CCITT SGXV
Working Party on Visual Telephony
Specialists Group on Coding for Visual Telephony

Doc# 85

March 86

Source: UK, Sweden and France

Title: Video multiplex for nx384kBit/s

1.Introduction

In the design of any video multiplex there is a compromise between absolute efficiency and error performance. Most simulations to date have been concerned only with efficiency. An example of this is to simply allocate one bit per block to indicate whether any particular block has data associated with it. In the case of a transmission errors whole fields of data are lost which is catastrophic in a codec which relies heavily on interframe coding. The opposite extreme is to give each block an absolute address within each field. In this case efficiency suffers.

The following proposal is a fairly simple compromise where absolute addressing is used for each line of blocks and a more efficient relative addressing mode is used between blocks. This will contain errors to within one or two lines of blocks without significantly reducing efficiency.

2. Possible Video Multiplex Arrangement

i. Field Start Code

1000 0000 0000 0000 0000 [Buffer State] [Temporal Ref.][Type]

Buffer State :- 6 bit number representing the encoder buffer fullness in 1kBit intervals at the begining of the current field.

Temporal Ref. :- A three bit number representing the time sequence in 1/30 sec. intervals of a particular field. All Field Start Codes should be transmitted.

Type :- This is a VLC code which allows block attributes to be applied to all blocks within a field (eg. all blocks may be intraframe coded or non-motion compensated). This saves overhead by removing the requirement to signal Block Type on a per block basis.

NB. The exact bit pattern and the word length of the Field Start Code will depend on the VLC code set chosen and is for further study.

ii. Line of Blocks Start Code (LBSC)

1000 0000 0000 1111 [Block Line Number][Type]

Block Line Number:- A six bit number representing the vertical spatial position in blocks of the current line of blocks. All line of block codes should be transmitted.

NB: A line of blocks may consist of two lines of luminance blocks, one line of U blocks and one line of V blocks if colour components are chosen to be twice the size of luminace blocks. This will mean that a Line of Block Start Code defines the begining of a physical region of picture (luminance and colour differences) outside of which errors are unlikely to extend.

The exact bit pattern and the word length of the Line of Blocks Start Code will depend on the VLC code set chosen and is for further study.

Type :- This is a VLC code which allows block attributes to be applied to all blocks within a line of blocks (eg. all blocks may be intraframe coded or non-motion compensated). This saves overhead by removing the requirement to signal Block Type on a per block basis.

iii. Block Address

[Block Address]

A VLC code indicating the relative position of a moving block (relative to the previous block or picture boundary).

Addresses of absolute position greater than 45 are considered to be chrominance blocks.

iv. Block Type

[block type]

A VLC code representing the type of the block (possible attributes are described in doc.69 France). Possible types are:-

i. Intraframe coded block
ii. Interframe coded block
iii. Motion compensated block
iv. Motion compensated with coded residue
v. Future expansion on
vi. Future expansion off

Codecs should be designed to ignore all data between types v. and vi. , also between type v. and the next LBSC. This will allow us to efficiently include some enhancements at a later date without effecting compatibility. (NB. to do this we must define some limitations on the structure of the future expansion data.)

v. Block Data

[Data]

The exact form of this is for further study. It will probably include motion vector data, scanning class, quantiser type, coefficient data and an end of block data code.

3. Typical data structure

```
  [Frame start code1][Frame start code2][Frame start code3]
..[LBSC1]LBSC2][LBSC3][BLOCK ADDRESS]....
..[BLOCK TYPE.][BLOCK DATA][BLOCK ADDRESS]..
..[BLOCK TYPE.][BLOCK DATA][LBSC4][LBSC5]..
..[LBSC6]....etc.
```

4. Generalised VLC

It would be highly desirable to fixed on one VLC code book which can be used for all situations where VLC occurs. This will both simplify hardware and aid retracking when transmission errors occur.

NB. All of the above bit allocations and calculations assume 8*8 block size.

5. A Possible Video Multiplex Arrangement

An example of bit allocations is shown below:-

1, Picture attributes,

1,1	Frame start code	f bits
1,2	Buffer state	6 bits
1,3	Temporal reference	3 bits
1,4	? Picture attribute	? bits

2, Group of Block (GOP) attributes,

2,1	GOP start code	g bits
2,2	GOP number	x bits
2,3	no/some motion vectors in GOP	1 bit
2,4	no/some coded blocks in GOP	1 bit
2,5	quantizer use	5 bits
2,6	inter/intra	1 bit
2,7	? GOP attribute	? bits
2,8	no/some ? block attributes in GOP	1 bit

attributes 2,5 and 2,6 can be left out if 2,4 flags "no",

3, Block attributes

3,1	moving/non-moving only for Y-blocks only if 2,3 flags "some"	m bits
3,2	motion vectors only for blocks flagged "moving" in 3,1	M bits
3,3	coded/uncoded for Y,U,V-blocks only if 2,4 flags "some"	c bits
3,4	? attributes only if 2,8 flags "some"	? bits

4. For each block flagged "coded" in 3.3

4.1	scanning class	? bits
4.2	no of non-zero components in block	? bits
4.3	vlc words for the components	? bits

COMMENTS

For future extension possibilities "? attributes" are introduced in both "picture" "GOP" and "block" level. If not used, these will only cost a very small number of bits.

