XAPP299 (v1.0) May 16, 2002

Serial Digital Interface (SDI) Ancillary Data and EDH Processors

Author: John Snow

Summary

The SMPTE 259M Serial Digital Interface (SDI) Standard describes how to transmit standard-definition digital video serially over coax cable. SDI is commonly used to transport digital video in broadcast studios and video production centers.

This application note describes implementations of an ancillary data packet processor and an error detection and handling processor for the SDI interface.

Introduction

This is one in a series of application notes describing SDI implementation in Xilinx FPGAs. Figure 1 is a block diagram showing correlation between the various application notes and the elements of the SDI link.

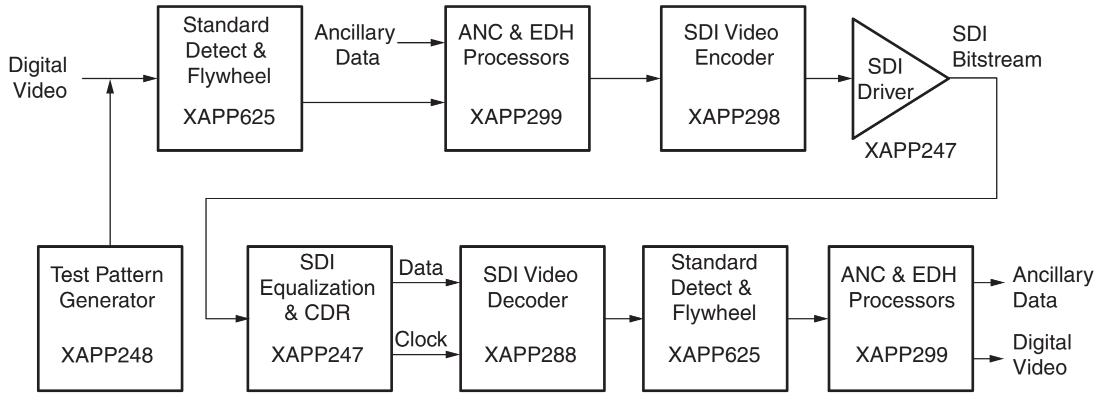

Digital Video
Standard Detect & Flywheel XAPP625
Ancillary Data
ANC & EDH Processors XAPP299
SDI Video Encoder XAPP298
SDI Driver
XAPP247
SDI Bitstream
Test Pattern Generator XAPP248
SDI Equalization & CDR XAPP247
Data
Clock
SDI Video Decoder XAPP288
Standard Detect & Flywheel XAPP625
ANC & EDH Processors XAPP299
Ancillary Data
Digital Video
x299_01_050702

Figure 1: **SDI Block Diagram and Related Application Notes**

Before transmission over an SDI link, a digital video stream is usually processed to insert error detection packets. These packets contain checkwords allowing the receiver to detect transmission errors. Ancillary data packets can also be inserted into the inactive (blanked) portions of the video to carry non-video data such as digital audio. At the receiving end of the SDI link, the digital video stream is again processed to detect transmission errors, extract ancillary data, and possibly insert additional types of ancillary data.

The functions described in this application note, combined with the video decoder described in XAPP625, form a processor capable of implementing the SDI standard error detection protocol and ancillary data packet processing. Figure 2 shows a block diagram of the complete processor.

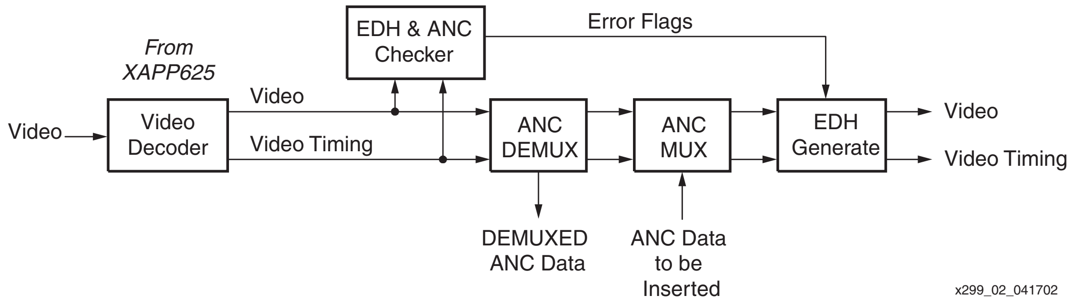


Figure 2: **EDH and ANC Processor Block Diagram**

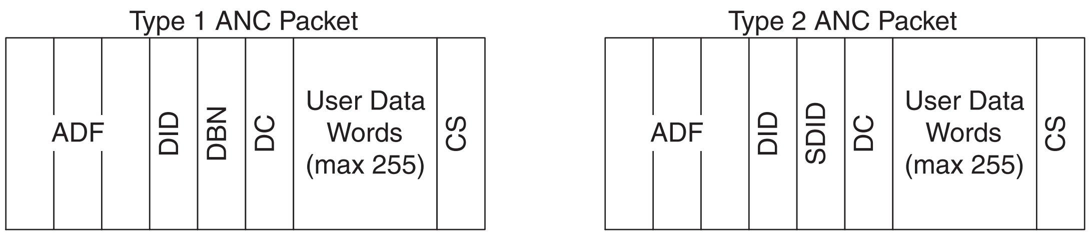
ANC Packets

Ancillary data (ANC) packets carry non-video information in the inactive portion of the digital video stream. ANC packets can carry any type of digital information. One of the most common uses of ANC packets is to carry the digital audio portion of the program. A number of commonly used ANC packet types have been standardized. User defined ANC packet types are also allowed.

The general format of ANC packets is defined in the SMPTE 291M[1] and the ITU-R BT.1364[2] standards. These standards also describe the spaces where ANC packets are permitted in the video frame. These standards do not define the contents of any particular ANC packet type. Standard ANC packet types are typically defined in separate documents. For example, the ANC packet type for digital audio is defined in SMPTE 272M[3].

ANC Packet Format

Figure 3 shows the general format of an ANC packet. There are two nearly identical formats permitted, Type 1 and Type 2. In Type 1 packets, an 8-bit identification word identifies the contents of the packet. In Type 2 packets, the identification value is a 16-bit value sent in two separate words in the packet.



Word Contents	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Ancillary Data Flag, Word 1 (000_{HEX})	0	0	0	0	0	0	0	0	0	0
Ancillary Data Flag, Word 2 ($3FF_{HEX}$)	1	1	1	1	1	1	1	1	1	1
Ancillary Data Flag, Word 3 ($3FF_{HEX}$)	1	1	1	1	1	1	1	1	1	1
Data ID	$\overline{P}$	P	did7	did6	did5	did4	did3	did2	did1	did0
Secondary DID/Data Block Number	$\overline{P}$	P	sdid7 dbn7	sdid6 dbn6	sdid5 dbn5	sdid4 dbn4	sdid3 dbn3	sdid2 dbn2	sdid1 dbn1	sdid0 dbn0
Data Count	$\overline{P}$	P	dc7	dc6	dc5	dc4	dc3	dc2	dc1	dc0
User Data Words (0 to 255 Words)	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Checksum	$\overline{cs8}$	cs8	cs7	cs6	cs5	cs4	cs3	cs2	cs1	cs0

Note: P is an even parity bit for bits b7 through b0 and is located in b8.
Words containing a P bit in b8, also have the inverse of b8 located in b9.

x299_03_041802

Figure 3: **ANC Packet Format**

Every ANC packet begins with a three-word ancillary data flag (ADF). The first word of the ADF is all zeros (000_{HEX}). The second and third words of the ADF are all ones ($3FF_{HEX}$). This three-word sequence is unique in the bitstream and only occurs at the beginning of an ANC packet.

The ADF is followed by three words that indicate the type and length of the packet. All three of these words contain eight-bit values located in the least significant eight bits (bits 7 to 0). In all three of these words, bit 8 contains an even parity bit calculated from bits 7 through 0. Bit 9 is the complement of bit 8. Requiring bit 9 to be the complement of bit 8 prevents these words from ever having the values of all ones or all zeros — values restricted from occurring anywhere in the video stream except in the timing-reference signal (TRS) symbols and in the ADF of an ANC packet.

The word immediately after the ADF contains the Data ID (DID) value identifying the type of ANC packet. Usually, bit 7 of the DID value indicates whether the packet is a Type 1 packet (b7 = 1) or a Type 2 packet (b7 = 0). However, if the 8-bit DID value is 00_{HEX}, this indicates an undefined packet type.

For Type 2 packets, the word after the DID contains the Secondary Data ID (SDID) value. The SDID value is combined with the DID to provide a 15-bit packet identification code. The identification code is effectively only 15 bits, because bit 7 of the DID word is always 0 for Type 2 packets).

For Type 1 packets, the word after the DID contains the Data Block Number (DBN) value. Use of the DBN is optional. The DBN is used to provide a block sequence number when a group of related Type 1 packets requires a continuity numbering system. Valid DBN values range from one through 255. When the DBN is unused, then the DBN value must be 00_{HEX}.

The word following the SDID/DBN contains the Data Count (DC). The DC indicates the number of words in the payload portion of the packet. The DC can range from zero (indicating that the payload is empty) to 255. ANC packets are restricted to a maximum of 255 payload words.

The payload section begins immediately after the DC word. The words in the payload section are called User Data Words (UDW). The definition of the UDW data is completely dependent upon the packet format. User data words are not restricted to 8-bit values. All ten bits of each UDW can be used.

The checksum word (CS) is immediately after the last UDW. The CS provides some error detection capabilities for the ANC packet. The CS is a 9-bit checksum value computed by adding the 9-bit values (bits 8 through 0) of all words in the ANC packet from the DID word through the last UDW, and discarding any carries that result from the additions. Bit 9 of the CS word is the complement of bit 8.

The checksum only provides limited error detection capabilities. The checksum calculation does not include the MSB of any of the words in the packet, so an error in the MSB of a word might go undetected. Many ANC packet formats simply follow the general format of the ANC packet and only carry 8-bit data in the UDW words, using bit 8 as a parity bit and bit 9 as the complement of the parity bit. Some ANC packet formats include error detection or even error correction information in the payload section itself.

Non-conforming ANC Packets

The ITU-R BT.1364 standard describes a third type of ANC packet called a non-conforming packet. Use of non-conforming ANC packets is not recommended by the standard, but is tolerated. The main advantage of a non-conforming ANC packet is that it allows for a contiguous payload of more than 255 words.

A non-conforming ANC packet is preceded by an ANC packet called a *start marker* packet. The *start marker* packet is a standard Type 1 ANC packet with a DID value of 88_{HEX}. The DC word of the *start marker* packet must be zero, indicating that the *start marker* packet contains no user data words. The length of the *start marker* packet is exactly seven words long including the ADF.

Immediately after the *start marker* packet, the non-conforming ANC data is inserted. This data has no ADF and no predefined format. However, the non-conforming data must not include words with values in the reserved ranges 000_{HEX} to 003_{HEX} and $3FC_{HEX}$ to $3FF_{HEX}$.

The non-conforming ANC packet ends with the ADF of a conforming ANC packet. Normally, this is an *end marker* packet. The *end marker* packet has a DID value of 84_{HEX}. The *end marker* packet is similar to the *start marker* packet. It has a DC value of zero, no user data words, and a total packet length of seven words including the ADF. It serves simply to denote the end of a non-conforming ANC packet.

When a processor inserts a new non-conforming ANC packet, it must always insert an *end marker* packet following the non-conforming data. However, downstream equipment that inserts a new conforming ANC packet can replace the *end marker* packet with the new conforming ANC packet, since any conforming ANC packet, *end marker* or otherwise, serves to mark the end of the non-conforming packet.

When two non-conforming ANC packets appear back-to-back in an ANC space, a *start marker* packet separates them. However, an *end marker* packet does not occur between them.

There are a number of disadvantages to non-conforming data packets. First, there is no standard method for identifying the contents of the non-conforming packet. It is just raw data and does not contain any identification words in standard fixed locations. Second, there is no easy procedure for marking the non-conforming packet for deletion. To delete a non-conforming packet, the entire space occupied by the non-conforming packet must be filled with one or more conforming packets marked for deletion. If, for some reason, the non-conforming space is smaller than the minimum length of a conforming ANC packet (seven words) then the non-conforming packet would have to be merged with the preceding *start marker* packet and the combination marked for deletion.

Figure 4 shows a non-conforming ANC packet.

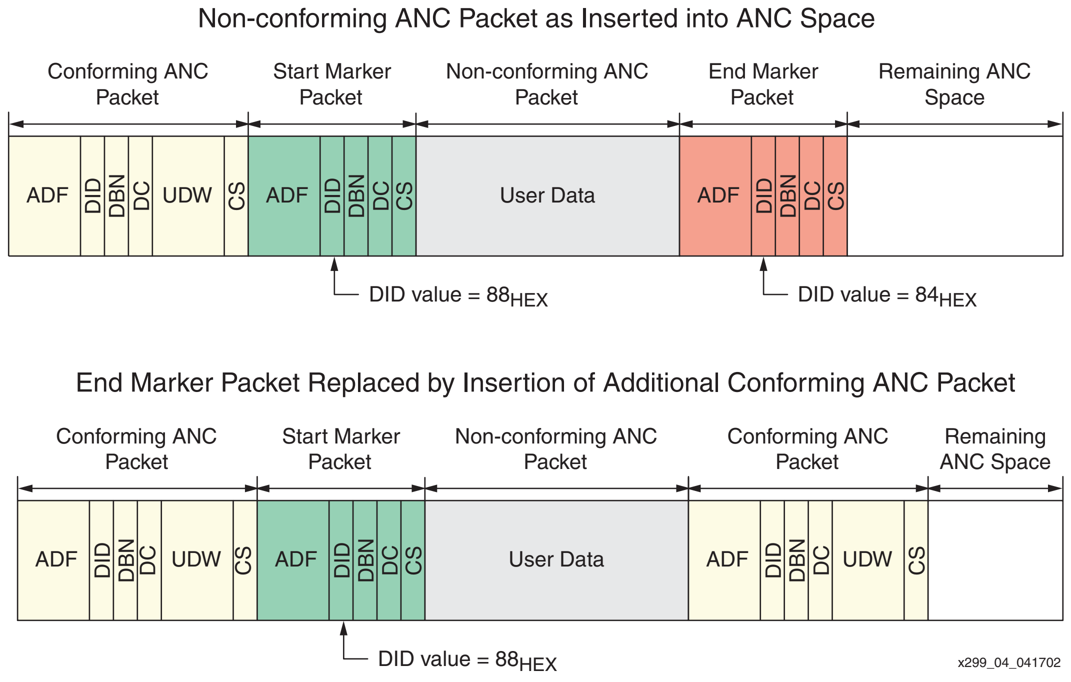


Figure 4: **Non-conforming ANC Packets**

Another Start and End Marker Protocol

The SMPTE 291M standard optionally allows the *start marker* packet and *end marker* packet to be used to identify the starting and ending locations of an ancillary data space. In this usage, the *start marker* packet, if used, always occurs immediately after the TRS symbol that begins the ANC space. The *end marker* packet is placed after the last ANC packet (conforming or non-conforming) in the data space. The use of the *start marker* packet at the beginning of the ANC space is optional, even if the *end marker* packet is used. If there is insufficient space for an *end marker* packet at the end of the ANC space, the *end marker* packet is not inserted.

The implication of this optional protocol is that any piece of equipment can consider the rest of the ANC space empty if it finds an *end marker* packet. A piece of equipment designed to insert new ANC packets should, therefore, always overwrite an *end marker* packet when inserting a new packet, regardless of whether the equipment supports the *start marker*/*end marker* protocol allowed by SMPTE 291M. Inserting a new packet after an *end marker* packet, rather than overwriting the *end marker* packet, can result in other equipment not recognizing or overwriting the packet.

8-bit Considerations

ANC packets are primarily designed to work with 10-bit equipment, but there are provisions in the standards for dealing with ANC packets generated by 8-bit equipment.

When 8-bit equipment inserts an ANC packet, the 8-bit information is inserted into the eight MSBs of the video stream and the two least significant bits (bits 1 and 0) are invalid. This limits the DID value to 6-bits. Certain DID values have been reserved to identify 8-bit packets. DID values in the range of 04_{HEX} to $0F_{HEX}$ are reserved for 8-bit packets. Because the two LSBs have to be ignored and a DID value of zero is reserved, there are only three valid 8-bit DID values (04_{HEX}, 08_{HEX}, and $0C_{HEX}$).

