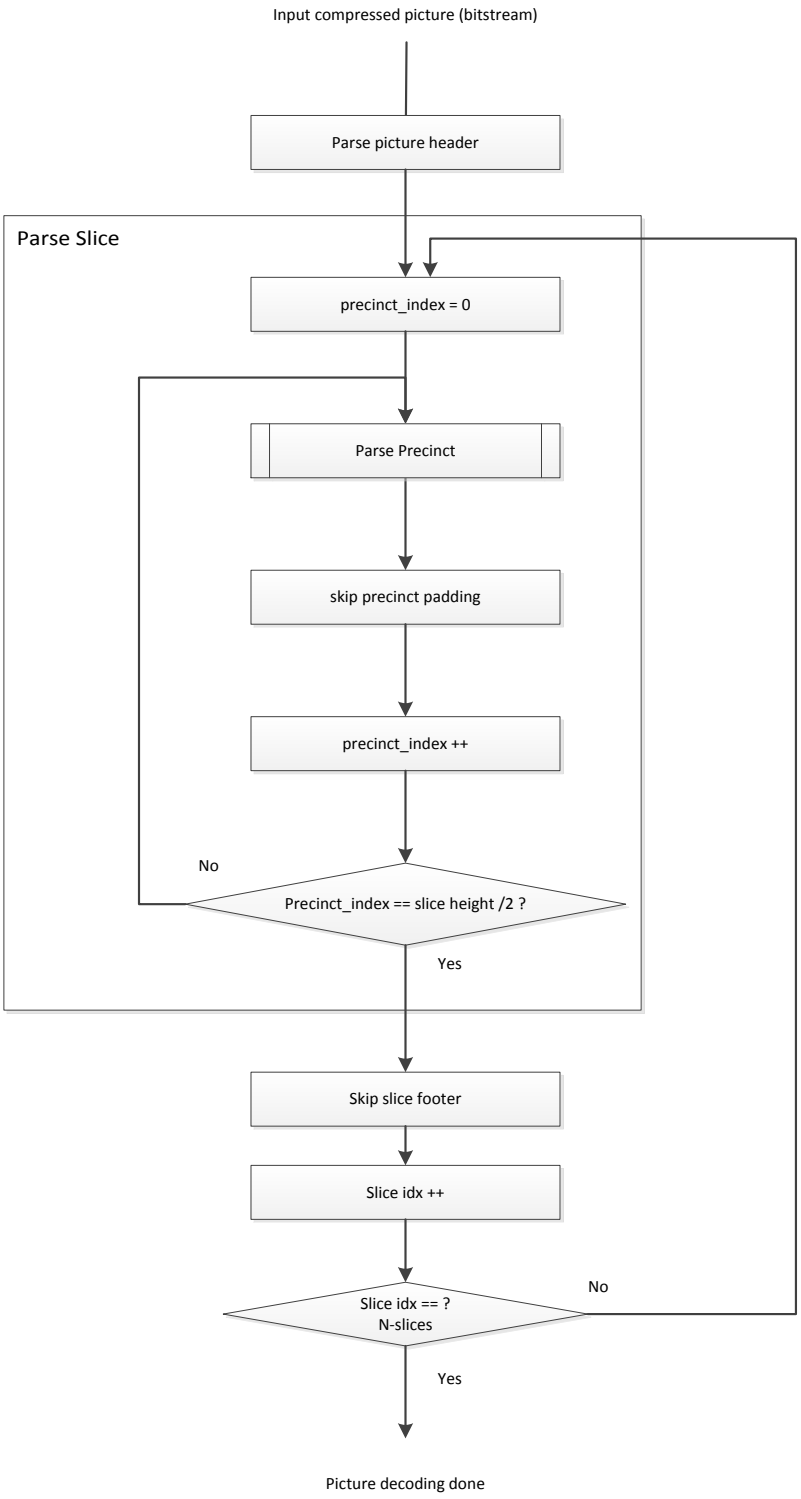

Figure 6 – TICO Decoding sequence

3.6.1.2 Picture Header Decoding

The picture header shall be read in raw from the bitstream. All picture header members shall be kept for further decoding steps.

3.6.1.3 Slices Iteration

Picture decoding may be performed slice after slice. Slices may also be decoded simultaneously if decoding speed is of concern.

The number of slices to decode may be computed from image_height and slice_height parameters found in the picture header.

$$n\_slices = ceil(PVS/PSH)$$

3.6.1.4 Precincts Iteration

Slice decoding shall be performed precinct after precinct. Since a precinct contains compressed data from 2 uncompressed lines, the number of precincts in a slice may be calculated using the following formula:

$$n\_precincts = PSH/2$$

The last slice may contain a smaller number of precincts if the image_height is not an exact multiple of slice_height:

$$n\_precincts(last\_slice) = (PVS \bmod PSH) / 2$$

3.6.1.5 Precinct Decoding

Precinct decoding is the core process of TICO decoding. Details of this process is described in Section 3.6.2.

3.6.1.6 Precinct Padding Management

After the decoding of each precinct, the number of padding bits shall be skipped. To know the amount of 4-bit words of padding, the number of consumed words shall be subtracted from PL value derived from precinct_length_marker . The read cursor shall be moved after those words before decoding the next precinct. The padding of the last precincts of a slice always ensures the next slice starts on an 8-bit boundary.

3.6.1.7 Slice Footer Management

After the last precinct of a slice, the decoder may assume the read cursor is aligned to an 8-bit boundary.

The slice length value (SL) shall be read (3 bytes read in big endian). The value may be compared to the number of bytes read from the slice (including those 3 bytes) to detect potential bitstream corruption.

3.6.2 Precinct Decoding

3.6.2.1 Summary

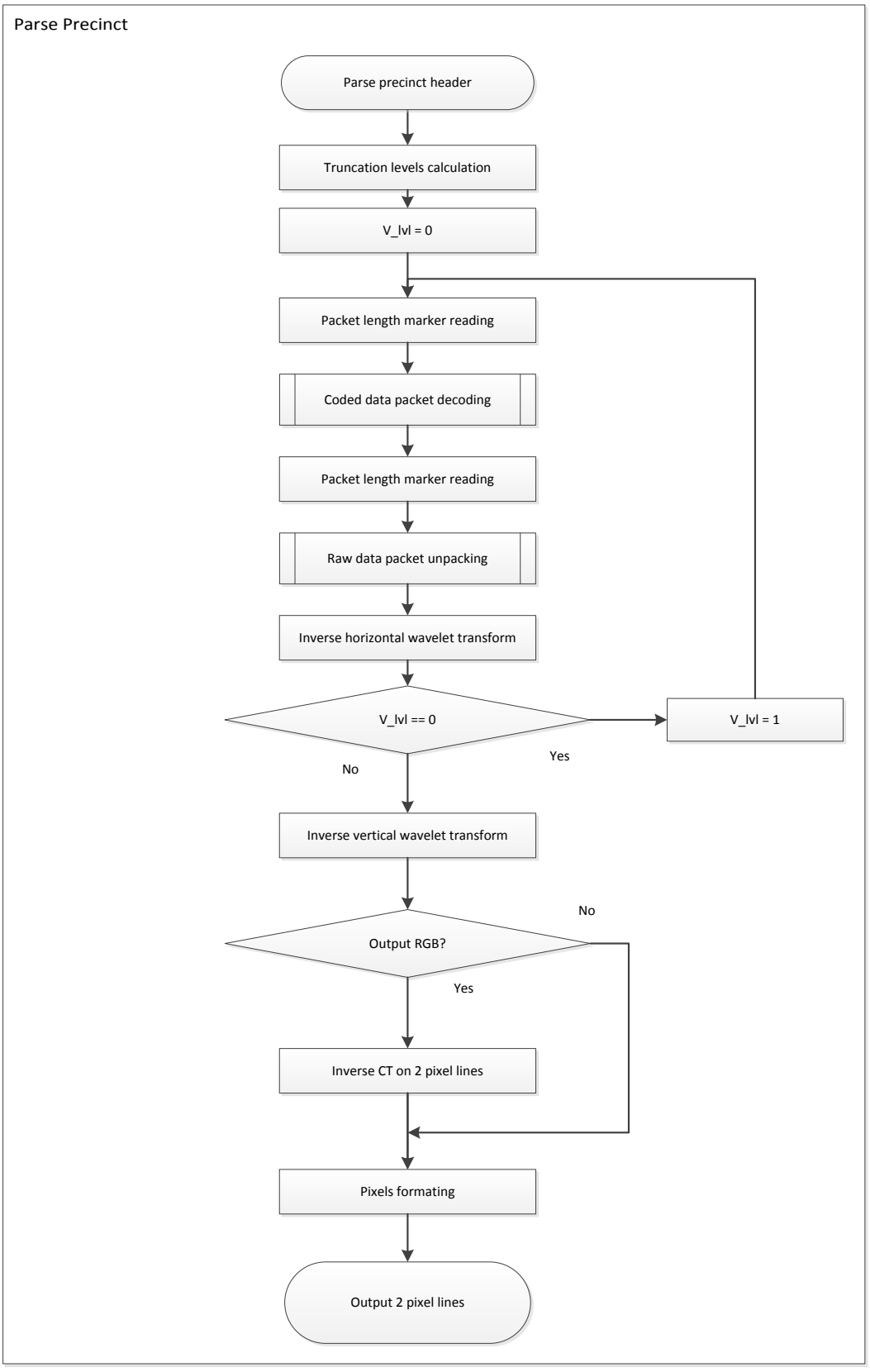

Figure 7 – Parse Precinct sequence

3.6.2.2 Precinct Header Parsing

The header of the precinct shall be read in raw from the bitstream.