Differently from other proposals, "moving/non-moving" and "coded/uncoded" are flagged seperately. This is done for two reasons:

1. In cases where motion compensation is not used, as few bits as possible should be spent on motion compensation related attributes.

2. Motion vectors are only transmitted for luminance. The above arrangement is a suitable way to avoid wasteful bits saying "non-moving" for chrominance blocks.

The number of bits (m and c) for the block attributes "moving/non-movin and "coded/uncoded" can be less than one bit per block if efficient variable length coding is used. One method for this is described in the document "ARITHMETIC CODING FOR COMPRESSION OF BLOCK ATTRIBUTES". (SHOWN AS APPENDIX 1)

An example of GOP:

The coder uses 8*8 blocks for both luminance and chrominance transform, as well as for motion compensation.
A suitable GOP is then two rows of Y-blocks (2*352/8=88), one row of U-blocks (176/8=22) and one row of V-blocks, altogether 132 blocks. An entire picture will then contain 18 GOPs.

A bit budget example:

If "? attributes" azre not used, and x is set to 5 bits, the number of bits for "picture" and "GOP" attributes are

$$N = f + 9 + 18 * (g + 14) \quad \text{bits/picture}$$

With f=20 and g=16 we get

$$N = 569 \quad \text{bits/picture.}$$

Assume now that each GOP contain 10 (=11.4 %) moving blocks (each motion vector coded with 8 bits) and 10 (=7.6 %) coded blocks.

If the block attributes are not variable length coded the number of bits for attributes becomes

$$NN = 569 + 18 * (88 + 10*8) + 18 * 132$$

$$= 569 + 5400 = 5969 \quad \text{bits/picture}$$

If the block attributes are variable length coded with arithmetic codin the number of bits become (entropies: H(0.114)=0.51 H(0.076)=0.39)

```
NNA  <  569   +   18 * ( 88*0,51+5 + 10*8 )   +   18 * ( 132*0,5
     =  569   +   18 * (   49,88   +  80  )   +   18 *   56,48
     =  569   +   3355   =   3924        bits/picture
```

On top of this it is necessary to transmit component information (4,1-4,3 above),

APPENDIX 1

ARITHMETIC CODING FOR COMPRESSION OF BLOCK ATTRIBUTES

Block attributes, such as changed/unchanged, coded/uncoded and moving/non-moving, usually require one bit per block of side information, At low bit rates, in combination with small blocks, this is not acceptable, Fortunately,the block attribute bits often contain considerable redundancy, and can therefore be compressed As an example, if 10 % of the bits are "ones" and 90 % are "zeroes", the information can be compressed to 50 % with variable length coding, Arithmetic coding is an efficient method for this compression,

Arithmetic coding has three nice properties:

1) It can handle very large blocks; hundreds of symbols is possible,

2) It can be made adaptive; probabilities may change from symbol to symbol,

3) Bit rate is very close to the entropy, provided true statistics is available,

n symbols from a source with probabilities P and entropy H can be coded with less than nH+2 bits if the P-values are stored in registers in the coding unit, Coding complexity is two multiplications and one addition per coded symbol,

For block attribute information, arithmetic coding can be used in the following way:

Block attribute bits for a "group of blocks" (GOB) is considered as a block of bits, Assume that a GOB has k "ones" and n-k "zeroes" for a certain attribute, which means that this block has probability p=k/n for "ones", and 1-p for "zeroes", The value of p is approximated with three bits, and transmitted as side information, The same value of p is then used for both coding and decoding, The number of bits to code the attributes is then n*H+2+3 plus a couple of bits due to the fact that p is an approximation of the true probability,

This coding has been carried out in swedish simulations using 8*8 bloc Both coded/uncoded and moving/non-moving attributes were transmitted, The following numerical results were achieved:

Without arithmetic coding:	1620+396+396 + 1620 =	4032 bits/pict
With arithmetic coding at 300 kbit/s:	appr,	3200 bits/pict
With arithmetic coding at 60 kbit/s:	appr,	2100 bits/puct

ref: L, Zetterberg, S, Ericsson and H, Brusewitz
"Interframe DPCM with Adaptive Quantization and Entropy Coding
IEEE Transaction on Communications, august 1982, pp, 1888-1899