The SDID value in 8-bit packets is also limited to 6 bits. An SDID value of zero is reserved for an undefined format type, so only 63 valid 8-bit SDID values are allowed.

The DC value is also limited to 6 bits. In order to allow up to 255 user data words in 8-bit ANC packets, the DC value in an 8-bit packet indicates the number of blocks of four user data words in the payload. The 8-bit equipment that generates the packet must pad the payload to an even multiple of four words, if necessary, to make the payload section end on a four-word block boundary.

ANC Packet Positioning

There are two types of spaces in the video stream where ANC packets are allowed. The first is the horizontal blanking interval of the video line. This is called the horizontal ANC space (HANC). The second space is the active portion of those video lines in the vertical blanking interval. This is called the vertical ANC space (VANC). Some ANC packet formats are always placed in the HANC area, others always in the VANC area, while some can be placed in either area. Figure 5 shows available ANC spaces in NTSC frames.

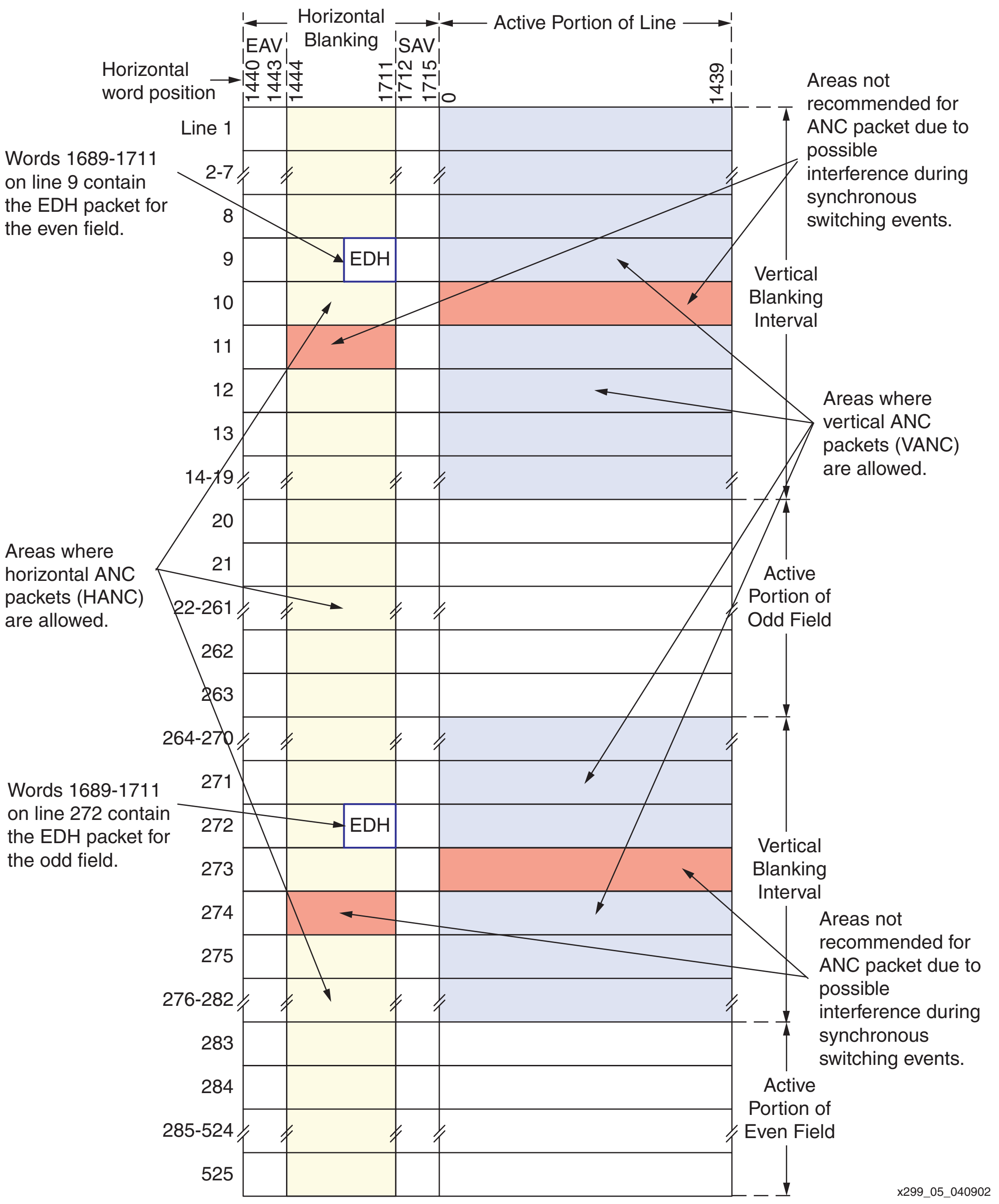

Horizontal Blanking
EAV
SAV
Active Portion of Line
Horizontal word position
1440
1443
1444
1711
1712
1715
0
1439
Line 1
2-7
8
9
10
11
12
13
14-19
20
21
22-261
262
263
264-270
271
272
273
274
275
276-282
283
284
285-524
525
EDH
EDH
Words 1689-1711 on line 9 contain the EDH packet for the even field.
Areas where horizontal ANC packets (HANC) are allowed.
Words 1689-1711 on line 272 contain the EDH packet for the odd field.
Areas not recommended for ANC packet due to possible interference during synchronous switching events.
Vertical Blanking Interval
Areas where vertical ANC packets (VANC) are allowed.
Active Portion of Odd Field
Vertical Blanking Interval
Areas not recommended for ANC packet due to possible interference during synchronous switching events.
Active Portion of Even Field
x299_05_040902

Figure 5: **Available ANC Spaces in NTSC Frame**

In a particular ANC space, ANC packets must be contiguous with each other. For example, the HANC space of a line begins with an end-of-active-video (EAV) symbol and ends with a start-of-active-video (SAV) symbol. If there are any ANC packets in a line's HANC space, the first ANC packet must begin immediately after the last word of the EAV symbol. The next ANC packet must begin immediately after the last word (CS) of the first ANC packet, and so on. If an ADF does not occur at the beginning of the HANC space, the receiver can consider the HANC space to be empty. If an ADF does not occur immediately after the last word of an ANC packet, the receiver can consider the rest of the space to be empty. An ANC packet must fit entirely within the space. It cannot overwrite the TRS symbol that marks the end of the space.

There are some exceptions to the rule requiring all ANC packets to be contiguous. For example, the EDH packet is a Type 1 ANC packet, but it always occurs immediately before the SAV (at the end of the HANC space) on a specified line in each field. The HANC space preceding the EDH packet can be empty or it can contain normal contiguous ANC packets. However, the space reserved for the EDH packet must be respected and cannot be overwritten when inserting a new ANC packet.

Some older equipment designed prior to the formalization of the ANC packet standards might not always generate contiguous ANC packets. For example, a video stream containing ANC packets inserted by older equipment can contain a few samples of blank video at the beginning of the ANC space preceding the first ANC packet. While generally not a problem when detecting and extracting packets, this is a problem for equipment designed to insert new ANC packets. ANC packets that do not start at the locations defined by the ANC standards are subject to being overwritten by equipment that inserts new ANC packets according to the rules defined by the standards.

ANC Packet Insertion Rules

The following procedure is used to locate free ANC space for insertion of a new packet.

1. Locate the beginning of an appropriate ANC space by finding a TRS symbol (EAV for HANC or SAV for VANC). For VANC space, the video line must also be in the vertical blanking interval.
2. If an ADF does not occur immediately (beginning the word after the TRS or the CS of a preceding ANC packet), then the entire remaining space is available. Any new ANC packet inserted in this space must begin immediately after the TRS symbol or the end of a preceding ANC packet.
3. If an ADF is found immediately, the DID value of the ANC packet is checked to determine if the ANC packet is an *end marker*, *start marker*, or *deletion marker*.
    a. If a *start marker* for non-conforming ANC data is found, test each word after the *start marker* until another ADF is found, then repeat step 3. If the end of the ANC space is reached before another ADF is found, repeat step 1.
    b. If an *end marker* is found, the area occupied by the *end marker* plus the remaining area in the ANC space is available.
    c. If a packet marked for deletion is found, then the area occupied by the packet marked for deletion is available. However, the ANC packet deletion rules must be obeyed.
4. If an ADF is found that is not a *start marker*, *end marker*, or *deletion marker*, then use the DC word to locate the end of the ANC packet. At the end of the packet, repeat step 2.

After free space is found, the following rules must be used to determine if a new ANC packet can be inserted.

1. The space available must be sufficient to hold the entire ANC packet. The new packet cannot overwrite the TRS symbol that ends the space. If the line is the one line per field where an EDH packet should occur, the space reserved for the EDH packet cannot be overwritten, even if there is no EDH packet present.
2. An *end marker* ANC packet can be replaced by a newly inserted conforming ANC packet or by a *start marker* packet for non-conforming ANC data.
3. If a non-conforming ANC packet is to be inserted, it must always be preceded by a *start marker* ANC packet and followed by an *end marker* ANC packet. Prior to insertion, it must be determined that there is sufficient space for the *start marker* packet, *end marker* packet, and the non-conforming ANC data.
4. If a new ANC packet replaces a packet marked for deletion, then the rules for ANC packet deletion, described in the next section, must be followed.

ANC Packet Deletion Rules

To delete an ANC packet, the DID word is simply changed to a value of 80_{HEX} and the checksum word of the packet is updated (Figure 6). The deleted ANC packet still has a valid DC value and occupies the same amount of space, maintaining the contiguity of packets in the ANC space.

It is possible to insert a new ANC packet in the space occupied by an ANC packet that has been marked for deletion. In doing so, the contiguity of the packets in the ANC space must be maintained. The newly inserted packet must not be larger than the deleted packet — unless that packet is the last one in the ANC space. If the inserted packet is smaller than the deleted packet, then a dummy packet must fill the remainder of the space not filled by the newly inserted packet in order to maintain contiguity. The dummy packet has a DID value of 80_{HEX}, the same as a packet marked for deletion. The minimum size of a dummy packet is seven words. Therefore, in order to replace a packet marked for deletion with a new ANC packet that is smaller, the new packet must be at least seven words smaller than the deleted packet in order to leave room for the dummy packet.

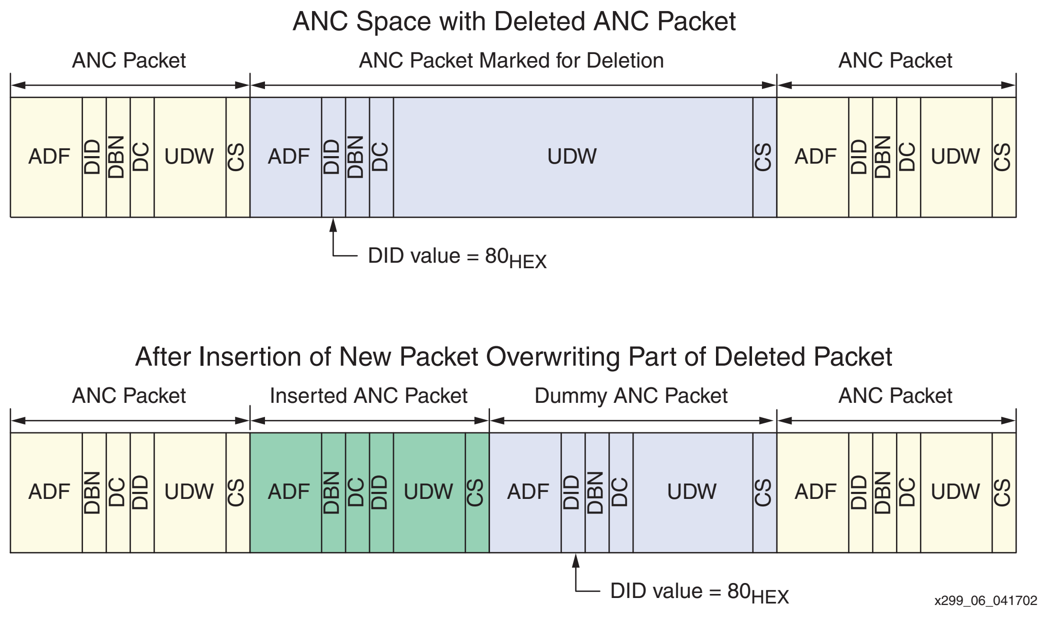


Figure 6: **Overwriting an ANC Packet Marked for Deletion**

Synchronous Switching Considerations

The standards recommend against inserting ANC packets into those areas of the field that can be affected by synchronous video switching. SMPTE RP-168[4] identifies a particular line in each video field where video-switching equipment should switch between synchronous video sources. Obviously, if a video stream is switched in the middle of an ANC packet, the packet will be lost. Therefore, the standards recommend certain "keep-out" areas where ANC packets are not recommended. Table 1 shows those keep-out areas for various common video standards. These areas are also noted on the NTSC ANC space diagram in Figure 5.

Table 1: **ANC Keep-Out Areas for Synchronous Switching**

Video Standard		**ANC Keep-Out Areas**	
Lines	*Sample Frequency*	*Line Numbers*	*Words (from-to)*
Standard NTSC		10 and 273	0 to 1439
525	13.5 MHz	11 and 274	1444 to 1711
Wide-screen NTSC		10 and 273	0 to 1919
525	18 MHz	11 and 274	1924 to 2283
Standard PAL		6 and 319	0 to 1439
625	13.5 MHz	7 and 320	1444 to 1723
Wide-screen PAL		6 and 319	0 to 1919
625	18 MHz	7 and 320	1924 to 2299

Error Detection and Handling (EDH)

The SMPTE Recommended Practice RP 165-1994[5] and the equivalent ITU standard ITU-R BT.1304[6] define an error detection protocol which is primarily designed for use with SDI, but can also be used with parallel digital video interfaces. The purpose of the error detection protocol is to allow detection of defective equipment and noisy connections, not to prevent loss of data due to errors. There is no retransmission protocol that allows the fields containing errors to be retransmitted.

The error detection protocol standards define a special type of ANC packet called the error detection and handling (EDH) packet. An EDH packet is generated and inserted into the video stream once per field at a specific position defined by the standards. The packet contains two cyclic redundancy code (CRC) checkwords calculated from the previous field. The EDH packet also contains three sets of error flags used to forward error detection information to help identify faulty equipment and noisy connections.

Two different CRC checkwords are calculated on a field of digital video. One CRC checkword is calculated on only the active samples of the field and the other is calculated on the full field (actually most of the field). Both checkwords are provided to allow error detection to remain intact on the active portion of the field, even when a piece of equipment inserts new data (such as ANC packets) into the inactive portion of the field without updating the full-field CRC checkword in the EDH packet. Generally, video equipment that modifies the video stream in any way should calculate new CRC checkwords and update the EDH packet. However, equipment not supporting the EDH protocol could modify the inactive portion of the video without updating the EDH packet.

Three sets of error flags are provided in the EDH packet to forward error detection information. One set is related to the active-picture CRC checkword. Another set is related to the full-field CRC checkword. The third set of error flags is used to provide error detection information based on evaluating all the ANC packet checksums in the field. This third set of flags is optional when implementing EDH packets.

CRC Checkword Calculations

Each of the CRC checkwords is calculated over a certain set of samples in a field. The starting and ending locations of these sample sets are specifically defined in the standards. These locations vary depending upon the video standard.

The standards also define the location of the EDH packet. The EDH packet location is immediately before the SAV on a specific line in each field.

Figure 8 through Figure 13 show the starting and ending locations for the samples sets of each CRC checkword and the EDH packet position for various video standards.

Each CRC checkword is a 16-bit value calculated using the CRC-CCITT polynomial generation method. Figure 7 shows the equation for the CRC calculation and a conceptual logic diagram of how the CRC value is calculated.

The standards require that any the values between $3FC_{HEX}$ and $3FE_{HEX}$ must be regarded as equaling $3FF_{HEX}$ for the purposes of the CRC calculation. This only affects the CRC generator and the actual value in the video stream does not need to be modified. This is done for compatibility between 8-bit and 10-bit video equipment.

The active-picture CRC only includes those samples in the active portion of the lines indicated in the drawings. The samples in the horizontal blanking interval of each line are not included in the active-picture CRC calculation.