Precinct_length shall be kept for precinct_padding computation.

truncation_scenario and truncation_refinement shall be provided to the next step (truncation level calculation). GCLI_coding_mode value shall be kept for the “coded data packet decoding” step.

3.6.2.3 Truncation Level Calculation

Arrays of $GTLI$ and $GTLI_{noref}$ shall be computed with values for each of the 36 sub-bands. This step shall use truncation_scenario,truncation_refinement from precinct header and lvl_weights (lvl_gain, lvl_priority pairs) from picture header. Sub-band number is called sb in following calculation. Calculation shall be performed in two steps. The second step shall be clamping in range [0,15] of the $GTLI$ and $GTLI_{noref}$ values.

$$GTLI_{noref}[sb] = truncationscenario - lvlgains[sb]$$

$$GTLI[sb] = \begin{cases} GTLI_{noref}[sb] - 1, & lvlpriority[sb] < truncationrefinement \\ GTLI_{noref}[sb], & lvlpriority[sb] \geq truncationrefinement \end{cases}$$

$$GTLI_{noref}[sb] = clamp(GTLI_{noref}[sb], 0, 15)$$

$$GTLI[sb] = clamp(GTLI[sb], 0, 15)$$

Those arrays shall be kept for the coded packet decoding ($GTLI_{noref}$ array) and for the raw data unpacking ($GTLI$ array).

3.6.2.4 Loop on Vertical Wavelet Levels

For both vertical wavelet levels (low frequency level and high frequency level), the same 3 operations shall be performed. Coded data packet decoding, raw data packet unpacking and inverse horizontal wavelet transform. After those steps, data from those two frequencies shall be processed by the inverse vertical wavelet transform.

3.6.2.5 Packet and Sub-band Orders.

The precinct shall contain a total of 4 packets. For the two vertical sub-bands there shall be one coded data packet and one raw data packet. Those packets shall contain data from all components and horizontal wavelet sub-bands. The number of sub-bands per packet shall be:

$$packet\_sb\_number = 3 * (hor\_decomposition\_number + 1)$$

In all the decoding loops, the order of the sub-bands shall always be the same (see table 16): within a component, sub-bands shall be ordered per horizontal wavelet level from lowest frequency to highest.

Table 16 – Sub-band order

Sb	0 5	6 11	12 17
Component hor DWT lvl	Comp0 L 1 2 3 4 H	Comp1 L 1 2 3 4 H	Comp2 L 1 2 3 4 H

3.6.2.6 Packet Length Marker Reading

The coded data packets and raw data packets shall be preceded by a length marker. This length marker shall be 16-bit wide for coded data packets, and 20-bit wide for raw data packets. Before reading those markers, the read cursor shall skip any coded data padding or raw data padding bits before the next syntax element until it is aligned to a 4-bit boundary of the bitstream. The values shall then be read as unsigned integer in the bitstream. The first bit read is the MSB of the length value, the last is the LSB. The length is expressed in number of 4-bit words. This length shall not include the length marker itself. Those markers may be used to detect potential corruption in bitstream.

3.6.2.7 Coded Packets Decoding

3.6.2.7.1 Overview

In this process, all the GCLIs of all the sub-bands present in the packet shall be decoded and provided to the RAW packet unpacking process. Decoding process depends on GCLI_coding_mode information provided in precinct header.

The GCLIs are the “Greatest Coded Line Indexes” of a group of 4 coefficients. There is one GCLI per 4 coefficients of a sub-band. Please refer to section RAW data packet unpacking for information about GCLI usage.

$$sb\_GCLI\_number(sb) = \lceil sb\_width(sb)/4 \rceil$$

Please refer to section “inverse wavelet transform” to know how sb_width can be computed from image format.

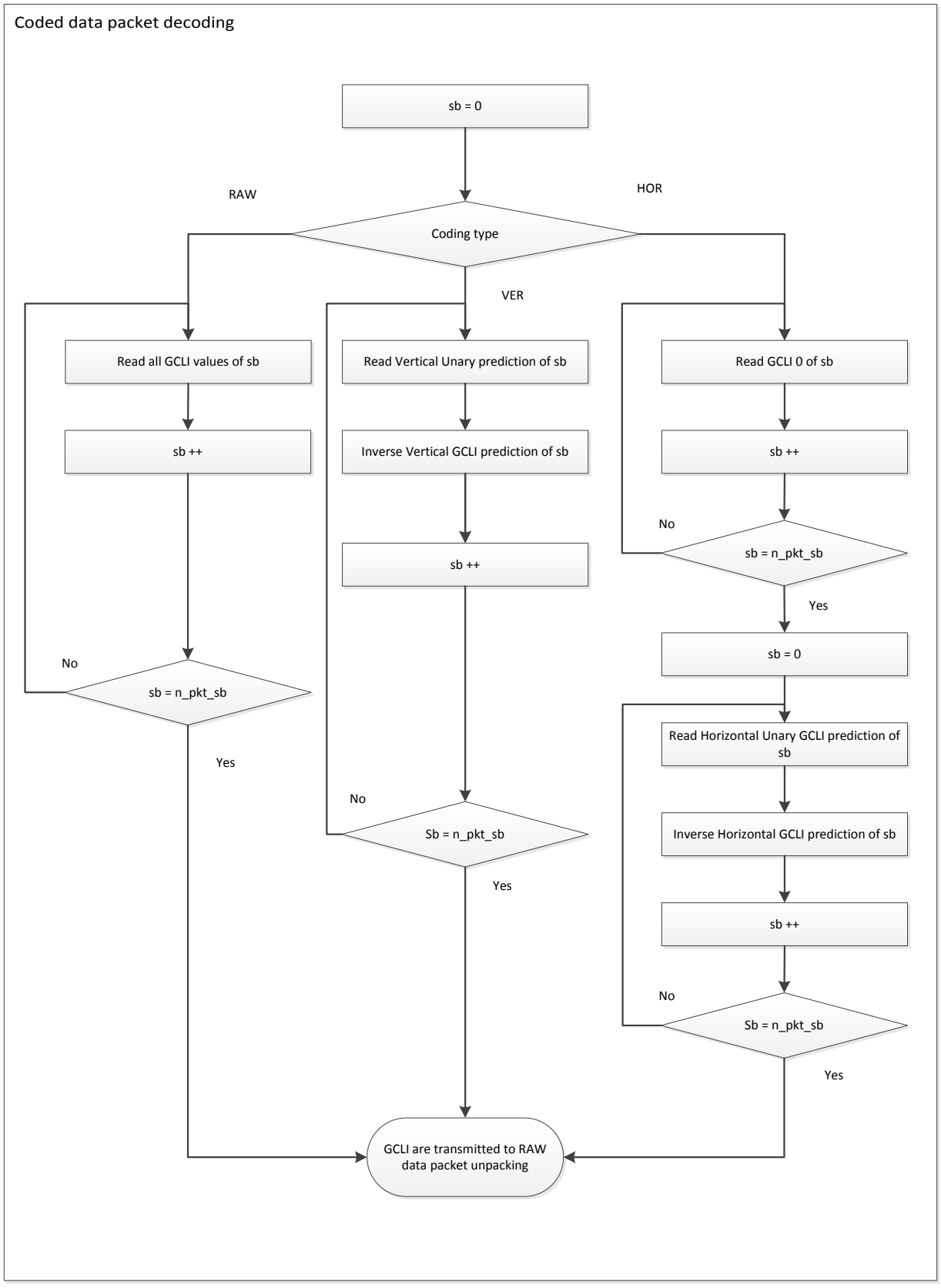

Figure 8 – Coded data packet decoding

3.6.2.7.2 RAW GCLIs Reading (GCLI_coding_mode RAW)

If the GCLI_coding_mode value is 'RAW', the GCLIs shall be read in raw from the bitstream as 4-bit unsigned integers. In total, this process shall read a number of 4-bit values equal to the sum of $sb\_gcli\_number(sb)$ for all sub-bands.

3.6.2.7.3 Read of First GCLIs (GCLI_coding_mode HOR)

If GCLI_coding_mode is HOR, first operation shall be to read the first GCLI of each sub-band in raw as 4-bit unsigned integers. In total, this process shall read a number of 4-bit values equal to the number of sub-bands in the packet ($packet\_sb\_number$).

3.6.2.7.4 Unary Coded Values Reading (GCLI_coding_mode HOR and VER)

In HOR and VER GCLI_coding_mode, a list of delta-values shall be read from the bitstream. Those values are signed integers encoded using unary coding. The absolute value of the coded value is the number of 0 read before first 1. When not zero, a sign bit follows that first 1. As maximal absolute value is 15, for this special value, the sign bit immediately follows the sequence of zeros.