In the NTSC video standards, lines 20 and 283 are not included in the active-picture CRC calculation. These lines are technically in the active portion of the field; the "V" bit in the TRS symbols on those lines is zero, indicating active video lines. Some video equipment manufacturers consider these two lines to be the last lines of the vertical blanking interval. Probably due to this ambiguity, the active-picture CRC calculations do not include these two lines. See **XAPP248** for a more detailed discussion of the active/inactive status of these two lines.

The full-field CRC calculation includes all samples, both active and inactive, from the starting point to the ending point shown in Figure 7. The full-field CRC includes those active samples that are also included in the active picture CRC calculation. The full-field CRC calculation does not include the line in each field defined by SMPTE RP 168 as the synchronous switching line nor the line immediately following. This is to prevent synchronous switching events from corrupting the CRC calculation. The line immediately before the synchronous switching line contains the EDH packet for the previous field. This line is also not included in the full-field CRC calculation. The following figures are included in this application note:

Figure 7: **CRC Calculations**

Figure 8: **NTSC 13.5 MHz 4:2:2 CRC Calculations and EDH Packet Positions**

Figure 9: **NTSC 18 MHz 4:2:2 CRC Calculations and EDH Packet Position**

Figure 10: **NTSC 4:4:4:4 CRC Calculations and EDH Packet Positions**

Figure 11: **PAL 13.5 MHz 4:2:2 CRC Calculations and EDH Packet Positions**

Figure 12: **PAL 18 MHz 4:2:2 CRC Calculations and EDH Packet Positions**

Figure 13: **PAL 4:4:4:4 CRC Calculations and EDH Packet Positions**

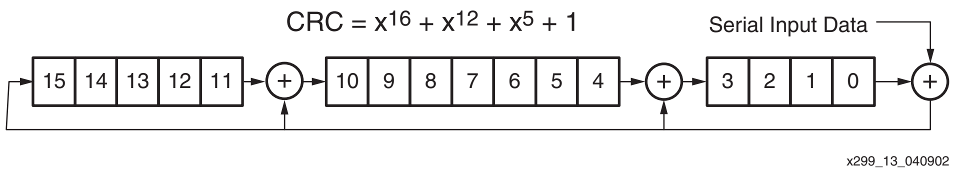


Figure 7: **CRC Calculations**

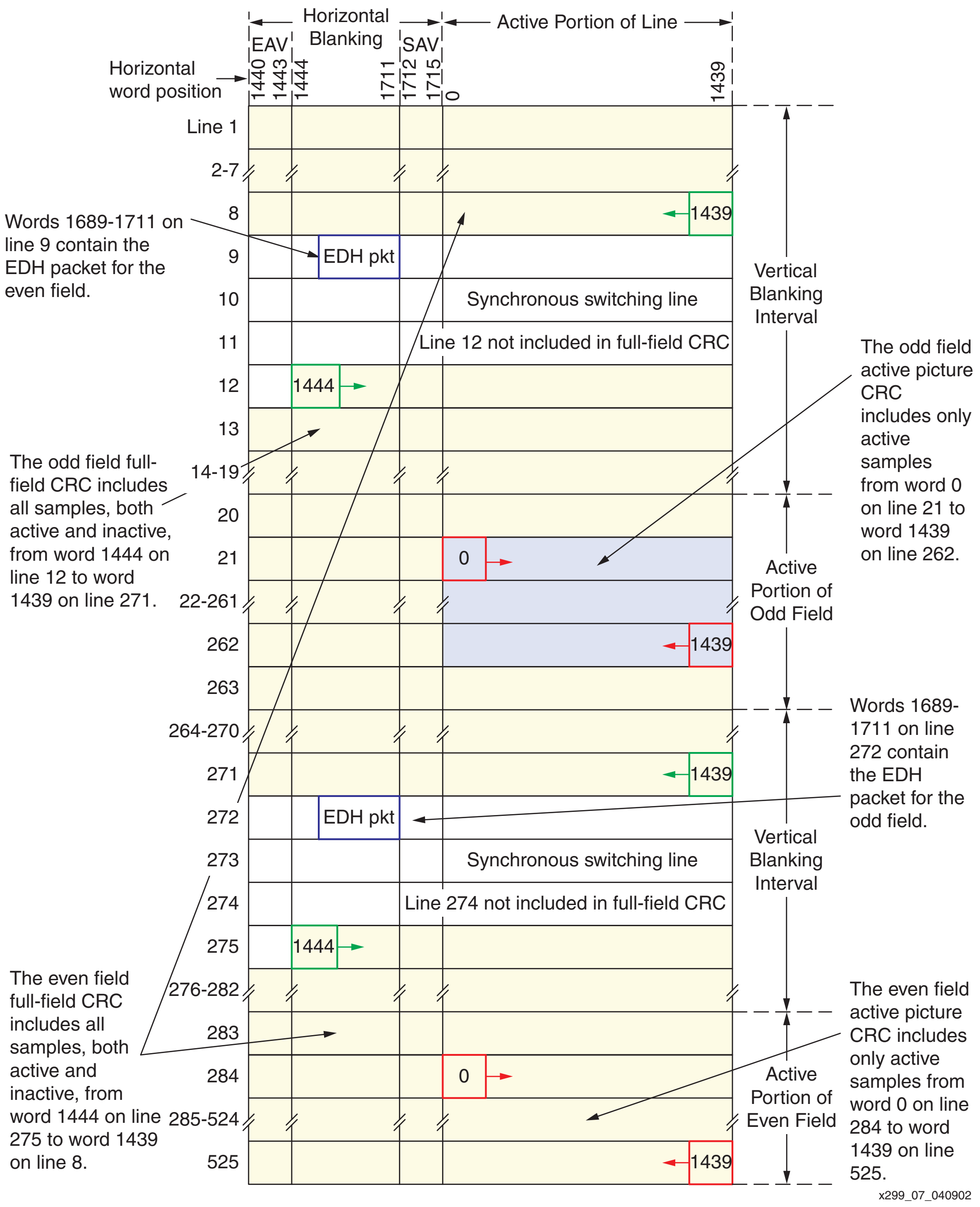

Horizontal Blanking
EAV
SAV
Active Portion of Line
Horizontal word position
1440
1443
1444
1711
1712
1715
0
1439
Line 1
2-7
8
9
10
11
12
13
14-19
20
21
22-261
262
263
264-270
271
272
273
274
275
276-282
283
284
285-524
525
1439
EDH pkt
Synchronous switching line
Line 12 not included in full-field CRC
1444
0
1439
1439
EDH pkt
Synchronous switching line
Line 274 not included in full-field CRC
1444
0
1439
Words 1689-1711 on line 9 contain the EDH packet for the even field.
The odd field full-field CRC includes all samples, both active and inactive, from word 1444 on line 12 to word 1439 on line 271.
The even field full-field CRC includes all samples, both active and inactive, from word 1444 on line 275 to word 1439 on line 8.
Vertical Blanking Interval
Active Portion of Odd Field
Vertical Blanking Interval
Active Portion of Even Field
The odd field active picture CRC includes only active samples from word 0 on line 21 to word 1439 on line 262.
Words 1689-1711 on line 272 contain the EDH packet for the odd field.
The even field active picture CRC includes only active samples from word 0 on line 284 to word 1439 on line 525.
x299_07_040902

Figure 8: **NTSC 13.5 MHz 4:2:2 CRC Calculations and EDH Packet Positions**

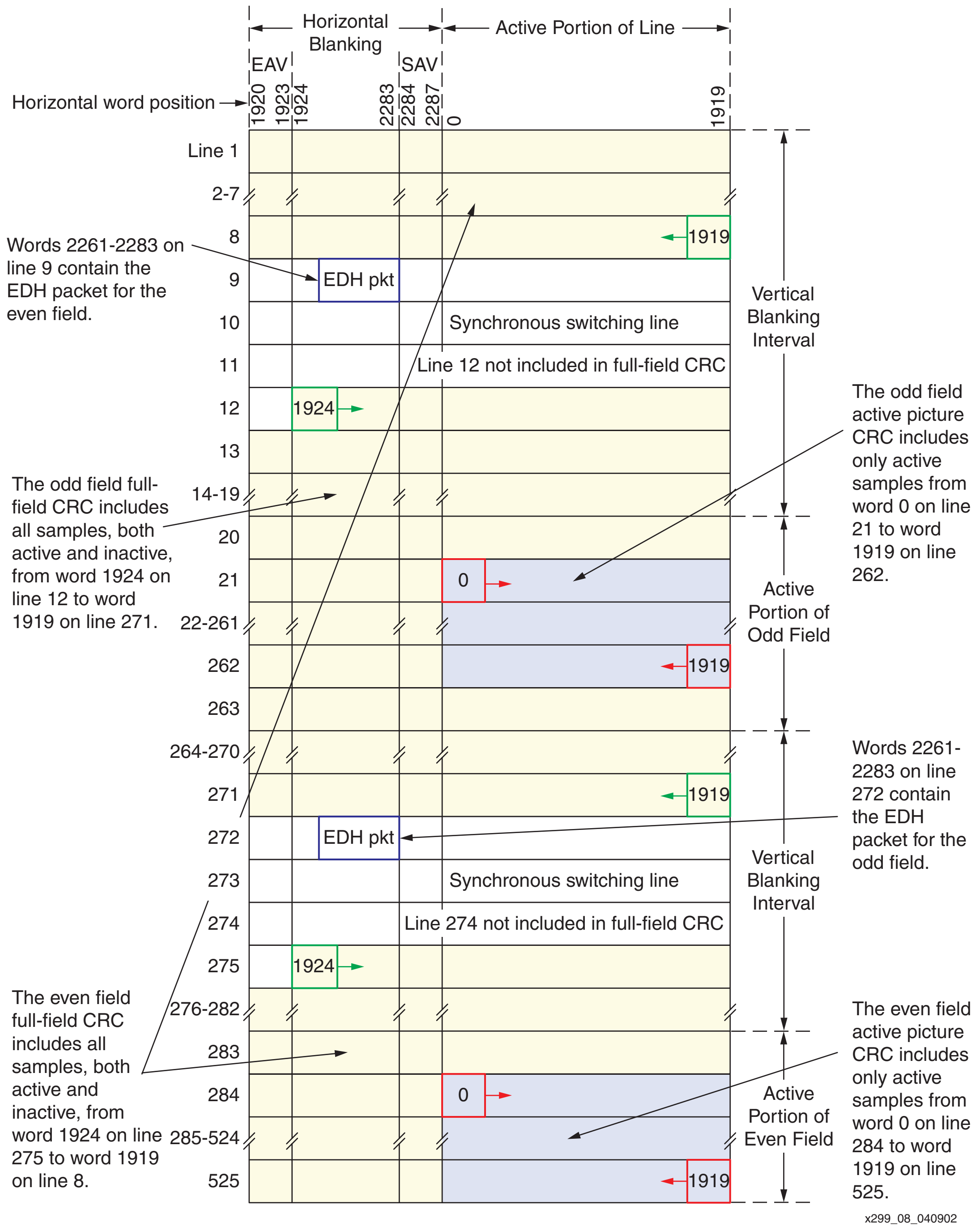

Horizontal Blanking
Active Portion of Line
EAV
SAV
Horizontal word position
1920
1923
1924
2283
2284
2287
0
1919
Line 1
2-7
8
9
10
11
12
13
14-19
20
21
22-261
262
263
264-270
271
272
273
274
275
276-282
283
284
285-524
525
Words 2261-2283 on line 9 contain the EDH packet for the even field.
EDH pkt
Synchronous switching line
Line 12 not included in full-field CRC
1924
The odd field full-field CRC includes all samples, both active and inactive, from word 1924 on line 12 to word 1919 on line 271.
Vertical Blanking Interval
The odd field active picture CRC includes only active samples from word 0 on line 21 to word 1919 on line 262.
Active Portion of Odd Field
Words 2261-2283 on line 272 contain the EDH packet for the odd field.
Line 274 not included in full-field CRC
The even field full-field CRC includes all samples, both active and inactive, from word 1924 on line 275 to word 1919 on line 8.
The even field active picture CRC includes only active samples from word 0 on line 284 to word 1919 on line 525.
Active Portion of Even Field
x299_08_040902

Figure 9: **NTSC 18 MHz 4:2:2 CRC Calculations and EDH Packet Position**

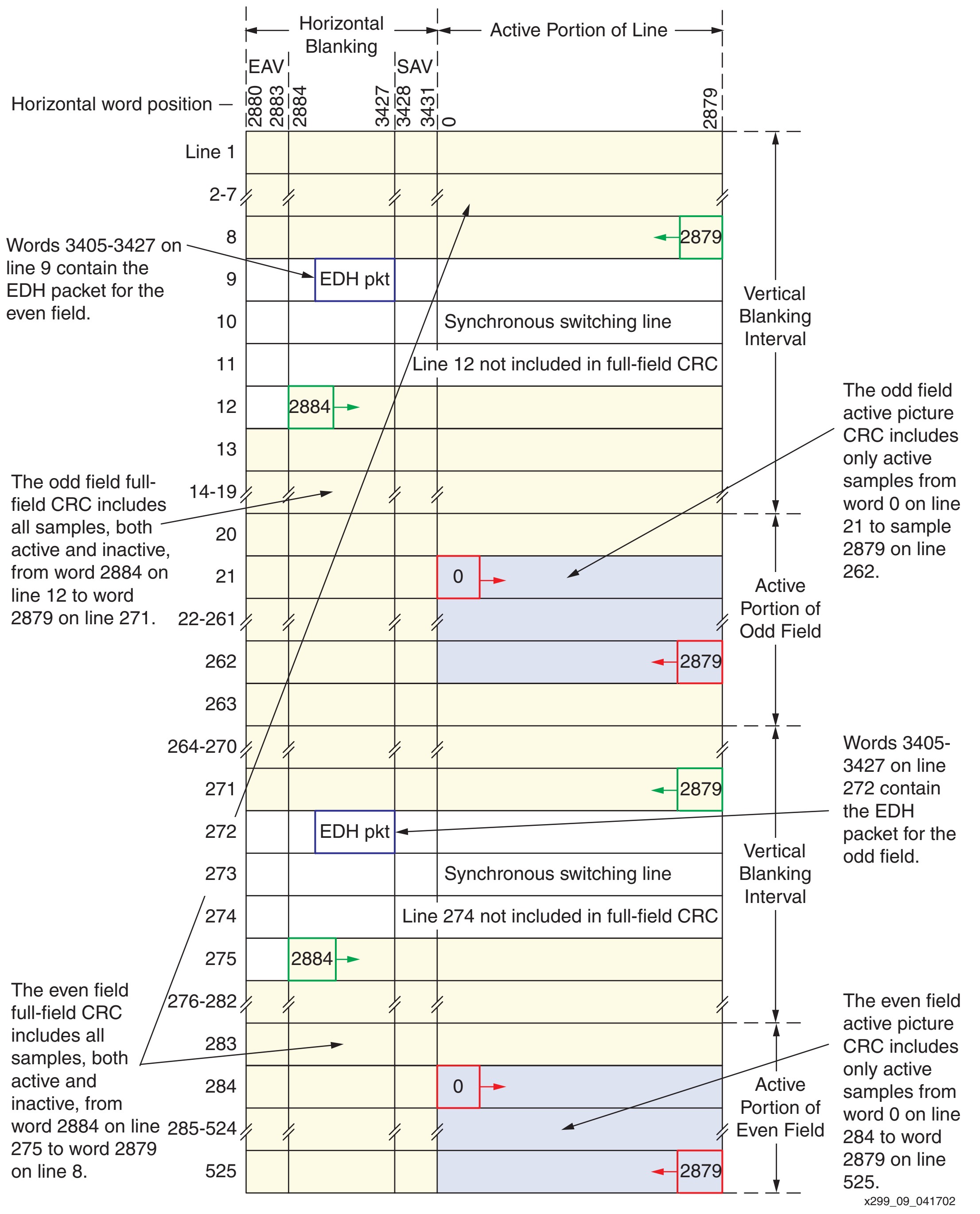

Horizontal Blanking
Active Portion of Line
EAV
SAV
Horizontal word position
2880
2883
2884
3427
3428
3431
0
2879
Line 1
2-7
8
9
10
11
12
13
14-19
20
21
22-261
262
263
264-270
271
272
273
274
275
276-282
283
284
285-524
525
EDH pkt
Synchronous switching line
Line 12 not included in full-field CRC
Line 274 not included in full-field CRC
Words 3405-3427 on line 9 contain the EDH packet for the even field.
The odd field full-field CRC includes all samples, both active and inactive, from word 2884 on line 12 to word 2879 on line 271.
The even field full-field CRC includes all samples, both active and inactive, from word 2884 on line 275 to word 2879 on line 8.
Vertical Blanking Interval
Active Portion of Odd Field
Active Portion of Even Field
The odd field active picture CRC includes only active samples from word 0 on line 21 to sample 2879 on line 262.
Words 3405-3427 on line 272 contain the EDH packet for the odd field.
The even field active picture CRC includes only active samples from word 0 on line 284 to word 2879 on line 525.
x299_09_041702