Table 17 – Unary coded values reading

Value	Unary Code
-15	0000 0000 0000 0001
…	…
-2	0011
-1	011
0	1
1	010
2	0010
…	…
15	0000 0000 0000 0000

The delta-values shall be read for all the sub-band of the packet. The number of delta-values to be read per sub-band shall be equal to $sb\_GCLI\_number(sb)$ in case GCLI_coding_mode is VER or $sb\_GCLI\_number(sb) - 1$ in case GCLI_coding_mode is HOR.

3.6.2.7.5 Inverse Horizontal GCLI Prediction (GCLI_coding_mode HOR)

The inverse horizontal prediction step shall reconstitute all the GCLIs of the sub-band from unary decoded deltas-values:

$$GCLI_{truncated}[sb][i+1] = GCLI_{truncated}[sb][i] + \mathrm{delta[sb][i]}$$

Note that the $GCLI_{truncated}[sb][0]$ values are known from "first GCLIs reading" earlier step.

The final (not truncated) GCLIs shall be derived from $GCLI_{truncated}$ values and $GTLI_{noref}$ value for the sub-band:

$$GCLI[sb][i] = \begin{cases} GCLI_{truncated}[sb][i] + GTLI_{noref}[sb], & GCLI_{truncated}[sb][i] > 0 \\ 0, & GCLI_{truncated}[sb][i] = 0 \end{cases}$$

3.6.2.7.6 Inverse Vertical GCLI Prediction (GCLI_coding_mode VER)

The inverse vertical prediction step shall reconstitute all the GCLIs of the sub-band based on:

- $\mathrm{delta}[\mathrm{sb}][x]$: unary decoded deltas-values for this sub-band
- $GTLI_{noref}[sb]$: value for this precinct.
- $GCLI'\ [sb][x]$: values of GCLIs from previous precinct.
- $GTLI'_{noref}[sb]$: value of GTLI from previous precinct.

The decoder may assume the GCLI_coding_mode VER is never used for the first precinct of the picture. It may also assume it is never used for the first precinct of slices since slices shall be completely independent.

The following calculation shall be performed to reconstitute the GCLIs:

$$GTLI_{worst}[sb] = max(GTLI_{noref}[sb], GTLIi'_{noref}[sb])$$
$$GCLI_{truncated}[sb][i] = max\big(GCLI'\ [sb][i], GTLI_{worst}[sb]\big) + delta[sb][i]$$

As for inverse horizontal prediction, the final (not truncated) GCLIs shall be derived from $GCLI_{truncated}$values and $GTLI_{noref}$ value for the sub-band:

$$GCLI[sb][i] = \begin{cases} GCLI_{truncated}[sb][i] + GTLI_{noref}[sb], & GCLI_{truncated}[sb][i] > 0 \\ 0, & GCLI_{truncated}[sb][i] = 0 \end{cases}$$

3.6.2.8 Raw Data Packet Unpacking

3.6.2.8.1 Summary

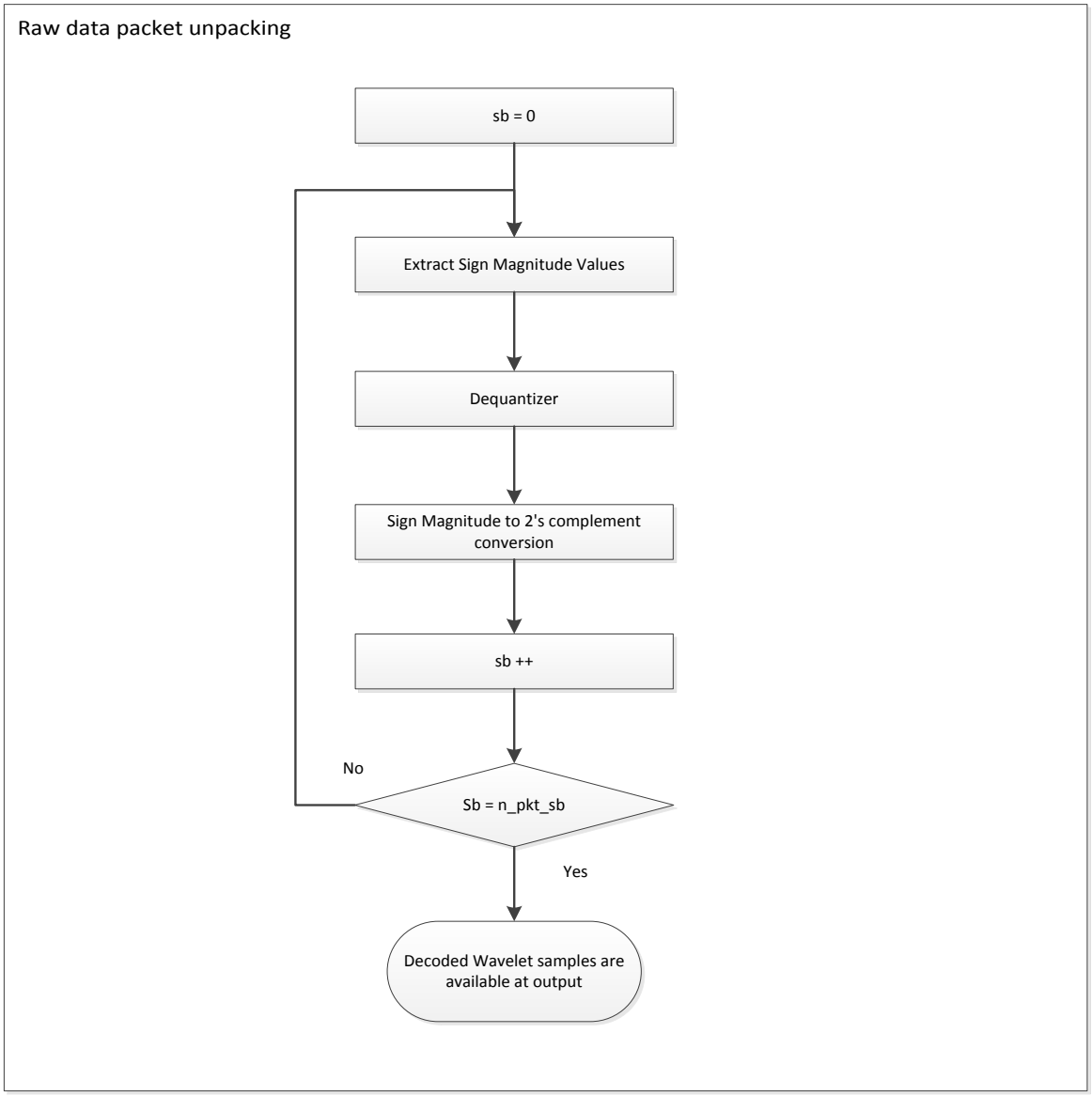

Figure 9 – Raw data packet unpacking

3.6.2.8.2 Sign Magnitude Coefficient Extraction

During this phase, all coefficients of the sub-band shall be extracted by groups of 4. The extraction of each group shall use one GCLI decoded earlier and the GTLI of the sub-band.
The first step shall initialize all coefficients of the group to 0:

$$sm[0] = sm[1] = sm[2] = sm[3] = 0$$

Then the number of bit-planes to read shall be computed using the following formula:

$$nbp_{toread} = \begin{cases} 0, & GCLI \leq GTLI \\ GCLI - GTLI + 1, & GCLI > GTLI \end{cases}$$

If nbp_{toread} is 0, the read cursor shall keep the same position and the coefficients shall be left with value 0.

Otherwise, nbp_{toread} groups of 4 bits shall be read from bitstream.

The first 4 bits shall correspond to the signs of the 4 coefficients (first bit for first coefficient, last bit for the fourth one). The next 4-bit groups shall correspond to the bits at indexes starting from GCLI-1 down to GTLI in the magnitude of the 4 coefficients.

3.6.2.8.3 Dequantizer and Rounding

Figure 11 illustrates an example of coefficients reconstitution. Example is taken with GTLI=2 and two groups, one with GCLI=5, the next one with GCLI=4.

group 0 (nbptoread=4)										group 1 (nbptoread=3)										
0	0	1	0	1	1	0	0	1	1	1	0	0	1	0	1	1	0	0	1	0	1	1	1	0	0	0	1	…
Signs				Bits4				Bits3				Bits2				Signs				Bits3				Bits2				
GCLI = 5																GCLI = 4												

Figure 10 – Example of data located in bitstream

Sign	0	0	1	0	1	0	0	1
14	0	0	0	0	0	0	0	0
…	0	0	0	0	0	0	0	0
4	1	1	0	0	0	0	0	0
3	1	1	1	0	0	1	1	1
2	0	1	0	1	0	0	0	1
1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
sign	+	+	-	+	-	+	+	-
magnitude	0x18	0x1C	0x08	0x04	0x0	0x08	0x08	0xC

Figure 11 – Example of coefficients reconstitution

The reconstitution of coefficient shall perform the inverse quantization.

- Dead zone dequantization

When dq_type equals to 0, after grouping all coefficient bit-planes, all recombined values shall be left shifted by GTLI value. The reconstructed value is called hereafter mag (for magnitude). Magnitude shall be in the range [0, 32767].