Figure 10: **NTSC 4:4:4:4 CRC Calculations and EDH Packet Positions**

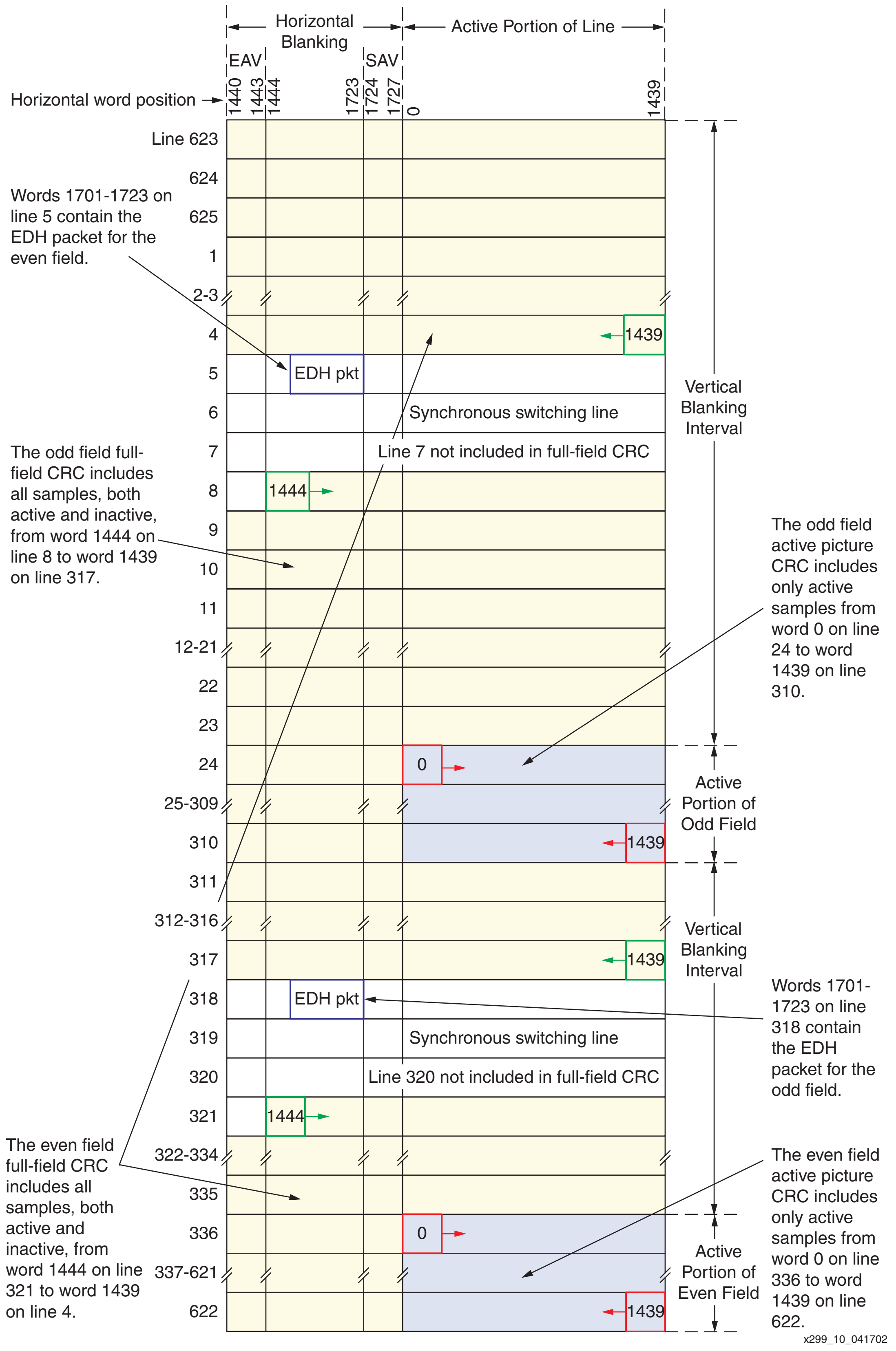

Horizontal Blanking
Active Portion of Line
EAV
SAV
Horizontal word position
1440
1443
1444
1723
1724
1727
0
1439
Line 623
624
625
1
2-3
4
5
6
7
8
9
10
11
12-21
22
23
24
25-309
310
311
312-316
317
318
319
320
321
322-334
335
336
337-621
622
Words 1701-1723 on line 5 contain the EDH packet for the even field.
EDH pkt
Synchronous switching line
Line 7 not included in full-field CRC
1444
1439
The odd field full-field CRC includes all samples, both active and inactive, from word 1444 on line 8 to word 1439 on line 317.
Vertical Blanking Interval
The odd field active picture CRC includes only active samples from word 0 on line 24 to word 1439 on line 310.
Active Portion of Odd Field
Words 1701-1723 on line 318 contain the EDH packet for the odd field.
Line 320 not included in full-field CRC
The even field full-field CRC includes all samples, both active and inactive, from word 1444 on line 321 to word 1439 on line 4.
The even field active picture CRC includes only active samples from word 0 on line 336 to word 1439 on line 622.
Active Portion of Even Field
x299_10_041702

Figure 11: **PAL 13.5 MHz 4:2:2 CRC Calculations and EDH Packet Positions**

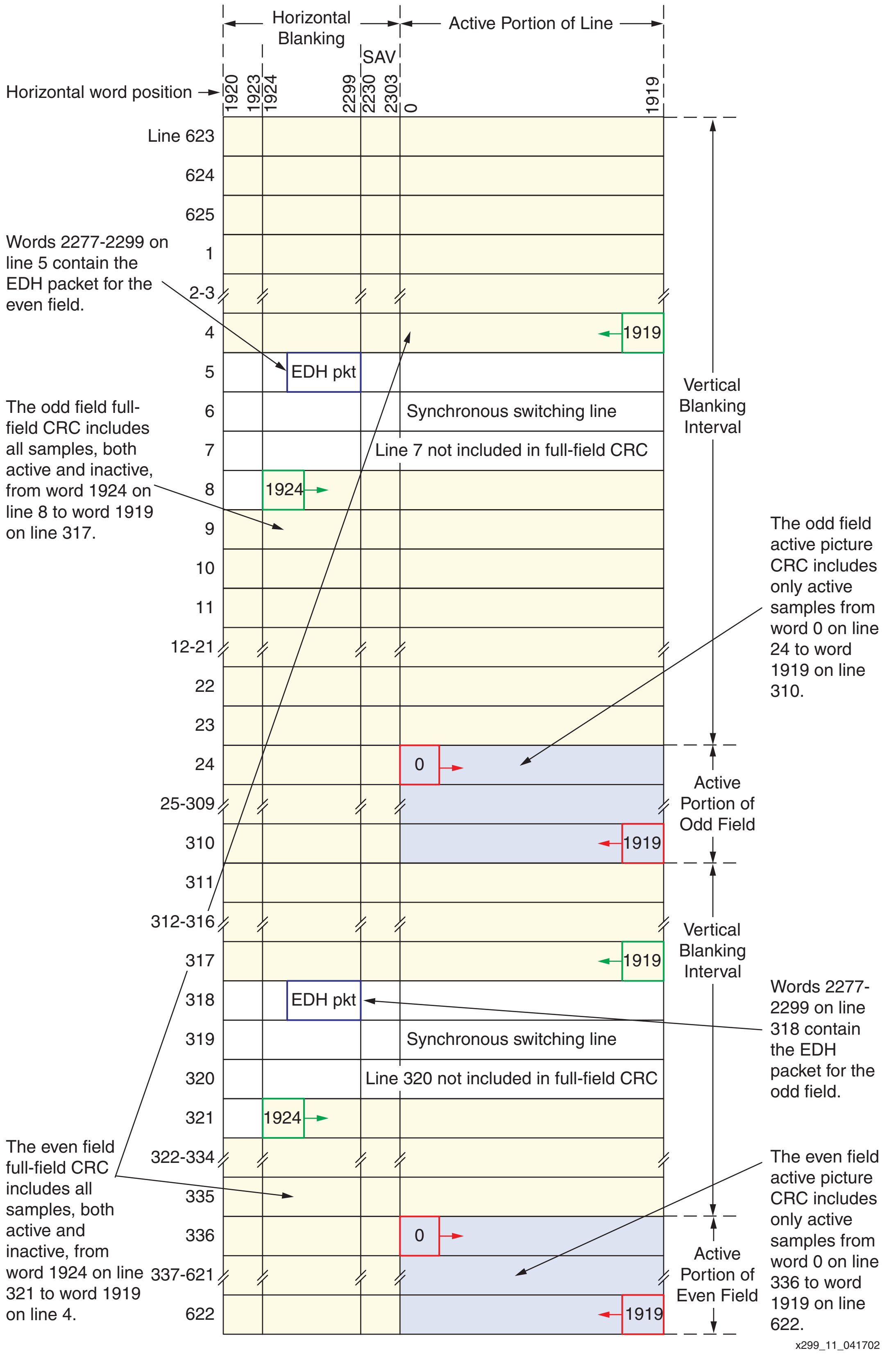

Horizontal Blanking
Active Portion of Line
SAV
Horizontal word position
1920
1923
1924
2299
2230
2303
0
1919
Line 623
624
625
1
2-3
4
5
6
7
8
9
10
11
12-21
22
23
24
25-309
310
311
312-316
317
318
319
320
321
322-334
335
336
337-621
622
Words 2277-2299 on line 5 contain the EDH packet for the even field.
EDH pkt
Synchronous switching line
Line 7 not included in full-field CRC
The odd field full-field CRC includes all samples, both active and inactive, from word 1924 on line 8 to word 1919 on line 317.
Vertical Blanking Interval
The odd field active picture CRC includes only active samples from word 0 on line 24 to word 1919 on line 310.
Active Portion of Odd Field
Words 2277-2299 on line 318 contain the EDH packet for the odd field.
Line 320 not included in full-field CRC
The even field full-field CRC includes all samples, both active and inactive, from word 1924 on line 321 to word 1919 on line 4.
The even field active picture CRC includes only active samples from word 0 on line 336 to word 1919 on line 622.
Active Portion of Even Field
x299_11_041702

Figure 12: **PAL 18 MHz 4:2:2 CRC Calculations and EDH Packet Positions**

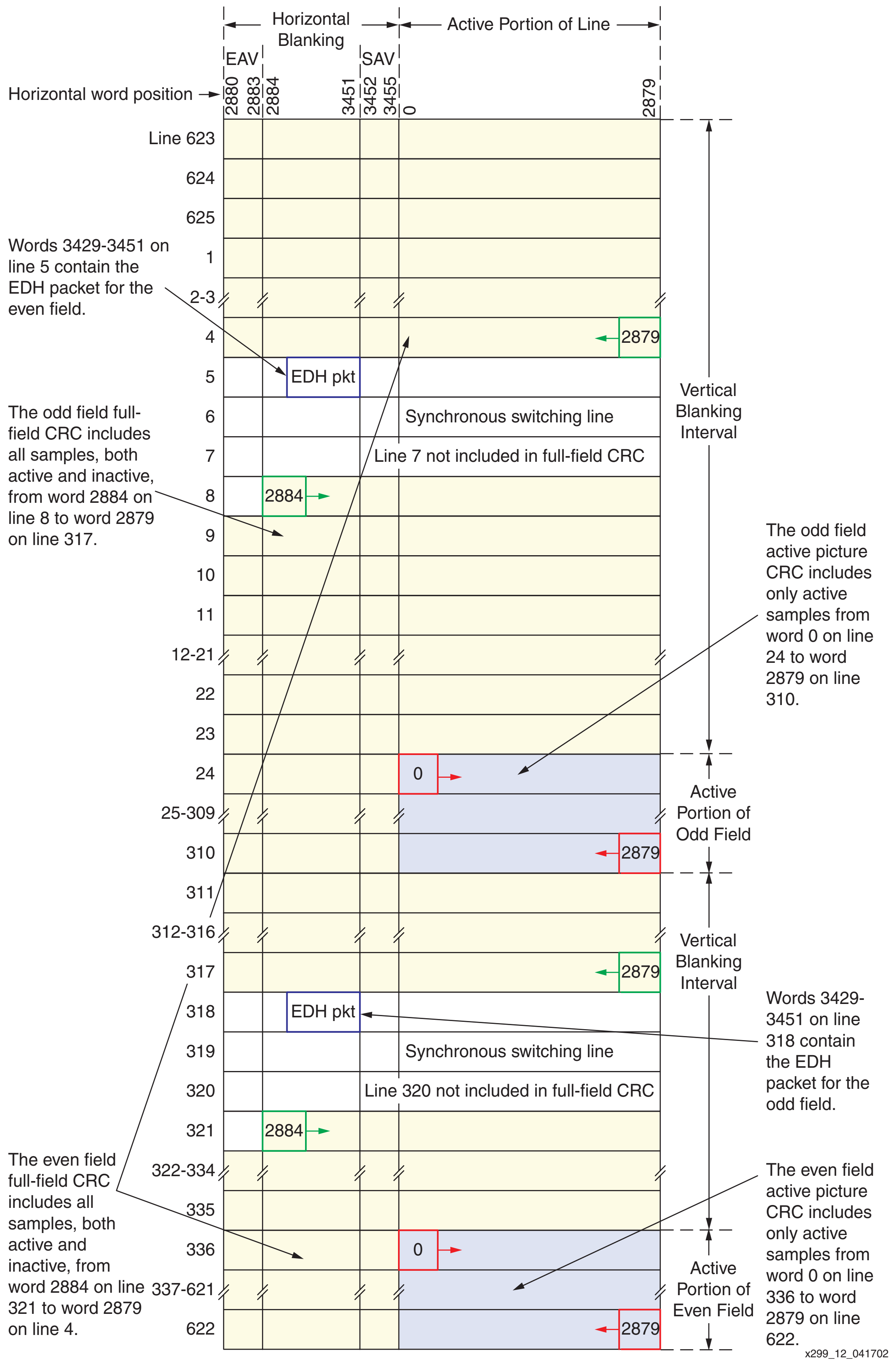

Horizontal Blanking
Active Portion of Line
EAV
SAV
Horizontal word position
2880
2883
2884
3451
3452
3455
0
2879
Line 623
624
625
1
2-3
4
5
6
7
8
9
10
11
12-21
22
23
24
25-309
310
311
312-316
317
318
319
320
321
322-334
335
336
337-621
622
Words 3429-3451 on line 5 contain the EDH packet for the even field.
EDH pkt
Synchronous switching line
Line 7 not included in full-field CRC
The odd field full-field CRC includes all samples, both active and inactive, from word 2884 on line 8 to word 2879 on line 317.
Vertical Blanking Interval
The odd field active picture CRC includes only active samples from word 0 on line 24 to word 2879 on line 310.
Active Portion of Odd Field
Words 3429-3451 on line 318 contain the EDH packet for the odd field.
Line 320 not included in full-field CRC
The even field full-field CRC includes all samples, both active and inactive, from word 2884 on line 321 to word 2879 on line 4.
The even field active picture CRC includes only active samples from word 0 on line 336 to word 2879 on line 622.
Active Portion of Even Field
x299_12_041702

Figure 13: **PAL 4:4:4:4 CRC Calculations and EDH Packet Positions**

Error Flags

An EDH packet contains three sets of error flags. One set is associated with the active picture (AP) CRC, one set is associated with the full field (FF) CRC, and one is associated with ANC packet errors. Each set of error flags contains five flags as described below.

edh — Error Detected Here

Any piece of equipment detecting a difference between the CRC value it calculates for the previous field and the CRC checkword located in the EDH packet sets (flag = 1) the *edh* flag. The ancillary data *edh* flag is set if a checksum error is detected in at least one ANC packet in the previous field.

eda — Error Detected Already

This flag indicates that some upstream piece of equipment detected an error. A video device processing an EDH packet having the *edh* flag set by the upstream device must set the *eda* flag in the packet and clear the *edh* flag unless it, too, detects an error. (See Figure 14.)

idh — Internal Error Detected Here

Any piece of equipment can assert the *idh* flag to indicate that some internal processing error, unrelated to the serial video transmission, has occurred. The *idh* flag is provided as a signaling mechanism to allow video equipment to indicate the occurrence of internal errors. These internal errors can be anything unrelated to the actual video stream, the detection of an overheating condition, for example.

ida — Internal Error Detected Already

This flag indicates that some upstream piece of equipment detected an internal error. A video device processing an EDH packet having the *idh* flag set by the upstream device must set the *ida* flag and clear the *idh* flag unless it, too, detects an internal error.

ues — Unknown Error Status

This flag indicates that the video stream was received from equipment not supporting the EDH standard. For example, a device that receives a video stream without any EDH packets can generate and insert EDH packets into the video stream. It should, however, set the *ues* flag in the packets it creates to signify that the video stream was not previously protected by the EDH error detection protocol.