When the GTLI of the sub-band is bigger than 0, a rounding shall be performed on magnitudes of the non-null samples. This corresponds to a dequantizer with dead zone.

$$mag := \begin{cases} mag, & mag = 0 \; or \; GTLI = 0 \\ mag + (1 \ll GTLI)/2, & mag > 0 \; and \; GTLI > 0 \end{cases}$$

- Center-rounding dequantization

When dq_type equals to 1, after grouping all coefficient bit-planes, the following mathematical operation shall be performed on the recombined values to reconstruct the magnitude of the samples. The magnitudes shall be in the range [0, 32767]:

$$dq_{val} = \frac{1 \ll (GCLI + 1)}{1 \ll (GCLI - GTLI + 1) - 1}$$
$$mag \coloneqq \lfloor mag * dq_{val} \rfloor$$

In either case, if the number of coefficients of the sub-band is not a multiple of 4, the unnecessary coefficients generated from the last GCLI shall be skipped and not transmitted to further steps.

At the end of this step, the sign and the magnitude of the coefficients shall be combined and provided to the next step. Note: the decoder may not assume 0-decoded magnitudes have a 0 sign bit since an encoder may want to transmit the sign of the coefficient before quantization.

3.6.2.8.4 Conversion Sign Magnitude to 2's Complement

All samples of the sub-band shall be converted from their sign magnitude representation to a 2's complement representation more adequate for further arithmetic operations.

3.6.2.9 Inverse Wavelet transform

3.6.2.9.1 Horizontal Wavelet Levels

TICO encoder shall apply consecutive 1D forward 5/3 wavelet transforms on lines of samples. It first shall decompose the original samples in LF (low frequency) and HF (high frequency) sub-bands. The number of samples in the LF/HF sub-band after one decomposition shall be the original number of samples divided by 2 with a ceil/floor operation respectively.

$$size_{lf} = \left\lceil \frac{size}{2} \right\rceil$$
$$size_{hf} = \left\lfloor \frac{size}{2} \right\rfloor$$

The resulting low frequency samples shall be then decomposed again in a LF and an HF sub-bands. The number of decompositions chosen by encoder shall be provided to decoder in the picture header (hor_decomposition_nb).

If the number of decomposition is 5, HF sub-band of the first decomposition is called Lvl5 and LF sub-band of the last decomposition is called Lvl0. To compute the number of samples in each sub-bands from Lvl5 to Lvl0, consecutive divisions by 2 shall be applied on the image component sizes. Table 18 shows the sub-band sizes and number of GCLIs per sub-band in the bitstream for typical HDTV and UHDTV1 resolutions.

Table 18 – Sub-band sizes and number of GCLIs per sub-band in the bitstream for HDTV and UHDTV1

			Number of samples in sub-bands						Number of GCLIs in sub-bands					
Format	Comp	Width	Lvl0	Lvl1	Lvl2	Lvl3	Lvl4	Lvl5	Lvl0	Lvl1	Lvl2	Lvl3	Lvl4	Lvl5
HDTV 422	Y'	1920	60	60	120	240	480	960	15	15	30	60	120	240
	$C'_BC'_R$	960	30	30	60	120	240	480	8	8	15	30	60	120
HDTV 444	R,G,B	1920	60	60	120	240	480	960	15	15	30	60	120	240
UHDTV1 444	R,G,B	3840	120	120	240	480	960	1920	30	30	60	120	240	480
UHDTV1 422	Y'	3840	120	120	240	480	960	1920	30	30	60	120	240	480
	$C'_BC'_R$	1920	60	60	120	240	480	960	15	15	30	60	120	240

3.6.2.9.2 Inverse Horizontal Wavelet Transform

The decoder shall execute consecutive inverse 5/3 wavelet transforms on the samples of sub-bands. First inverse transform shall be applied on Lvl0 and Lvl1 sub-band pair. Second on the output of the first inverse transform and Lvl2 and so on.

To do the inverse transform, a lifting scheme implementation may be performed. The two lifting steps of the inverse transform, which recover the original data, are defined as:

$$s'_i = s_i - (d_{i-1} + d_i + 2) \gg 2$$

$$d'_i = d_i + (s'_i + s'_{i+1}) \gg 1$$

The *s/d* values are the coefficient of low/high frequency. The *s'/d'* values are even/odd index part of the inverse wavelet output coefficients.

When indexes of coefficient are out of the boundaries, the first/last coefficient shall be duplicated as shown in Figure 12.

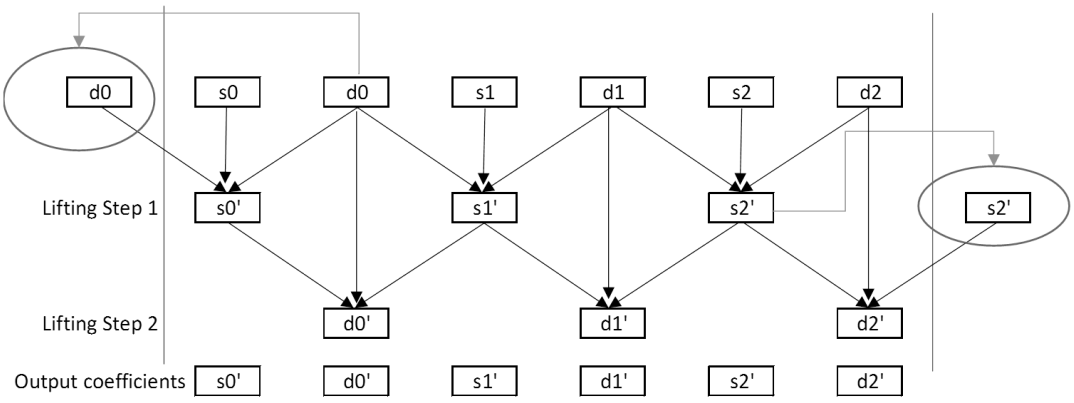

Figure 12 – Inverse Wavelet Transform (IWT)

3.6.2.9.3 Inverse Vertical Wavelet Transform

This step shall apply an inverse HAAR transformation on the two vertical frequencies data (l_{lf}, l_{hf}) to produce two lines of pixels (l_0, l_1).

$$l_0[comp][x] = (2l_{lf}[comp][x] + l_{hf}[comp][x] + 1) \gg 1$$
$$l_1[comp][x] = (2l_{lf}[comp][x] - l_{hf}[comp][x] + 1) \gg 1$$

This transformation shall be performed for all coefficients of the three components, one after each other ($comp = 0 \rightarrow 2, x = 0 \rightarrow image\_width[comp]$).

3.6.2.9.4 Inverse Color Transform

An inverse color transform shall be applied when the color_transform field in the picture header differs from 0. When this field equals to 1, the following inverse reversible transform shall be applied to recover the original data:

$$comp0' = comp0 - \lfloor \frac{comp1 + comp2}{4} \rfloor \quad comp2' = comp1 + comp0' \quad comp1' = comp2 + comp0'$$

3.6.2.9.5 Pixels Formatting

The signed pixel values shall be converted to unsigned values. To do that, a shift of (1 << (bit_depth-1)) shall be performed on all the output samples. Resulting samples shall be clamped to initial interval [0, (1<<bit_depth)-1].

3.6.2.9.6 Outputting 2 Lines of Pixels

The pixel values for both lines shall be made available at output of decoder.

4 TICO Bitstream Mapping

4.1 Video Essence Box

4.1.1 Introduction

TICO Video_essence_box() provides additional information complementary to the TICO picture header. It characterizes the video essence compressed by TICO and is part of a compliant TICO video stream. The Video_essence_box() shall be used for both SDI and RTP mapping. There shall be one Video_essence_box() per TICO picture().

4.1.2 Video Essence Box Syntax

The Table 19 defines the syntax variables of the video essence box with their category as described in Table 1 in Section 3.4.1 and their alias.

Table 19 – TICO Video Essence Box Syntax

Video_essence_box() {	vl(VL_TVEB)	
video_essence_box_identifier	f(32)='TVEB'	
length	u(8)	TVEBL
bitstream_profile	u(4)	
frame_rate	u(6)	
interlace_mode	u(2)	
sample_bitdepth	u(4)	
sampling_structure	u(4)	
reserved	u(4)	
horizontal_size	u(16)	
vertical_size	u(16)	
color_primaries	u(8)	
transfer_characteristic	u(6)	
matrix_coefficients	u(6)	
reserved	u(12)	
source_type	u(4)	
video input_info	u(6)	
audio_input_info	u(6)	
VBI_input_info	u(6)	
reserved	u(42)	
reserved_optional	vl(VL_TVEBRO)	

4.1.3 Video Essence Box Semantics

video_essence_box_identifier

A four-character ASCII code that identifies the TICO video essence box of a TICO picture. It shall be ‘TVEB’ value.

length

length shall contain the number of 64-bit words present in video essence box. Minimum value shall be 3. TVEBL = VL_TVEB/64 = (192 + VL_TVEBRO) / 64

bitstream_profile

It is the profile number of the bitstream. This field shall be the same as defined in the picture() header.

horizontal_size

horizontal_size shall specify the width of the frame in luma samples.

vertical_size

vertical_size shall specify the height of the frame in luma samples.

sample_bitdepth

sample_bitdepth code shall specify the bit depth of the sample quantization. Values and their meanings shall be listed as in the Table 20.

Table 20 – Meaning of sample_bitdepth

sample_bitdepth	Meaning
0	8 bits per sample
1	10 bits per sample
2	12 bits per sample
3-15	Reserved

interlace_mode

interlace_mode code shall specify whether the original picture is progressive or interlaced (as well as field order in the latter case). Values and their meanings shall be listed as in the Table 21.

Table 21 – Meaning of interlace_mode

interlace_mode	Meaning
0	Progressive frame (frame contains one full-height picture)
1	Interlaced frame (picture is first field)
2	Interlaced frame (picture is second field)
3	Reserved

frame_rate

frame_rate code shall indicate frame rate. Values and their meanings shall be listed as in the table. The Table 22 is consistent with the Picture rate table defined in SMPTE ST 352 document.

Table 22 – Meaning of frame_rate

frame_rate	Meaning
0	No defined value
1	Reserved
2	24 ÷ 1.001 (23.976…)
3	24
4	48/1.001
5	25
6	30 ÷ 1.001 (29.97…)
7	30
8	48
9	50
10	60 ÷ 1.001 (59.94…)
11	60
12	100
13	120 ÷ 1.001 (119.88…)
14	120
15-63	Reserved

sampling_structure

sampling_structure code shall indicate the sampling structure of the picture. Values and their meanings shall be listed as in the Table 23. This table is coherent with the sampling structure defined in SMPTE ST 352 document.

Table 23 – Meaning of sampling_structure

sampling_structure	Meaning
0	4:2:2 ($Y'C'_BC'_R$)
1	4:4:4 ($Y'C'_BC'_R$)
2	4:4:4 (G/B/R)
3-15	Reserved

color_primaries

color_primaries code shall indicate the source color primaries. The chromaticities of the primaries and white point for each code value shall be as stated in the CIE 1931 xy chromaticity column in Table 24.

Table 24 – Meaning of color_primaries

color_primaries	CIE 1931 xy chromaticity			Note
0	Unknown/unspecified			
1		x	y	corresponds to ITU-R BT.601-7 with a D65 white point
	Red	0.6400	0.3300	
	Green	0.2900	0.6000	
	Blue	0.1500	0.0600	
	White	0.3127	0.3290	
2		x	y	corresponds to ITU-R BT.709-6 with a D65 white point
	Red	0.6400	0.3300	
	Green	0.3000	0.6000	
	Blue	0.1500	0.0600	
	White	0.3127	0.3290	
6		x	y	corresponds to ITU-R BT.2020-2 with a D65 white point
	Red	0.7080	0.2920	
	Green	0.1700	0.7970	
	Blue	0.1310	0.0460	
	White	0.3127	0.3290	
7	Reserved			
8	Reserved			
9		x	y	corresponds to SMPTE RP 431-2, with a DCI white point
	Red	0.6800	0.3200	
	Green	0.2650	0.6900	
	Blue	0.1500	0.0600	
	White	0.3140	0.3510	
10		x	y	corresponds to SMPTE RP 431-2, with a D65 white point
	Red	0.6800	0.3200	
	Green	0.2650	0.6900	
	Blue	0.1500	0.0600	
	White	0.3127	0.3290	
11	Reserved			
12	Reserved			
13		x	y	corresponds to sRGB with a D65 white point
	Red	0.6400	0.3300	
	Green	0.3000	0.6000	
	Blue	0.1500	0.0600	
	White	0.3127	0.3290	
14 – 255	Reserved			

transfer_characteristic

transfer_characteristic code shall indicate the opto-electronic transfer characteristic of the source video data. Value and their meanings shall be listed as in Table 25.

Table 25 – Meaning of transfer_characteristic

transfer_characteristic	Meaning
0	Unknown/unspecified
1	Signifies the opto-electronic transfer function specified by ITU-R BT.601/BT.709-6/BT.2020-2
2-15	Reserved
16	Signifies the Inverse-EOTF function in Section 5 of SMPTE ST 2084:2014
17-63	Reserved

matrix_coefficients

matrix_coefficients code shall indicate the matrix coefficients used to derive luma and color difference values from the red, green, and blue primaries.

For values that specify luma coefficients (KR, KG, and KB), the derivation is E′Y’ = KR * E′R + KG * E′G + KB * E′B, $E'C'_B$ = (E′B − E′Y) ÷ (2 * (1 − KB)), and E′ C'_R = (E′R − E′Y’) ÷ (2 * (1 − KR)), where E′Y is the normalized luma value and $E'C'_B$ and $E'C'_R$ are the normalized color difference values corresponding to the normalized gamma pre-corrected primary values E′R, E′G, and E′B.

In BT.2020-2, this derivation implies the use of Non Constant Luma signal format.

The matrix coefficients for each code value shall be as stated in the meaning column as in Table 26.

Table 26 – Meaning of matrix_coefficients

matrix_coefficients	Meaning	Note
0	Unknown/unspecified	
1	K_R = 0.2126, K_G = 0.7152, K_B = 0.0722	corresponds to ITU-R BT.709-6
2	K_R = 0.299, K_G = 0.587, K_B = 0.114	corresponds to ITU-R BT.601
3	K_R = 0.2627, K_G = 0.6780, K_B = 0.0593	corresponds to ITU-R BT.2020-2
4 - 63	Reserved	

source_type

source_type shall be indicating the type of video source. Values and their meanings shall be listed as in Table 27.

Table 27 – Meaning of source_type

source_type	Meaning
0	Unknown
1	SDI
2	Native
3-16	Reserved

video_input_info

video_input_info code shall be indicating the video source so that TICO enabled decoder/receiver knows how the source was presented to the TICO encoder/transmitter. This field shall be linked to the value of the source_type field. This is for enabling the decoder/receiver to output the very same format when set to 'Auto' operation.

Example if source_type is 1 (SDI):

TICO transmitter is fed with a Square Division (SQD) Quad Split format, the Receiver can output in either SQD or 2SI (2-sample interleave) format. In 'Auto' mode, following the example, the Receiver would output SQD. The operator can also force the output format to whatever format is supported by the Receiver. It is up to the Receiver to use this information or to simply ignore it.

Values and their meanings shall be listed as in Table 28, Table 29, and Table 30.

Table 28 – Meaning of video_input_info if source_type =0 (unknown)

video_input_info	Meaning
0	Source unknown
1-63	Reserved

Table 29 – Meaning of video_input_info if source_type = 1 (SDI)

video_input_info	Meaning
0	Source unknown
1	SDI Source SQD, Level A mapping of sub images
2	SDI Source SQD, Level B mapping of sub images
3	SDI Source 2SI, Level A mapping of sub images (SMPTE ST 425-5)
4	SDI Source 2SI, Level B mapping of sub images (SMPTE ST 425-5)
5	SDI Source Single Link 12G(SMPTE ST 2082-10)
6	SDI Source Single Link 6G (SMPTE ST 2081-10)
7	SDI Source Dual Link 3G (SMPTE ST 425-3)
8	SDI Source Quad Link 1.5G
9-63	Reserved

Table 30 – Meaning of video_input_info if source_type = 2 (Native*)*

video_input_info	Meaning
0	Source unknown
1-63	Reserved

audio_input_info

audio_input_info code shall be describing the audio mapping from the source_type. This field shall be linked to the value of the source_type field.

As an example with a SDI source_type, it might be useful for some Receivers to know from which of the four sub-images (SQD/2SI) the audio data has been extracted for transmission (i.e. transmission together with the compressed bitstream via a single 3G-SDI stream).

Values and their meanings shall be listed as in Table 31, Table 32, and Table 33.

Table 31 – Meaning of audio_input_info if source_type = 0 (unknown*)*

audio_input_info	Meaning
0	Unknown
1-63	Reserved

Table 32 – Meaning of audio_input_info if source_type = 1 (SDI)

audio_input_info	Meaning
0	No audio data
1	16 Channels, from Link #1
2	16 Channels, from Link #2
3	16 Channels, from Link #3
4	16 Channels, from Link #4
5	32 Channels, from Link #1
6	32 Channels, from Link #2
7	32 Channels, from Link #3
8	32 Channels, from Link #4
9	16 Channels, from Link #1 + 16 Channels from Link #2
10	16 Channels, from Link #1 + 16 Channels from Link #3
11	16 Channels, from Link #1 + 16 Channels from Link #4
12	16 Channels, from Link #2 + 16 Channels from Link #3
13	16 Channels, from Link #2 + 16 Channels from Link #4
14	16 Channels, from Link #3 + 16 Channels from Link #4
15-63	Reserved

Table 33 – Meaning of audio_input_info if source_type = 2 (Native*)*

audio_input_info	Meaning
0	Unknown
1-63	Reserved

VBI_input_info

VBI_input_info shall be describing the VBI data origin to the receiving device. This field shall be linked to the value of the source_type field.

Values and their meanings are listed in Table 34, Table 35, and Table 36.

Table 34 – Meaning of _VBI_input_info if source_type = 0 (unknown*)*

VBI_input_info	Meaning
0	Unknown
1-63	Reserved

Table 35 – Meaning of _VBI_input_info if source_type = 1 (SDI)

VBI_input_info	Meaning
0	No VBI data
1	VBI from Link #1
2	VBI from Link #2
3	VBI from Link #3
4	VBI from Link #4
5-63	Reserved

Table 36 – Meaning of _VBI_input_info if source_type = 2 (Native*)*

VBI_input_info	Meaning
0	Unknown
1-63	Reserved

4.2 TICO Mapping to Active Video Area of 3G-SDI

4.2.1 Overview

This section defines how TICO bitstream shall be mapped onto a single 3G-SDI link, based upon the SMPTE ST 424 data structure and SMPTE ST 425-1 Level A mapping, to support the transport of 2160-line videos. As these systems require a raw uncompressed data capacity 4 times that of UHDTV1 (SMPTE ST-2082-10 for example), TICO compression method shall be used to reduce the video data rate.