The flag pairs, *edh/eda* and *idh/ida*, can be used to track down faulty video equipment in the serial transmission chain. For example, if the *eda* flag is set at any location, then it is known that some upstream piece of equipment detected an error. If the errors are occurring repeatedly, each piece of video equipment can be checked, starting with the downstream device and moving upstream to see where the *eda* flag changes to an *edh* flag. The connection or piece of equipment prior to the device asserting the *edh* flag is suspect.

The EDH standards allow video equipment to implement only some or all of the defined error flags. If a piece of equipment does not support a particular flag, it must clear the flag to zero.

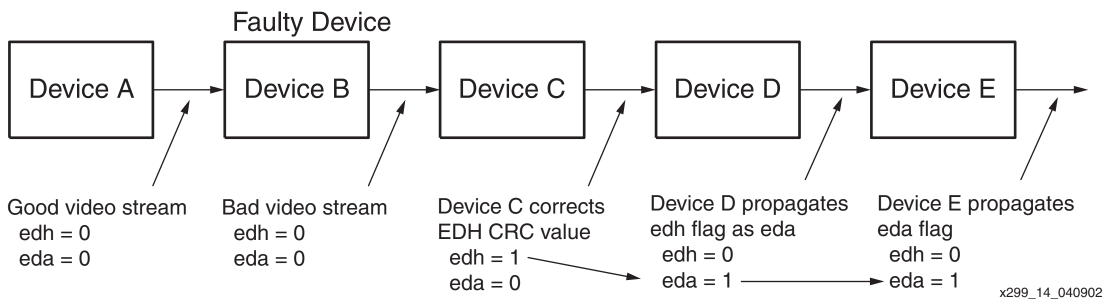


Figure 14: **Error Flag Forwarding**

EDH Packet Format

The EDH packet has the same format as a standard Type 1 ANC packet. The format of an EDH packet is shown in Figure 15.

Each CRC value has an associated valid bit. The standards allow implementations of the EDH protocol where only one of the two CRC values is calculated. A CRC value that is not calculated is considered to be invalid and must have its "V" bit cleared to a zero.

Word Contents	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Ancillary Data Flag, Word 1 (000_{HEX})	0	0	0	0	0	0	0	0	0	0
Ancillary Data Flag, Word 2 ($3FF_{HEX}$)	1	1	1	1	1	1	1	1	1	1
Ancillary Data Flag, Word 3 ($3FF_{HEX}$)	1	1	1	1	1	1	1	1	1	1
Data ID ($1F4_{HEX}$)	0	1	1	1	1	1	0	1	0	0
Block Number	1	0	0	0	0	0	0	0	0	0
Data Count (16 Words of User Data)	0	1	0	0	0	1	0	0	0	0
Active-picture CRC bits [5:0]	$\overline{P}$	P	ap5	ap4	ap3	ap2	ap1	ap0	0	0
Active-picture CRC bits [11:6]	$\overline{P}$	P	ap11	ap10	ap9	ap8	ap7	ap6	0	0
Active-picture CRC bits [15:12]	$\overline{P}$	P	V	0	ap15	ap14	ap13	ap12	0	0
Full-field CRC bits [5:0]	$\overline{P}$	P	ff5	ff4	ff3	ff2	ff1	ff0	0	0
Full-field CRC bits [11:6]	$\overline{P}$	P	ff11	ff10	ff9	ff8	ff7	ff6	0	0
Full-field CRC bits [15:12]	$\overline{P}$	P	V	0	ff15	ff14	ff13	ff12	0	0
Ancillary Data Error Flags	$\overline{P}$	P	0	ues	ida	idh	eda	edh	0	0
Active-picture Error Flags	$\overline{P}$	P	0	ues	ida	idh	eda	edh	0	0
Full-field Error Flags	$\overline{P}$	P	0	ues	ida	idh	eda	edh	0	0
Reserved Words (7 total)	1	0	0	0	0	0	0	0	0	0
Checksum	$\overline{S8}$	S8	S7	S6	S5	S4	S3	S2	S1	S0

Notes:
1) P is an even parity bit for bits b7 through b0 and is located in b8.
Words containing a P bit in b8, also have the inverse of b8 located in b9.
2) Each CRC value has an associated valid bit (V). If the CRC value is valid, V is set to 1.

x299_15_041802

Figure 15: **EDH Packet Format**

Reference Design

The reference design for this application note is available in both VHDL and Verilog code on the Xilinx FTP site at **ftp://ftp.xilinx.com/pub/applications/xapp/xapp299.zip**.

ANC and EDH Processor

Figure 16 shows a block diagram of a complete ANC and EDH processor. The video decoder block is described in **XAPP625**. This video decoder processes the incoming video stream to determine the video standard and to provide timing information about the video stream such as the current horizontal and vertical positions and locations of TRS symbols, ANC packets, and EDH packets.

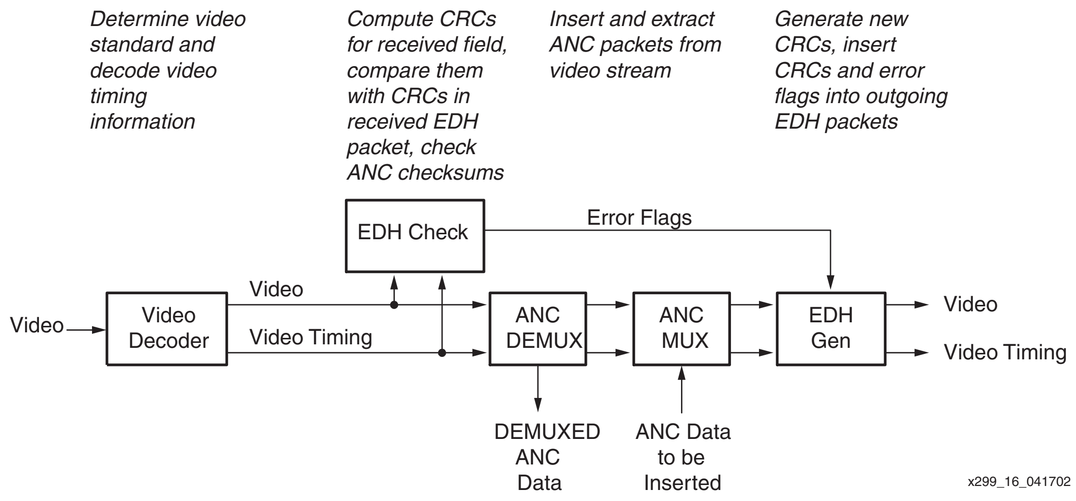

Determine video standard and decode video timing information
Compute CRCs for received field, compare them with CRCs in received EDH packet, check ANC checksums
Insert and extract ANC packets from video stream
Generate new CRCs, insert CRCs and error flags into outgoing EDH packets
EDH Check
Error Flags
Video
Video Decoder
Video
Video Timing
ANC DEMUX
ANC MUX
EDH Gen
Video
Video Timing
DEMUXED ANC Data
ANC Data to be Inserted
x299_16_041702

Figure 16: **ANC and EDH Processor Block Diagram**

The *anc_edh_processor* design implements the complete ANC and EDH processor, including the video decoder. The various blocks that make up this processor are described below.

edh_check Module

This section calculates CRC values for the field, finds the EDH packets in the video stream, and compares the CRC values in the EDH packets with the calculated values. It also verifies the checksum values of every ANC packet in the video stream. Based on these checks, error flags are generated and provided to the *edh_gen* module for transmission to downstream equipment in the next EDH packet. The module maintains a running count of the number of fields with errors.

If the input video stream is known not to contain EDH packets, the *receive_mode* input of this module can be negated. This prevents the module from generating errors for each missing EDH packet.

The module has error flag inputs for any EDH flags not internally generated (*idh* flag). These inputs can be asserted by another module to send error information in the EDH packets.

The module has an error counter flag enable input for each of the various error condition flags. This allows selection of which error conditions will increment the counter.

The *edh_check* section captures and outputs the flags from each EDH packet. These outputs can be used to determine what error conditions are being received in the EDH packets. The module also generates and outputs error signals related to the reception of the EDH packet itself. The parity, format, and checksum of the EDH packet are checked and separate error flags provided to indicate each of these error conditions. Another error flag is asserted when the EDH packet is missing from the video stream.

anc_demux Module

This module de-multiplexes ANC packets from the video stream. The module searches for and de-multiplex up to four different ANC packet types. The module has four sets of DID/SDID inputs used to specify which ANC packet types are to be de-multiplexed. The four sets of DID/SDID inputs are compared against the DID and SDID words in the ANC packets and matching packets are de-multiplexed. The module decodes the input DID values to determine whether to also use the SDID value in the matching process. The SDID value is only used for Type 2 ANC packets.

Each DID/SDID input set has an enable input. If the enable is Low, the DID/SDID pair is not compared with the incoming ANC packets.

Also associated with each DID/SDID input pair is a *del_pkt* input. If this input is asserted and the corresponding enable input is asserted, packets matching the DID/SDID pair will be demultiplexed and marked for deletion in the video stream. The module will change the DID value of the packet in the video stream to mark it for deletion and will calculate a new checksum value for the packet. The modified packet replaces the original packet in the video stream. The modified video stream is sent out on the module's *vid_out* port. The de-multiplexed packet is sent out on the *data_out* port with its original DID and checksum values.

The module has a *data_out_valid* signal indicating when a de-multiplexed ANC packet is being sent out the *data_out* port. This signal becomes asserted when the DID word is available on the *data_out* port and stays asserted through the checksum word. This signal is not asserted during the three words of the ADF.

In addition to the *data_out_valid* signal, the module also provides a number of output signals indicating what is present on the *data_out* port. A 2-bit *match_code* value indicates which one of the four input DID/SDID pairs matched the de-multiplexed packet. A set of output signals (*did*, *sdid*, *dbn*, *dc*, *udw*, and *cs*) indicate which word of the packet is currently available on the *data_out* port.

anc_mux **Module**

This module multiplexes new ANC packets into the video stream.

When the module is ready to accept new packet data, it asserts the *pkt_in_empty* output. A new packet is formed by writing the DID, SDID/DBN, and DC words into the module's internal registers. These values are 8-bit values and must be placed on the eight least significant bits of the module's *data_in* port. Each word is loaded into the module by asserting the associated load signal (*ld_did*, *ld_dbn*, and *ld_dc*). The *ld_dbn* signal is used to load either the SDID word or the DBN word, depending on the type of packet.

The UDW words of the packet are written by placing the 10-bit words on the *data_in* port, placing the word number (0 for the first word, 1 for the second word, etc.) on the *udw_wr_adr* port, and asserting the *ld_udw* input. If the packet uses eight-bit UDW words with an even parity bit in bit 8 and the complement of the parity bit in bit 9, the module can automatically calculate and insert bits 8 and 9. This is done if the *calc_udw_parity* signal is asserted as the words are written to the module.

After the entire packet has been written to the module, the *pkt_rdy_in* signal must be asserted. At the same time, the *hanc_pkt* and *vanc_pkt* inputs must also be set appropriately to indicate whether the packet is to be inserted in HANC space, VANC space, or either. The module will respond immediately by negating the *pkt_in_empty* signal. No new information can be written to the module until the *pkt_in_empty* signal is reasserted.

After *pkt_rdy_in* is asserted, the module will look for room in the specified ANC data spaces large enough to accommodate the packet. When an appropriate space is found, the packet is inserted. The module creates the ADF and calculates and inserts a checksum word for the packet.

This module is not designed to overwrite a packet marked for deletion. The module will, however, overwrite an *end marker* packet.

edh_gen **Module**

The *edh_check* module calculates CRC values on the incoming video stream and compares them with the CRC values in the incoming EDH packets. However, the ANC MUX and DEMUX modules can modify the video stream, invalidating the CRC values in the EDH packets.

The *edh_gen* module calculates new CRC values and uses them, along with the error flags generated by the *edh_check* module, to update the contents of the EDH packets in the video stream. If no EDH packets are present in the video stream, the module generates new EDH packets and inserts them at the appropriate places in the video stream.

edh_processor **Module**

If the ANC MUX and DEMUX functions are not used, the *edh_processor* module is an efficient EDH-only processor. It is more efficient than simply combining the *edh_check* and *edh_gen* modules. This design uses the same submodules that make up the *edh_check* and *edh_gen* modules. However, only one CRC calculation is done since the video stream is not subject to modification by the ANC MUX and DEMUX processes. The CRC calculation done on the input video stream is valid for the output video stream.

The *edh_processor* module has the same inputs as the *edh_check* module.

Results

Table 2 shows the results after place and route of the reference design. The *anc_edh_processor* results include the *video_decode* module from **XAPP625** and both the *anc_mux* and *anc_demux* function. The sizes of the *anc_mux* and *anc_demux* modules are shown separately so that an estimate can be made of how much smaller the *anc_edh_processor* would be with either of them removed. The *edh_processor* results include the size of the *video_decode* module from **XAPP625**.

The *anc_mux* module contains a RAM to store the user data words of the ANC packet. The module contains code to allow the RAM to be implemented as either distributed RAM or block RAM. Results for the *anc_edh_processor* and the *anc_mux* are given with both block RAM and distributed RAM. The module's UDW RAM fits in one block RAM.

The Virtex-II results were achieved when the design was constrained to run at 54 MHz, allowing it to support the fastest SDI bit-rate. The Spartan-II results were achieved when the design was constrained to run at 27 MHz, allowing support for the most commonly used 270 Mb/s SDI bit-rate.

All results were obtained using the Verilog versions of the designs with Xilinx ISE version 4.1i using XST as the synthesis tool. Results using the VHDL files are not shown but are essentially identical. Virtex-II results are for a -5 speed grade device. Spartan-II design results are for a -6 speed grade device.

Table 2: **Reference Design Results**

	Virtex-II (-5 Speed Grade)			Spartan-II (-6 Speed Grade)		
Design Name	**Size LUTs**	**Size FFs**	**Speed**	**Size LUTs**	**Size FFs**	**Speed**
`anc_edh_processor.v` (distributed RAM)	1496	856	55 MHz	1549	856	30 MHz
`anc_edh_processor.v` (block RAM)	1326	846	60 MHz	1326	846	30 MHz
`anc_demux.v`	136	179	70 MHz	136	179	50 MHz
`anc_mux.v` (distributed RAM)	448	105	80 MHz	492	105	45 MHz
`anc_mux.v` (block RAM)	260	95	90 MHz	268	95	50 MHz
`edh_processor.v`	810	537	85 MHz	810	537	55 MHz

Conclusion

In an SDI transmission link, digital video is normally preprocessed prior to transmission to insert error detection checkwords and ancillary data. At the receiving end of the SDI link, the data is again processed to check for transmission errors and possibly to extract the ancillary data.

The application note demonstrates how to implement the EDH and ANC packet processors for an SDI link using Xilinx FPGAs.