For all TICO mapping, the same frame rate as the original signal shall be used:

- UHDTV1 @ 60Hz (or 60/1.001) == SMPTE ST 425-1 Level A (SMPTE ST 274 system 1 or 2)
- UHDTV1 @ 50Hz == SMPTE ST 425-1 Level A (SMPTE ST 274 system 3)
- UHDTV1 @ 30Hz (or 30/1.001) == SMPTE ST 292-1 (SMPTE ST 274 system 7 or 8)
- UHDTV1 @ 25Hz == SMPTE ST 292-1 (SMPTE ST 274 system 9)
- UHDTV1 @ 24Hz (or 24/1.001) == SMPTE ST 292-1 (SMPTE ST 274 system 10 or 11)

4.2.2 3G-SDI container

4.2.2.1 Video Payload Identifier

The SMPTE ST 352 payload identifier that is present in the SDI output shall reflect the SDI “container”, not the TICO picture contained therein. Information about the TICO picture shall be inside the TICO data bitstream (see *3. TICO Compressio*) and TICO Video_essence_box (see *4.1 Video essence box*). When the decoder reconstructs the image, it may also output appropriate payload identification on whatever output interface it has.

The payload identifier shall be present and shall be mapped into the ancillary data space in conformance with SMPTE ST 352. Byte 1 shall be set to 89h (according to the payload identifier definition for SMPTE ST 274 from SMPTE ST 425-1 document).

4.2.2.2 Timing Reference Codes

Timing Reference codes shall be in conformance with those specified in SMPTE ST 425-1 for the corresponding mapping: Level A Structure 1 – SMPTE ST 274. The Audio of the SDI shall be from the same period of time as the video samples (not from a frame before or a frame after).

4.2.2.3 Ancillary Data

Ancillary data, if present, shall be mapped into the blanking area in conformance with SMPTE ST 291-1. The VANC of the SDI should be from the same period of time as the video samples (not from a frame before or a frame after).

4.2.2.4 Audio Data

Audio data, if present, shall be mapped into the ancillary data space in conformance with SMPTE ST 299-2.

4.2.2.5 Time Code

Time code, if present, shall be mapped into the ancillary data space in conformance with SMPTE ST 12-2.

4.2.2.6 Frame Rate

As described in section 4.2.1, Frame rate shall be the same frame rate as the SDI essence.

4.2.3 Active Area with new TICO transported data

4.2.3.1 Overview

The active area shall be the combination of two main parts:

- Part 1 shall be the Detection_box() in order to ensure a reliable detection of TICO compression usage. This box is described in the section 4.2.3.2. TICO Detection box. The size of the box shall be greater or equal to 660 bits and shall be always smaller than 2 x 660 bits.

- Part 2 shall be the Video_essence_box() followed by the TICO picture() box. This part shall be reformatted to 10-bit value with the exclusion of forbidden values. It shall be achieved with an efficient 640-bit to 660-bit conversion function (see section 4.2.3.4 *640_to_660() Function for 10-bit Formatting*). To ease the remapping, the size of this second part shall always be a multiple of 660 bits.

It shall not be allowed to map two consecutive compressed TICO picture() in one active area.

As an example for UHDTV1 @ 60Hz, the UHDTV1 TICO compressed stream shall be transported on the active area of a 1080p @ 60Hz. The total size of this active area is 2,073,600 x 20 bits (being 1920 * 1080 * 20). The part 1 and part 2 sizes are therefore:

Table 37 – Example of Part 1 and Part 2 sizes on the active area

	bits	*20-bit words*	*660-bit words*
Part1_size	900	45	1 + 4/11
Part2_size	41,471,100	2,073,555	62,835
Video_active_area_size	41,472,000	2,073,600	62,836 + 4/11

4.2.3.2 Part 1: TICO Detection Box

The Detection_box() is the Part 1 of the active area. This box shall be inserted to ensure a reliable detection of the usage of TICO compression in the active area of the 3G-SDI link. This box contains enough random data so that the detection is statistically reliable.

The bit length size shall be always computed from the active area size according to the following formula:

Part_1_size (bits) = 660 bits + (*video_active_area_size* modulo 660) bits

The size of the box is therefore always greater or equal to 660 bits and always smaller than 2 x 660 bits.

The Detection_box() shall include two fields:

- The first field shall be a single 10-bit word. It is the *detection_box_length* field. It is the first 10-bit value of the active area. It shall be equal to the size of the Detection_box() expressed as the number of 10-bit words (so *Part_1_size* / 10). The valid range is therefore [66, 131].

- The second field shall be the random data field. It shall contain (*detection_box_length-1)* * 10-bit words. The random values shall be produced using a Lehmer Pseudo Random generator of 9-bit word (values between 0 and 511). The selected generator shall be:

  $x(n) = (3*x(n-1)+1) \bmod 512$ with an initial seed $x(-1)$ equals to 32.

  The 10-bit value (avoiding the forbidden values) shall be produced by the addition of 512 if the 9-bit value is below 256. If the 9-bit value is greater or equal to 256, the 10-bit value shall be equal the 9-bit value.

This Detection_box() is very simple to be implemented in hardware (only addition and shift). In software, it can be easily inserted and detected using a constant ROM.

The generated sequence is normative. The goal is to have a sequence of random data to ensure a reliable detection. The probability of false detection is at least 1 picture every $2^{66*9} = 2^{594}$ assuming that real video is purely random data.

For a 1080p active area, the following Detection_box() shall be inserted. This is normative. Knowing that the Part1_size is 900 bits, the first 10-bit value (being the *detection_box_length* field) shall have the value of 90 (0x05A in hexadecimal), as in Table 38.

Table 38 – Detection box values

0	1	2	3	4	5	6	7	8	9
0x05A	0x261	0x124	0x16D	0x248	0x2D9	0x28C	0x1A5	0x2F0	0x2D1
10	11	12	13	14	15	16	17	18	19
0x274	0x15D	0x218	0x249	0x2DC	0x295	0x1C0	0x141	0x1C4	0x14D
20	21	22	23	24	25	26	27	28	29
0x1E8	0x1B9	0x12C	0x185	0x290	0x1B1	0x114	0x13D	0x1B8	0x129
30	31	32	33	34	35	36	37	38	39
0x17C	0x275	0x160	0x221	0x264	0x12D	0x188	0x299	0x1CC	0x165
40	41	42	43	44	45	46	47	48	49
0x230	0x291	0x1B4	0x11D	0x158	0x209	0x21C	0x255	0x100	0x101
50	51	52	53	54	55	56	57	58	59
0x104	0x10D	0x128	0x179	0x26C	0x145	0x1D0	0x171	0x254	0x2FD
60	61	62	63	64	65	66	67	68	69
0x2F8	0x2E9	0x2BC	0x235	0x2A0	0x1E1	0x1A4	0x2ED	0x2C8	0x259
70	71	72	73	74	75	76	77	78	79
0x10C	0x125	0x170	0x251	0x2F4	0x2DD	0x298	0x1C9	0x15C	0x215
80	81	82	83	84	85	86	87	88	89
0x240	0x2C1	0x244	0x2CD	0x268	0x139	0x1AC	0x105	0x110	0x131

4.2.3.3 Part 2: 640_to_660() Function for 10-bit Formatting of the Video_ essence_box() and TICO picture().

According to SMPTE ST 292-1 and SMPTE ST 425-1, active video data values 000h – 003h and 3FCh – 3FFh shall be excluded and reserved for sync words (EAV, SAV and ancillary data start). These markers play a specific role for the descrambling and synchronization of the SDI data. The new transported active area shall therefore also exclude those words as illustrated bellow.

Part 2 includes the recoding of the Video_essence_box() and the TICO picture() box. The recoding shall be achieved using a 640-bit to 660-bit algorithm that ensures that all generated 10-bit values are bounded between 64 and 959. Thanks to this property, all Part2 data are easy to map on SDI where video data values from 0 to 3 and from 1020 to 1023 shall be reserved as specified by SMPTE ST 292-1 and SMPTE ST 425-1.

Knowing the Video_essence_box size (a multiple of 64 bits), user shall configure the TICO compressed frame budget in such way that the total data size (size of video essence box and Tico picture box) shall always be a multiple of 640 bits. All 640-bit packets shall be recoded easily to 660-bit. By doing so, there shall be no need of additional padding.

The first part of 640_to_660() function shall consist of dividing the 640 consecutive bits into 16 consecutive groups of 40 bits, as illustrated in Figure 13. Each 40-bit word is formed by filling it byte by byte, LSB first. (i.e. little-endian) : word[n] = (data[5*n+4] << 32) | … | (data[5*n+0] << 0).

640-bit																
40-bit	40-bit	40-bit	40-bit	40-bit	40-bit	40-bit	40-bit	40-bit	40-bit	40-bit	40-bit	40-bit	40-bit	40-bit	40-bit	

660-bit																
20-bit	40-bit	40-bit	40-bit	40-bit	40-bit	40-bit	40-bit	40-bit	40-bit	40-bit	40-bit	40-bit	40-bit	40-bit	40-bit	40-bit

Figure 13 – 640 consecutive bits division into 16 consecutive groups of 40 bits

As shown in Figure 14, the bitstream recoding shall be performed in different steps on every 40-bit group (being 4 x 10-bit words).

- The first step shall generate one supplementary bit (S bit, as illustrated in blue below) per 40-bit value (so 2x8 in total for 640-bit). This step shall generate two extra 10-bit words (8 bits per 10-bit word). The two MSB bits of each 10-bit word are zero bits. One extra 20-bit word shall be therefore generated per 640-bit. The extra bits shall be filled into the 20-bit word by filling MSBs first. (bit 7 of first extra 10-bit word corresponds to word[0])

- The second step shall recode few bits of the 40-bit value. Only 12 bits shall be modified (in yellow); three MSB of each 10-bit word (MSB_{11}, MSB_{10}, MSB_{01} and MSB_{00}). The remaining bits (in green) remain the same (not modified).

- The third step shall increment by 64 all 10-bit words, including both 10-bit words of supplementary bits.

Those steps are applied on all 640-bit packets. The final recoded bitstream starts first with the 20-bit of extra bits followed by 640-bit of the recoded data and so on. Having the extra words at beginning of bitstream eases the decoding. The budget overhead for this transcoding is therefore 20/640 being exactly 3,125% (so 4,125:1 for 4:1 compression ratio).

The 660-bit packets in following order: first the two extra 10-bit word (starting with the one containing word[0] extra bit). Then the word[0] is transmitted with LSB first (so bit [9:0] are transmitted first, then [19:10] and so forth).

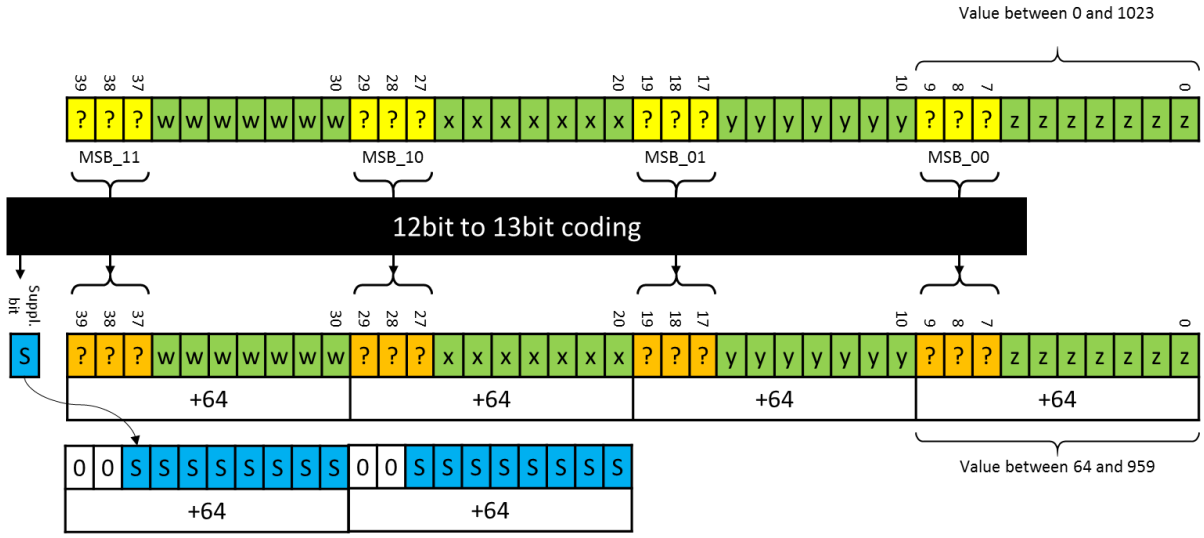

Figure 14 – Bitstream recoding

The first step requires further explanations. The supplementary bit shall be set to ‘1’ if any of the four MSB equals “111”. If no MSB equals “111”, the supplementary bit shall be set to ’0’.

The second step shall be function of the number of MSBs equal to “111”:

If no MSB equals “111” the four 10-bit words shall be unchanged.

If only 1 MSB equals “111”, the MSB shall be recoded as shown in the table hereunder. The index of the single MSB equal to “111” shall be mapped on bits 38 and 37, while the other MSB values shall be mapped raw in an ordered way ($a > b > c$).

39	38	37	29	28	27	19	18	17	9	8	7	others
0	Index of MSB equal to 111		MSB_a			MSB_b			MSB_c			idem

If 2 MSB equal “111”, the MSB shall be recoded as shown in the table hereunder. There are two indexes of MSB equal to “111”. The highest index shall be mapped on bits 37 and 29, and the lowest index on bits 28 and 27. The value of the other MSB different than “111” shall be mapped raw in an ordered way ($a > b$).

39	38	37	29	28	27	19	18	17	9	8	7	others
1	0	Highest index of MSB equal to 111		Lowest index of MSB equal to 111		MSB_a			MSB_b			idem

If 3 MSB equal “111”, the MSB shall be recoded as shown in the table hereunder. Bits 28, 27, 18 and 17 shall be set to ‘1’ if their respective MSB equal to “111”, else they shall be set to ‘0’. The value of the only MSB different than “111” shall be mapped raw in bits 9 down to 7.

39	38	37	29	28	27	19	18	17	9	8	7	others
1	1	0	0	$MSB_{11} = 111$	$MSB_{10} = 111$	0	$MSB_{01} = 111$	$MSB_{00} = 111$	MSB_a			idem

If all 4 MSB equal “111”, the MSB shall be recoded as shown in the table hereunder. The single difference with the previous case (3 MSB equal to “111”) shall be that bits 9 down to 7 shall be set to “000” since there is no MSB different than “111” left.

39	38	37	29	28	27	19	18	17	9	8	7	others
1	1	0	0	1	1	0	1	1	0	0	0	idem

4.2.4 TICO Mapped in SDI with SMPTE ST 2022-5/6/7

Since the TICO bitstream is embedded in standard 3G-SDI, that 3G-SDI can be transported in SMPTE ST 2022-5/6/7. System shall work without any modification of SMPTE ST 2022-5/6/7 standards.

4.3 TICO Mapping to RTP

This section specifies the payload format for TICO video streams over the Real-time Transport Protocol (RTP).

In order to be transported over RTP, each TICO video stream is transported in a distinct RTP stream, identified by a distinct SSRC.

Each of those RTP streams is divided into Application Data Units (ADUs). Each ADU shall correspond to a single Video_essence_box and a single TICO picture.

Each ADU is then split into packets, depending e.g. on the Maximum Transmission unit (MTU) of the network. Every packet has the same size, last packet could be shorter.

The packet boundaries coincide with the ADU boundaries.

The SSRC RTP field is used to discriminate each separate TICO video stream from others. Within a specific TICO video stream, identified by its SSRC, the picture counter field is used to identify to which picture a packet corresponds to.

Figure 15 illustrates the RTP payload header used in order to transport each TICO video stream (identified by a distinct SSRC).

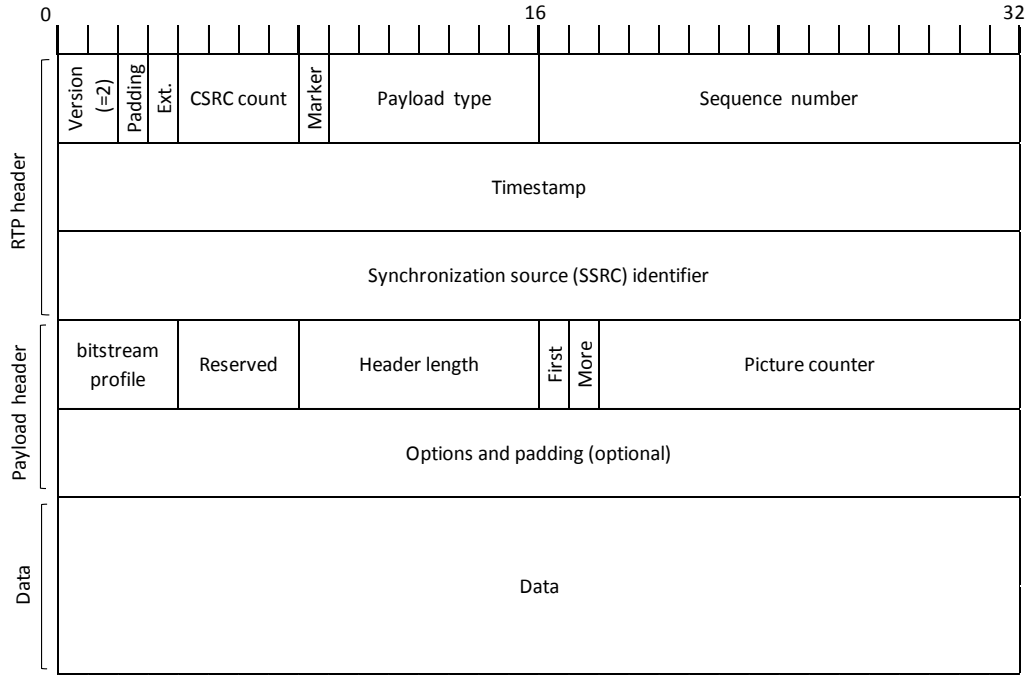

Figure 15 – RTP and payload headers

A 90-kHz timestamp shall be used. If the sampling instant does not correspond to an integer value of the clock, the value shall be truncated to the next lowest integer, with no ambiguity.