References

1. SMPTE 291M-1998, SMPTE Standard for Television - Ancillary Data Packet and Space Formatting (The Society of Motion Picture and Television Engineers). The SMPTE standards referenced in this application note can purchased at the SMPTE web site: **http://www.smpte.org**.
2. ITU-R BT.1364, Format of Ancillary Data Signals Carried in Digital Component Studio Interfaces (International Telecommunication Union). The ITU-R BT.1364 standard can be purchased from the International Telecommunication Union at: **http://www.itu.int/itudoc/itu-r/rec/bt/**.
3. SMPTE 272M-1994, SMPTE Standard for Television - Formatting AES/EBU Audio and Auxiliary Data into Digital Video Ancillary Data Space (The Society of Motion Picture and Television Engineers).
4. RP 168-1993, SMPTE Recommended Practice - Definition of Vertical Interval Switching Point for Synchronous Video Switching (The Society of Motion Picture and Television Engineers).
5. RP 165-1994, SMPTE Recommended Practice - Error Detection Checkwords and Status Flags for Use in Bit-Serial Digital Interfaces for Television (The Society of Motion Picture and Television Engineers).
6. ITU-R BT.1304, Checksum for Error Detection and Status Information in Interfaces Conforming with Recommendations ITU-R BT.656 and ITU-R BT.799 (International Telecommunication Union).

Appendix A Additional Reference Design Information

edh_processor Module

The *edh_processor* contains the *video_decode* module from XAPP625 plus the modules to do CRC checking on the input video stream, ANC packet checksum checking, and outgoing EDH packet generation (Figure 17).

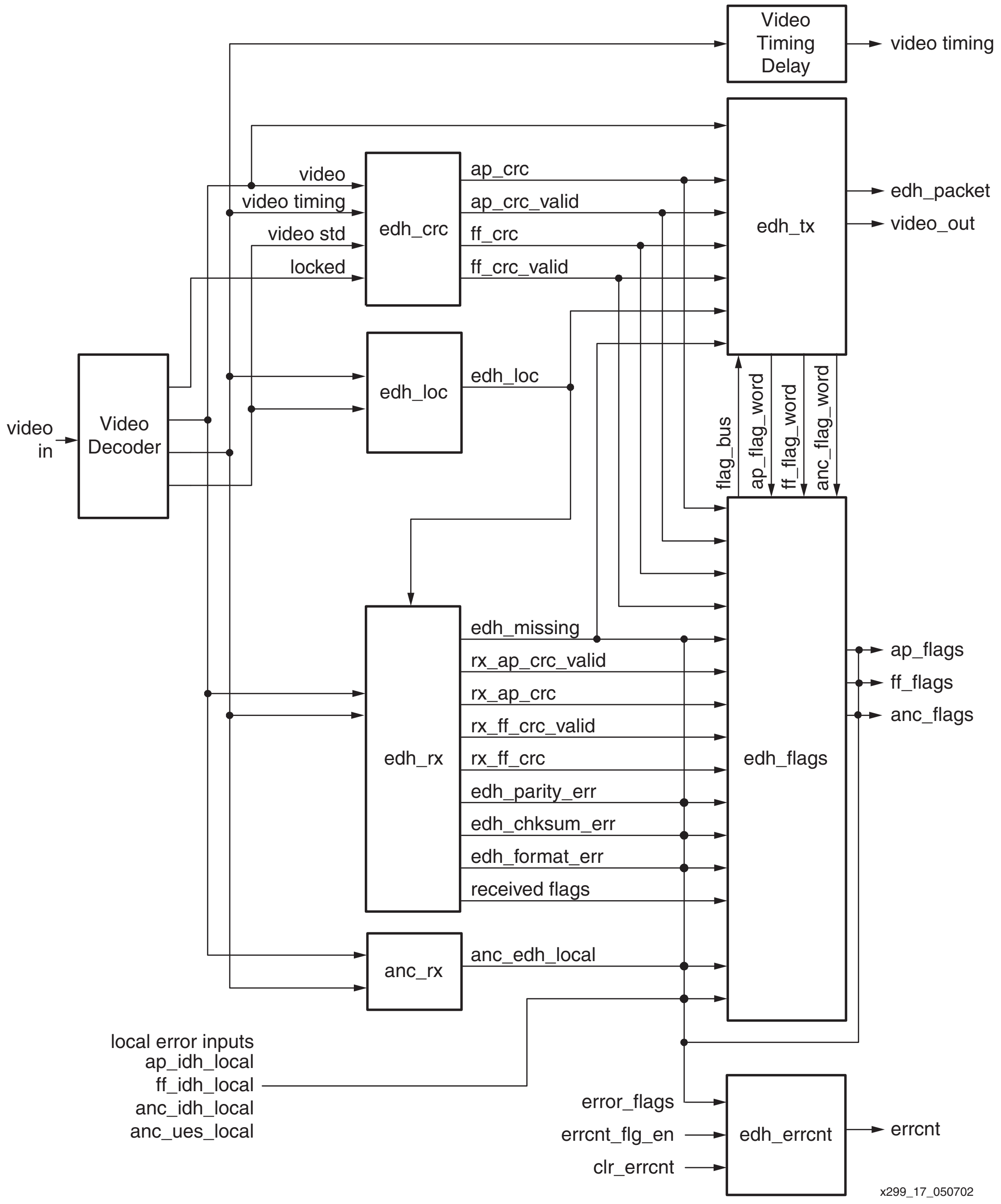


Figure 17: **EDH Processor Block Diagram**

edh_rx Module

The *edh_rx* module (Figure 18) is included in both the *edh_processor* and *anc_edh_processor* modules. It monitors the input video stream until an EDH packet is found, then it captures the various CRC checkwords and flags from the EDH packet. It also performs various checks on the received EDH packet. It asserts the *edh_missing* signal if an EDH packet is not found where one is expected. It asserts the *edh_parity_err* signal if a parity error is detected in any parity protected word of the EDH packet. It asserts the *edh_chksum_err* signal if the checksum in the received EDH packet does not match the checksum calculated by the *edh_rx* module. It asserts the *edh_format_err* signal if the DBN or DC words do not match the proper values for an EDH packet.

The *edh_rx* module has an input signal called *reg_flags*. This signal affects the timing of the received flag output ports. When the module is used with the *edh_processor*, *reg_flags* is strapped Low. When the module is used with the *anc_edh_processor*, *reg_flags* is strapped High. Figure 19 is the state diagram for the finite state machine in the *edh_rx* module.

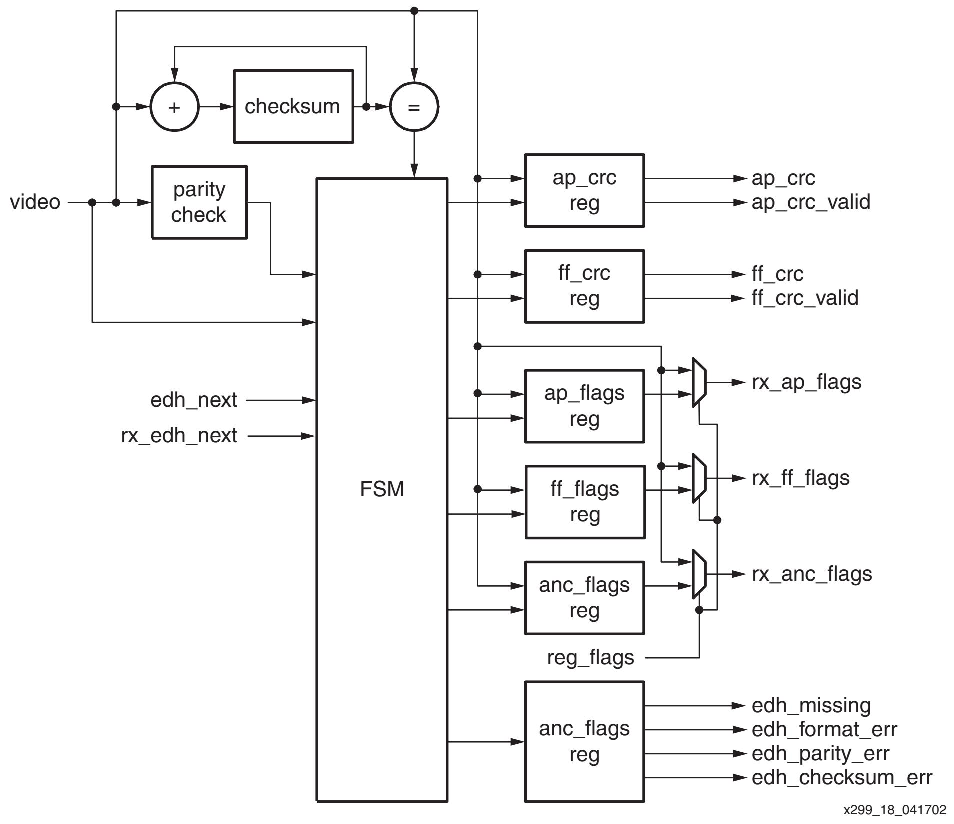


Figure 18: ***edh_rx*** **Block Diagram**

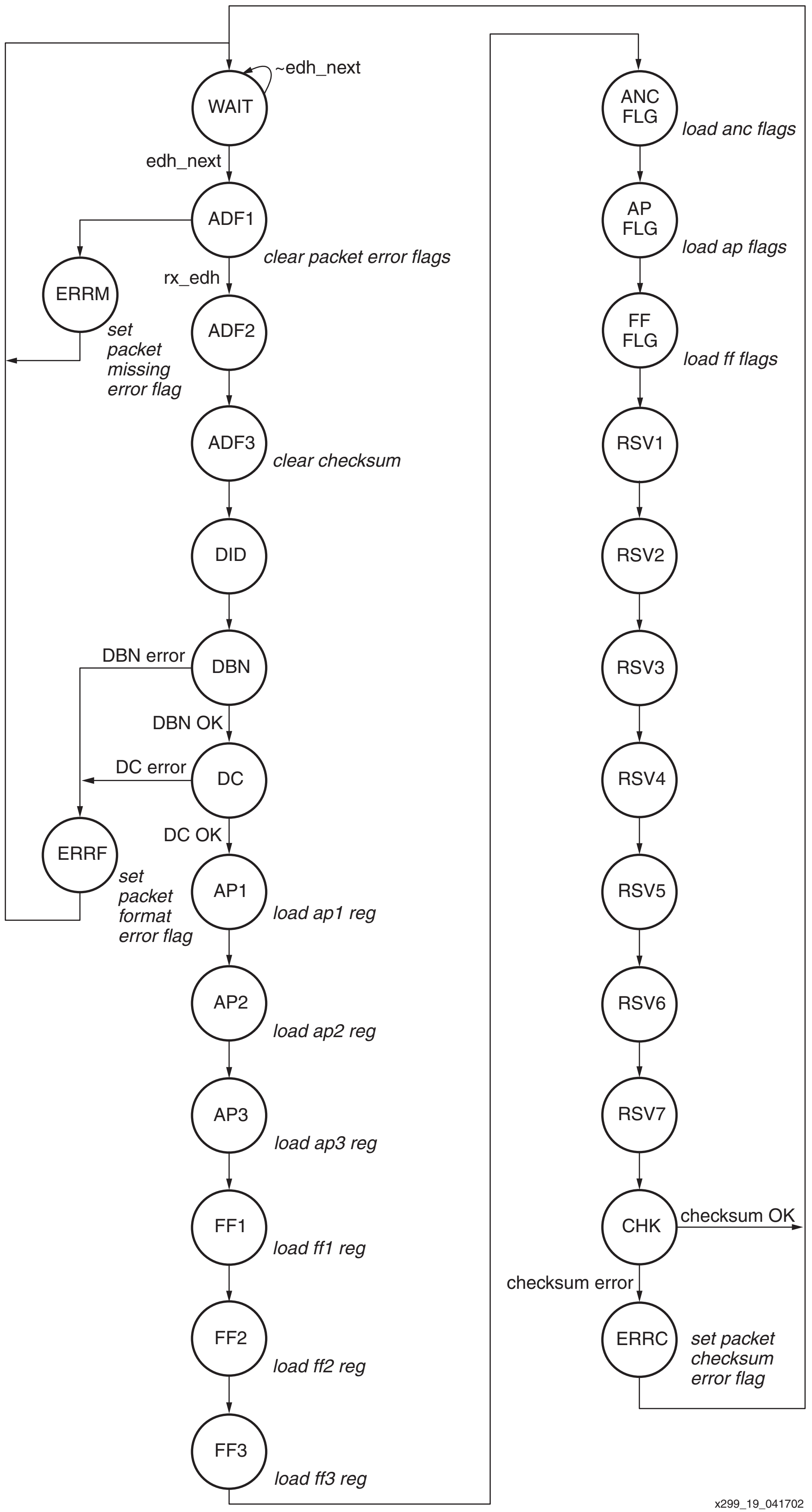


Figure 19: ***edh_rx*** **State Diagram**

anc_rx Module

The *anc_rx* module (Figure 20) is included in both the *edh_processor* and *anc_edh_processor* modules. It calculates the checksum for every received ANC packet and compares this calculated checksum with the CS word of the ANC packet. If they do not match, an error signal is sent to the *edh_gen* module allowing the error to be reported in the next outgoing EDH packet.

The finite state machine (shown in the state diagram Figure 21) in the *anc_rx* module waits until an ANC packet starts. It checks the parity on the parity-protected words. It calculates the checksum and compares it to the CS word. If either a parity error or a checksum error is detected the *anc_edh_local* output is asserted. This signal remains asserted until the next EDH packet has been sent — as signaled by the *edh_packet* signal from the *edh_gen* module going High then Low.