As per specified in RFC 3550 and RFC 4175, the RTP timestamp designates the sampling instant of the first octet of the picture to which the RTP packet belongs. Packets shall not include data from multiple frames, and all packets belonging to the same frame shall have the same timestamp. Several successive RTP packets will consequently have equal timestamps if they belong to the same picture, and the timestamp will only be increased when first = 1.

The RTP timestamp designates the sampling instant of the first octet of the picture to which the RTP packet belongs. Packets shall not include data from multiple frames, and all packets belonging to the same frame shall have the same timestamp. Several successive RTP packets will consequently have equal timestamps if they belong to the same picture, and the timestamp will only be increased when first = 1.

The timestamp should be based on a globally synchronized 90 kHz clock reference, and should correspond to the number of cycles since the SMPTE Epoch (as per defined in SMPTE ST 2059-1) modulo 2^32:

timestamp = ((now - epoch)*90000) % 2^32

where *now* and *epoch* are real numbers expressed in seconds, *now* being the current timestamp and *epoch* the reference timestamp (see SMPTE ST 2059-1).

The corresponding headers shall be as described in Table 39.

Table 39 – Payload header fields description

Field	Width	Description
bitstream_profile	4 bits	It is the profile number of the TICO bitstream. This field shall be the same as defined in the picture() header.
Reserved	4 bits	Shall be set to 0 by the encoder, Shall be ignored by the decoder
Header length	8 bits	total payload header length in 32-bit words if no options are used, Shall be set to 1
First	1 bit	Shall be set to 1 if the current packet is the first of the current picture, Shall be set to 0 otherwise
More	1 bit	Shall be set to 0 if the current packet contains the end of the current picture Shall be set to 1 otherwise (more packets are expected to complete the current frame)
Picture counter	14 bits	counter indicating the current picture number. Counter value increments by 1 every time First flag equals to 1.
Options	(header length - 1) × 32 bits	options (reserved, optional)

Annex A RAND IP Licensing

1. The content of this document and the use of the disclosed technology (notably for compliance with the TICO® compression and its subsequent mapping on SDI), implies the use of intellectual property rights (IPR) owned by intoPIX.

2. All inquiries as to intellectual property and licensing requirements that may be attached to the use the disclosed technology, should be addressed to the proponent identified in this document.

Annex B Complementary Pseudo Code (Informative)

B.1 Coded Packets Decoding Pseudo Code

```
 1
 2    /* Unary value extraction routine */
 3    read_unary(cursor)
 4    {
 5        unary_value = 0
 6        while ((read_1_bit(cursor) == 0) && unary_value < 15)
 7            unary_value++;
 8        if ((unary_value > 0) && (read_1_bit(cursor) == 1))
 9            unary_value = -unary_value;
10    }
11
12    extract_coded_data_packet()
13    {
14        /* Inputs */
15        gcli_coding_mode = coding mode read in precinct header
16        cursor = pointer to current bit of bitstream
17        sb_width4[][] = subband sizes in number of GCLI
18        gtli_noref[][] = subband GTLIs without refinement
19        gtli_noref_prev[][] = GTLIs without refinement same line previous precinct
20        gclis_prev[][][] = gclis of same line of previous precinct
21
22        /* Local Variables */
23        delta = temp variable for storing gcli delta
24        comp = component number
25        sb = horizontal subband number (0 for lowest frequency, 5 for highest)
26        x = gcli index in subband
27
28        /* Output Result */
29        gclis_line[][][] = array of output gclis for all sb of this line
30
31        for (comp = 0; comp < COMP_NB; comp++ )
32        {
33            for (sb = 0; sb < HOR_DECOMPOSITION_NB+1; sb++)
34            {
35                if (gcli_coding_mode == 0) /* Raw values */
36                {
37                    /* Raw read of GCLIs */
38                    for (x = 0; x < sb_width4[comp][sb]; x++)
39                        gclis_line[comp][sb][x] = read_4_bits(cursor);
40                }
41                else if (gcli_coding_mode == 2) /* Horizontal deltas */
42                {
43                    /* extract the truncated GCLI values */
44                    for (x = 0; x < sb_width4[comp][sb]; x++)
45                    {
46                        if (x==0)
47                        {
48                            gclis_line[comp][sb][x] = read_4_bits(cursor);
49                        }
50                        else
51                        {
52                            delta = read_unary(cursor);
53                            /* horizontal delta inversion */
54                            gclis_line[comp][sb][x] = gclis_line[comp][sb][x-1]
55                            gclis_line[comp][sb][x] += delta;
```

```
56                  }
57               }
58               /* Restore GCLI from truncated GCLIs */
59               for (x = 0; x < sb_width4[comp][sb]; x++)
60               {
61                  if (gclis_line[comp][sb][x] > 0)
62                     gclis_line[comp][sb][x] += gtli_noref[comp][sb]
63               }
64            }
65            else if (gcli_coding_mode == 3) /* Vertical deltas */
66            {
67               /* extract GCLI values */
68               for (x = 0; x < sb_width4[comp][sb]; x++)
69               {
70                  delta = read_unary(cursor);
71                  /* vertical delta inversion */
72                  gclis_line[comp][sb][x] = MAX(gclis_prev[comp][sb][x],
73                                 gtli_noref_prev[comp][sb],
74                                 gtli_noref[comp][sb]);
75
76                  gclis_line[comp][sb][x]+=delta;
77
78                  if (gclis_line[comp][sb][x] <= gtli_noref[comp][sb])
79                     gclis_line[comp][sb][x] = 0;
80               }
81            }
82         }
83      }
84   }
```

B.2 Raw Data Packet Unpacking Pseudo Code

```
 1
 2    extract_raw_data_packet()
 3    {
 4        /* Inputs */
 5        cursor = pointer to current bit of bitstream
 6        sb_width[][] = number of coeffients per subband
 7        gtli[][] = GTLI for every subband
 8        gclis_line[][] = GCLI of line as extradted from coded data packet
 9
10        /* Local Variables */
11        comp = component number
12        sb = horizontal subband number (0 for lowest frequency, 5 for highest)
13        x = index of current gcli
14        sm[4] = array to store the four coefficients per group
15        i = index of coef in group
16        bp = bitplane index
17
18        /* Output Result */
19        coef_line[][][] = array of output coefficients for all subbands of line
20
21        for (comp = 0; comp < COMP_NB; comp++ )
22        {
23            for (sb = 0; sb < HOR_DECOMPOSITION_NB+1; sb++)
24            {
25                for (x = 0; x < (sb_width[comp][sb]+3) / 4; x++)
26                {
27                    sm[0]=sm[1]=sm[2]=sm[3]=0;
28                    if (gclis_line[comp][sb][x] > gtli[comp][sb])
29                    {
30                        sm[0] |= read_1_bit(cursor)<<15;//Sign bits
31                        sm[1] |= read_1_bit(cursor)<<15;//Sign bits
32                        sm[2] |= read_1_bit(cursor)<<15;//Sign bits
33                        sm[3] |= read_1_bit(cursor)<<15;//Sign bits
34
35                        for (bp=gclis_line[comp][sb][x]; bp>gtli[comp][sb];bp--)
36                        {
37                            sm[0] |= read_1_bit(cursor)<<(bp-1);
38                            sm[1] |= read_1_bit(cursor)<<(bp-1);
39                            sm[2] |= read_1_bit(cursor)<<(bp-1);
40                            sm[3] |= read_1_bit(cursor)<<(bp-1);
41                        }
42                    }
43                    for (i=0; i<4; i++)
44                    {
45                        /* rounding */
46                        if (sm[i] > 0 && gtli[comp][sb] > 0)
47                            sm[i] = sm[i] | (1 << (gtli[comp][sb]-1));
48
49                        if (x*4 + i < sb_width[comp][sb])
50                        {
51                            //Sign magnitude conversion to 2's complement
52                            coef_line[x*4 + i] = sm[i] & (~(1<<15));
53                            if (sm[i] & (1<<15))
54                                coef_line[x*4 + i] = -coef_line[x*4 + i];
55                        }
56                    }
57                }
```

```
58          }
59       }
60    }
61
```

Annex C IANA Considerations (Informative)

The following MIME type will be registered with IANA relating to this document:

video/tico

Annex D Bibliography (Informative)

D. Le Gall and A. Tabatabai, “Sub-band coding of digital images using symmetric short kernel filters and arithmetic coding techniques,” IEEE International Conference on Acoustics, Speech and Signal Processing, New York, NY, pp. 761–765, 1988

I. Daubechies and W. Sweldens, “Factoring wavelet transforms into lifting steps,” Journal of Fourier Analysis and Applications, no.3, pp. 247-269, 1998

Schulzrinne, et al., RFC 3550: RTP: A Transport Protocol for Real-Time Applications, 2003

Schulzrinne & Casner, RFC 3551: RTP A/V Profile, 2003

Handley & Perkins, RFC 2736: Guidelines for Writers of RTP Payload Formats, December 1999

SMPTE RP 431-2:2011, D-Cinema Quality — Reference Projector and Environment