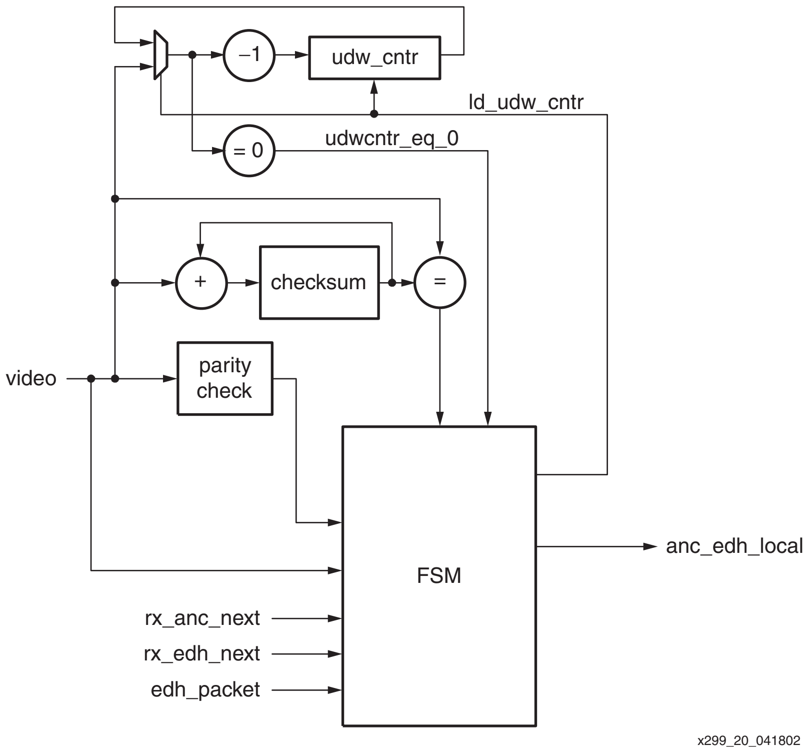


Figure 20: ***anc_rx*** **Block Diagram**

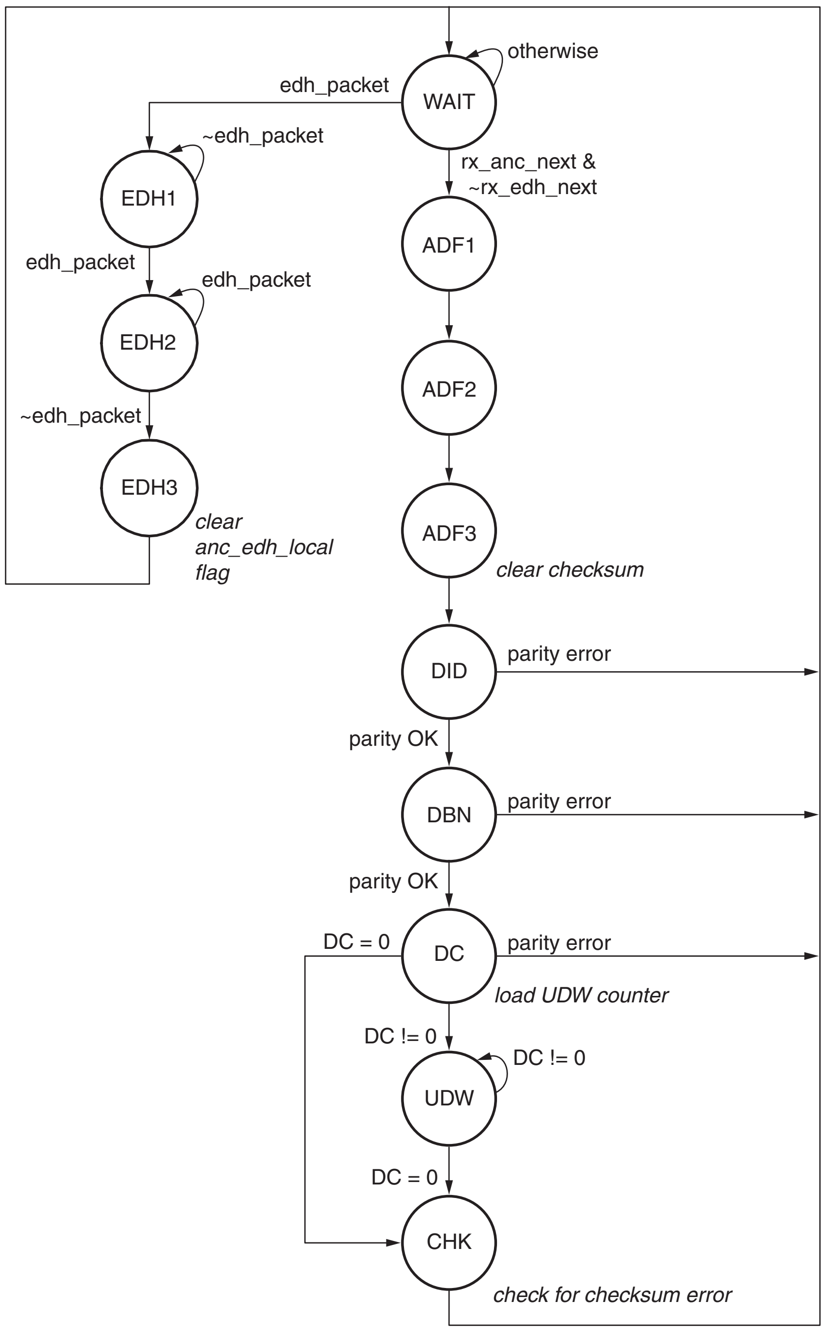


Figure 21: ***anc_rx* State Diagram**

edh_loc Module

The *edh_loc* module (Figure 22) locates the position in each field where the EDH packet should occur. The *edh_rx* module uses this signal to determine if the EDH packet is present or missing in the input video stream. The *edh_gen* module uses this signal to determine when it is time to send an EDH packet.

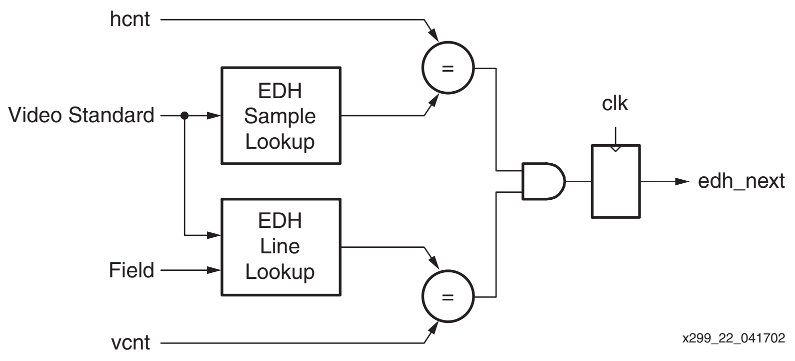


Figure 22: ***edh_loc*** **Block Diagram**

edh_crc Module

The *edh_crc* module (Figure 23) calculates the active picture and full-field CRC checkwords for each field of the video stream. In the *anc_edh_processor* design, this module is instanced twice. One instance calculates the CRC checkwords for the input video stream for comparison against the checkwords in the EDH packet. The second instance calculates the CRC checkwords for the output video stream for the *edh_gen* module to insert into the EDH packet. In the *edh_processor* design, only one instance of *edh_crc* is required because there is no ANC processing to modify the video stream. So, the CRC values calculated on the input video stream are valid for the output video stream as well.

The ITU and SMPTE standards require that any video word with 1s in all eight MSBs must also have 1s in the two LSBs for the purposes of CRC calculation. This makes the CRC calculation generate the same checkword regardless of whether the video stream was generated by eight-bit or ten-bit equipment. This requirement only applies to the input of the CRC generator and does not affect the actual words in the video stream.

The Valid Flag Logic section generates signals indicating whether the CRC checkwords are valid. The checkwords are considered valid as long as the video decoder's *locked* signal does not rise during the time when checkword is being calculated. A rising edge of the *locked* signal indicates a change in synchronization between the video decoder and the input video stream. In this case, any CRC checkword being calculated at the time of the rising edge of the *locked* signal was probably not calculated over the correct number of samples and should be considered invalid.

The actual CRC calculations are done in the *edh_crc16* modules instanced in the *edh_crc* module. Each *edh_crc16* module computes a 16-bit CRC value by combining the 10-bit video input word with the current 16-bit CRC value stored in the associated CRC register. At the beginning of a CRC region, the CRC register is cleared to zero to start a fresh CRC calculation. Load enable signals from the CRC Region Logic block control each CRC register to include into the CRC calculation only the appropriate video words.

The equations in the *edh_crc16* module have been optimized for the four-input LUT structure of Xilinx FPGAs.

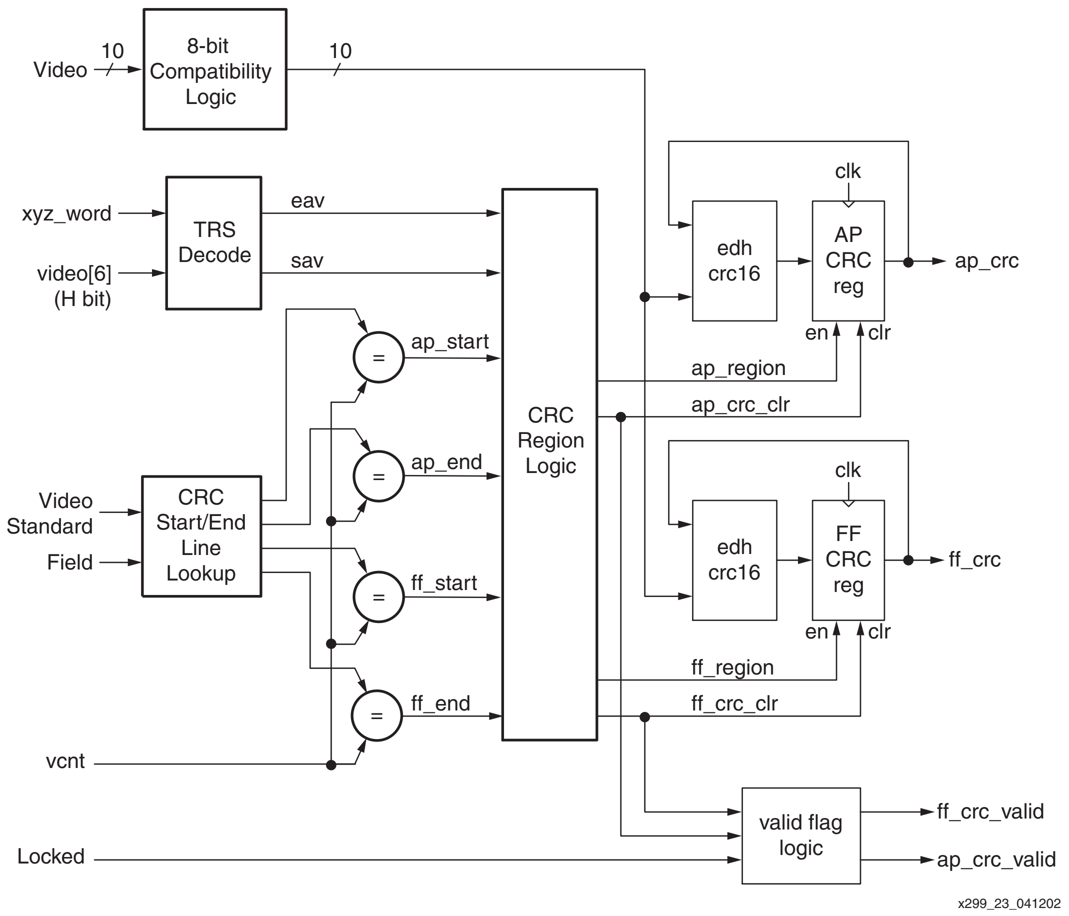


Figure 23: ***edh_crc*** **Block Diagram**

anc_mux Module

The *anc_mux* module (Figure 24) multiplexes new ANC packets into the video stream. The module contains two submodules, *anc_pkt_gen* and *anc_insert*. The *anc_pkt_gen* module accepts externally supplied raw ANC data, generates a properly formatted ANC packet from the raw data, and provides the formatted ANC packet to the *anc_insert* module. The *anc_insert* module searches the video stream for an appropriate ANC space large enough to hold the packet generated by *anc_pkt_gen*. When an appropriate space is found, the packet is transferred from *anc_pkt_gen* and inserted into the video stream.

The *anc_insert* module overwrites an end-marker ANC packet if one exists in the ANC space. However, it is not designed to overwrite an ANC packet marked for deletion.

Because the *anc_insert* module overwrites end-marker packets, it must tell the *anc_pkt_gen* module to begin sending the packet (by asserting *send_pkt*) before it can determine whether the new packet can overwrite the current packet. This determination is not made until *anc_insert* examines the DID word of the packet being overwritten to determine if it is an end-marker packet. If the packet cannot be overwritten, the *anc_insert* module asserts the *abort_pkt* signal, causing *anc_pkt_gen* to abort the packet and resend the same packet the next time *send_pkt* is asserted.

All the video timing signals from the video decoder pass through the *anc_mux* module, but are not registered. The current implementation of the *anc_mux* module does not add any cycles of latency to the video signal so there is no need to delay the video timing signals. However, future versions of the *anc_mux* module can add cycles of latency to the video signal. Passing the video timing signals through the *anc_mux* module allows them to be delayed appropriately in the future to match the video without having to change any upper-level signal connections.

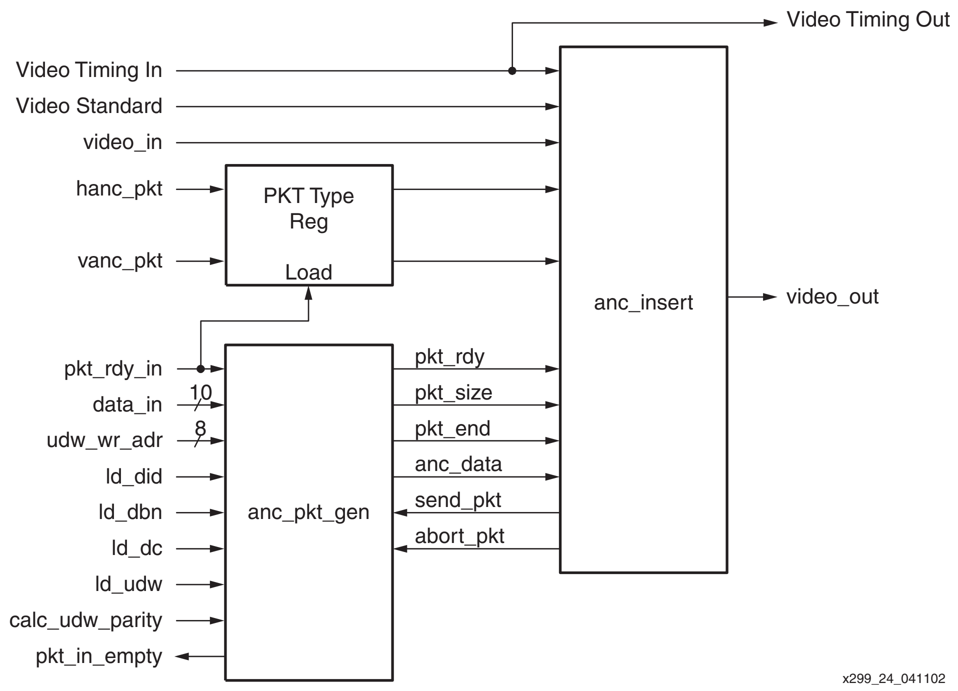


Figure 24: ***anc_mux*** **Block Diagram**

anc_insert Module

The *anc_insert* module (Figure 25) multiplexes ANC packets generated by the *anc_pkt_gen* module into the video stream.

A state machine (shown in the state diagram Figure 26) searches for EAV and SAV symbols in the video stream. An EAV symbol marks the beginning of HANC space and an SAV symbol marks the beginning of VANC space if the line is in the vertical blanking interval. If the *anc_pkt_gen* module asserts the *pkt_rdy_in* signal, the state machine determines if the packet can be inserted immediately after the EAV or SAV symbol. The packet can be inserted if there is no ANC packet already in the video stream immediately after the EAV or SAV. If the *pkt_rdy_in* signal becomes asserted after the state machine finds free ANC space, but before the end of the space, the ANC packet cannot be inserted. Doing so would violate the requirement for contiguity of the ANC packets.

If an ANC packet is found in the video stream, it will be overwritten if it is an end-marker packet. Otherwise, the state machine examines the DC word of the packet to determine the length of the packet and waits until the end of the packet. If another ANC packet immediately follows, this procedure is repeated. If not, the state machine inserts the new packet if there is enough space remaining in the ANC space.

The state machine tells the *anc_pkt_gen* module to begin sending the new packet if there is chance that it can be inserted. If the state machine determines that the packet cannot be inserted, then the module asserts the *abort_pkt* signal to cancel the packet. The packet is aborted if the state machine finds an existing ANC packet in the video stream that is not an end-marker packet. The state machine also cancels the packet if the ANC space is part of the synchronous switching interval. The switching signal cannot be generated soon enough to determine start of the synchronous switching interval until the clock cycle after *send_pkt* signal must be asserted to the *anc_pkt_gen* module. The abort mechanism is used to cancel the packet if the switching signal is asserted.

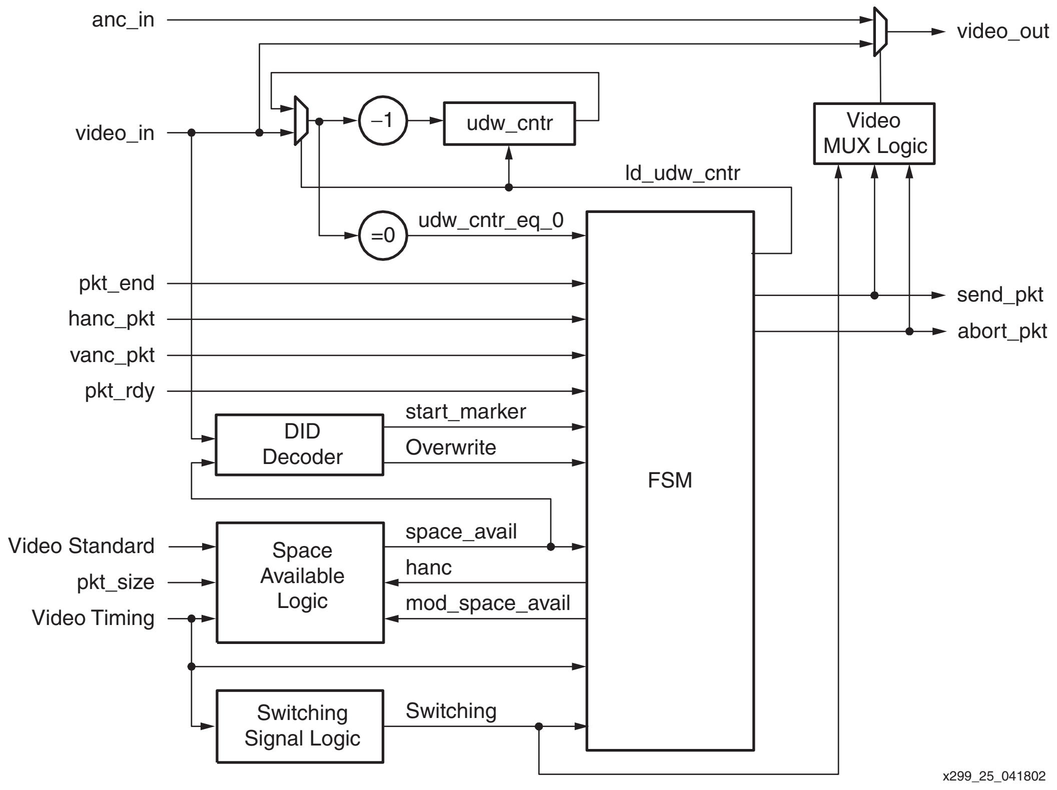


*Figure 25: **anc_insert*** **Block Diagram**

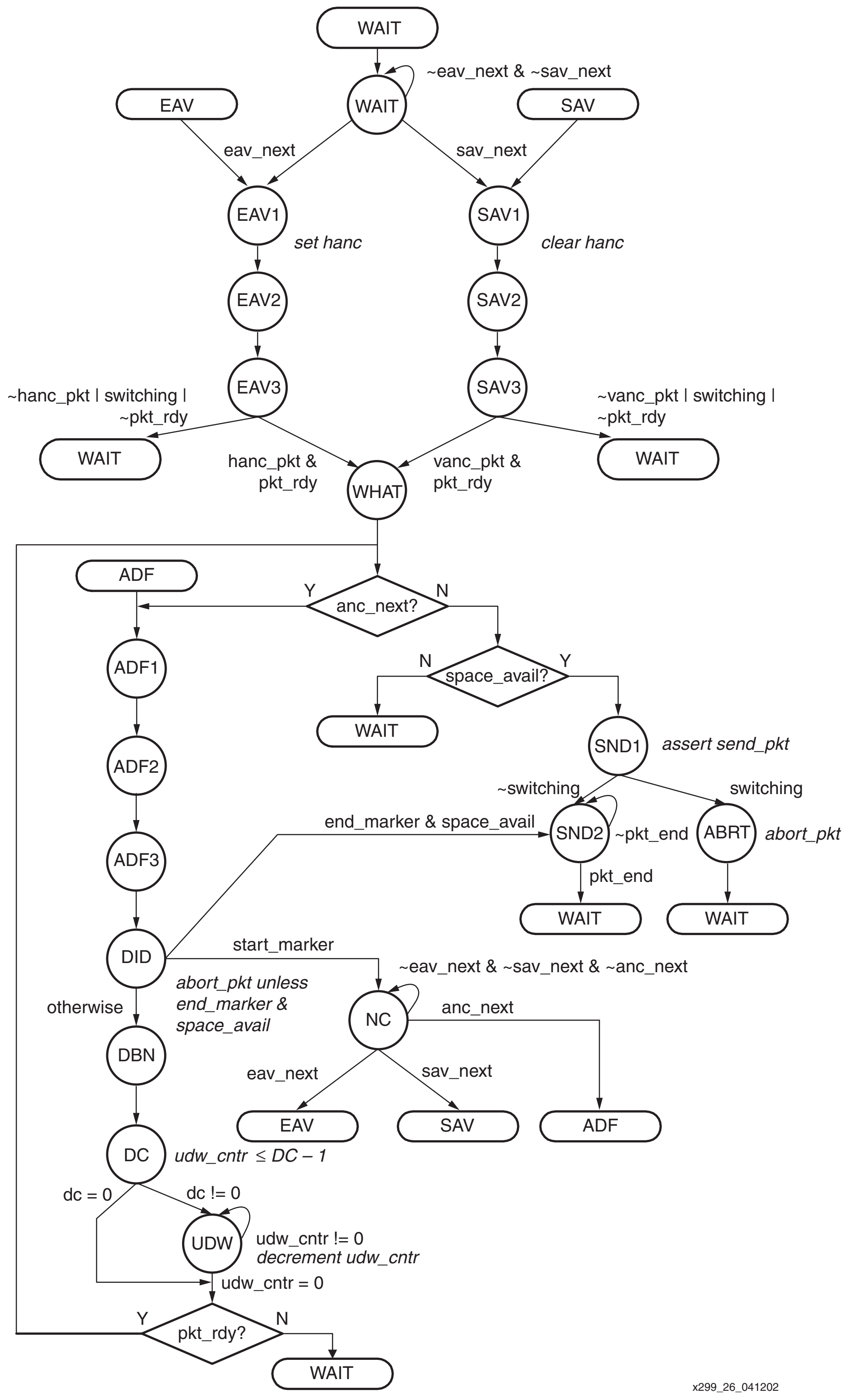


*Figure 26: **anc_insert*** **State Diagram**

anc_pkt_gen Module

The *anc_pkt_gen* module (Figure 27) generates an ANC packet from raw ANC data.

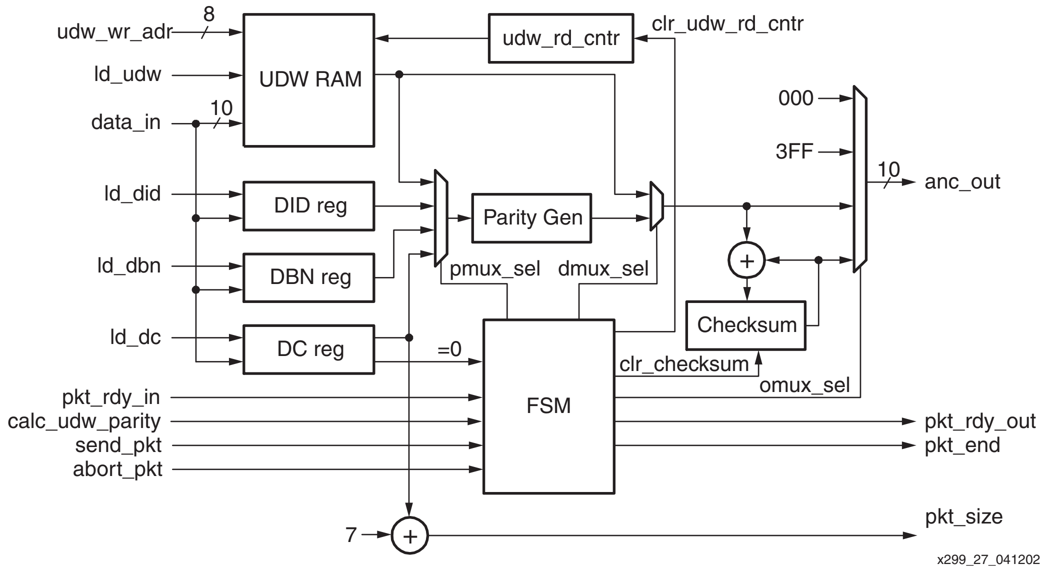


Figure 27: ***anc_pkt_gen*** **Block Diagram**

An external processor or another module writes the raw ANC data into the *anc_pkt_gen* module. The external processor can begin loading the ANC data as soon as the *anc_pkt_gen* module asserts the *pkt_in_empty* signal. The external processor must provide 8-bit DID, DBN/SDID, and DC values. These values must be placed onto the LS eight bits of the *data_in* port and the appropriate load signal (*ld_did*, *ld_dbn*, or *ld_dc*) must be asserted at the same time until the rising edge of the clock. The *ld_dbn* signal is used to load either the DBN or the SDID value. The user data words, if any, are written into the module one at a time. To write the UDWs, each 10-bit UDW is placed on the *data_in* port, the word number of the UDW is placed on the *udw_wr_adr* port (0 for the first word, 1 for the second word, etc.), and the *ld_udw* signal is asserted until the rising edge of the clock. The DID, DBN/SDID, DC, and UDW words can be written to the module in any order.

If the ANC packet format requires bit 8 of every UDW to be an even parity bit and bit 9 to be the complement of bit 8, the *anc_pkt_gen* module can calculate bits 8 and 9. The eight LSBs of each UDW are placed on the eight LSBs of the *data_in* port, and the *calc_udw_parity* signal is asserted by the external processor.

When all the data for the packet has been written to the module, the external processor must assert the *pkt_rdy_in* signal for one clock cycle. During the same cycle, the processor must also indicate whether the packet is to be inserted into HANC space by asserting the *hanc_pkt* input or VANC space by asserting the *vanc_pkt* input. If both *hanc_pkt* and *vanc_pkt* are asserted, the packet is inserted into the first ANC space that has sufficient room for the packet. The *hanc_pkt* and *vanc_pkt* signals are captured in a register in the *anc_mux* module when *pkt_rdy_in* is asserted and sent to the *anc_insert* module. The *anc_pkt_gen* module does not use these signals.

The *anc_pkt_gen* module stores the UDW values in a RAM. This RAM can be implemented in either distributed RAM or block RAM. The source files for this module contain code to allow either distributed RAM or block RAM to be inferred by the synthesizer. In the Verilog code, the following statement:

```
    `define UDW_BLOCK_RAM
```

causes block RAM to be inferred if present. If this statement is commented out or deleted, distributed RAM will be inferred. In the VHDL file, the two sections of code exist in the file with one of them commented out.

Different code is used to infer the two types of RAM rather then using synthesis options, because a common code base would infer a dual-port distributed RAM. Only a single-port distributed RAM is required and this is half the size of a dual-port distributed RAM.

The UDW RAM uses 2560 bits, 256 words times 10-bits each, to support the maximum number of UDW words allowed in an ANC packet. If an application always creates ANC packets with less than the maximum number of UDW words, the size of the UDW RAM could be made smaller, saving space if distributed RAM is used. Parameters or generics at the beginning of the module control the size of the UDW RAM and the width of the address bus or busses supplied to the RAM and to the module.

Only one 2560-bit UDW RAM will fit in the 4096-bit block RAMs of the Virtex and Spartan-II families. However, the larger block RAM in Virtex-II family devices could hold multiple UDW RAMs. The current design does not allow multiple ANC packets to be written to the *anc_pkt_gen* module. Consequently, only one ANC packet can be inserted into any ANC space. However, the module could be modified to allow multiple ANC packets to be stored using a FIFO technique. Some modifications to the state machine would be required to allow the module to insert multiple consecutive ANC packets into the same ANC space.

There is another way to insert multiple ANC packets in an ANC space that requires no modification to the existing *anc_mux* design. Two or more *anc_mux* modules can be cascaded. The first *anc_mux* module will insert its ANC packet into the first available ANC space. The second *anc_mux* module will insert its ANC packet immediately after the ANC packet inserted by the first module, and so on.

Cascaded *anc_mux* modules inherently provide priority to the first *anc_mux* module. Consider what happens if the first *anc_mux* inserts its ANC packet, but there is no room in the same ANC space for the second *anc_mux* to insert its ANC packet. If a new ANC packet is written into the first *anc_mux* before the next ANC space occurs, then the first *anc_mux* inserts its new ANC packet into the video stream before the second *anc_mux* has a chance to insert its ANC packet. If such behavior is not desired, the *pkt_rdy_in* signals of the various *anc_mux* modules need to be carefully controlled to prevent the first *anc_mux* from always taking priority.

The current *anc_mux* module design provides a purely combinatorial path for the video signal. There is no input or output register on the video path. Cascading *anc_mux* modules will increase the number of logic levels on the video path, making it more difficult to meet timing. A pipeline register can be required between cascaded *anc_mux* modules in order to meet timing. Be sure to delay all video timing signals as well, if a pipeline register is inserted into the video path to keep them synchronized.

Figure 28 shows the state diagram for the finite state machine in the *anc_pkt_gen* module.

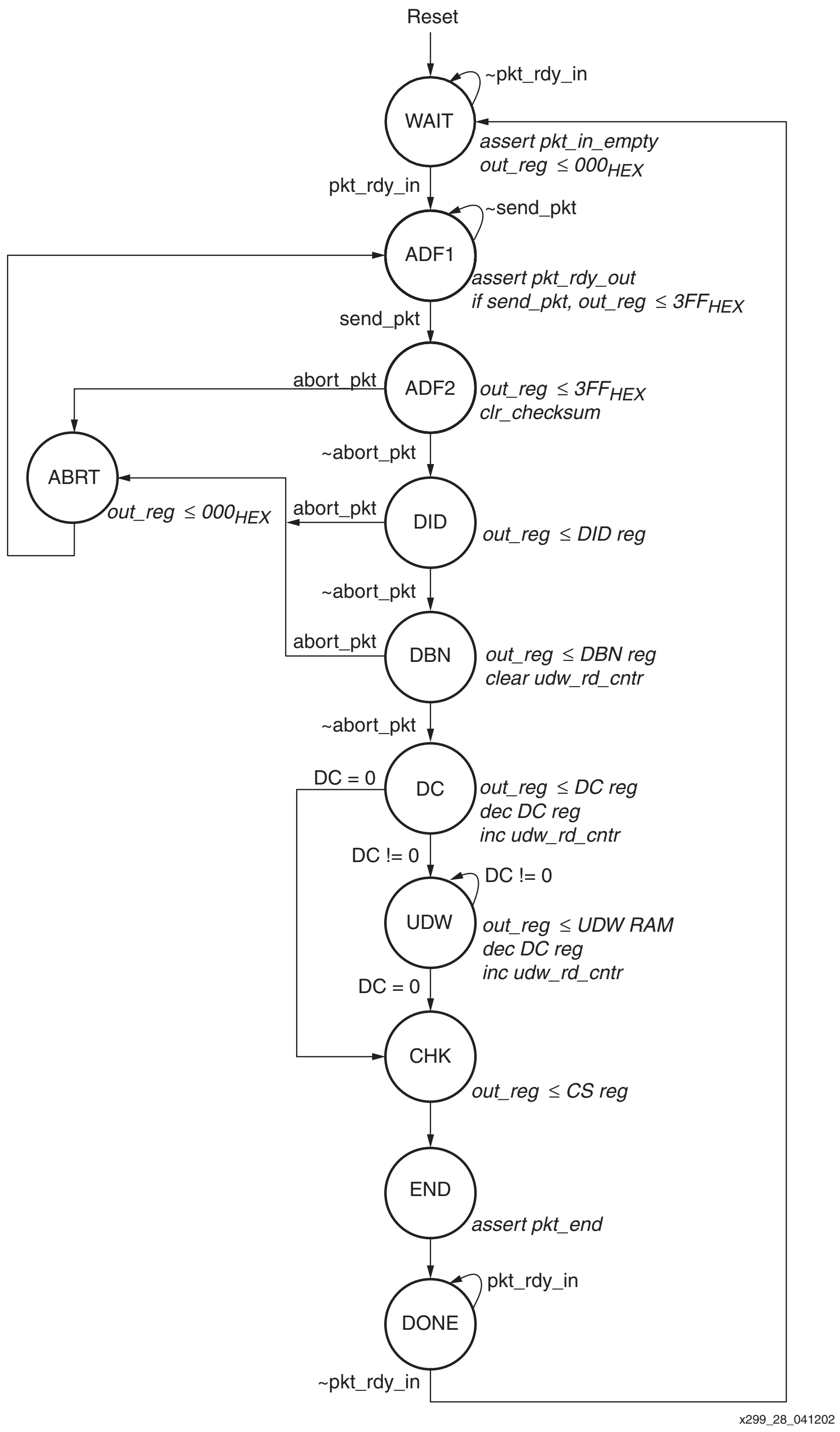


Figure 28: ***anc_pkt_gen*** **State Diagram**

anc_demux and *anc_extract* Modules

The *anc_demux* module searches for certain types of ANC packets and demultiplexes them from the video stream. When a matching ANC packet is found, the module provides the ANC packet data to a separate output port, *data_out*. The module also provides a number of signals indicating when the ANC packet information is available on the *data_out* port and which word of the ANC packet is currently available. These signals can be used by another module or external processor to store or process the demultiplexed ANC packet. The demultiplexed ANC packet can either be left intact in the video stream or it can be marked for deletion.

The *anc_demux* module is actually a wrapper around the *anc_extract* module. The *anc_extract* module (Figure 29) does all the work of searching for and demultiplexing ANC packets. The *anc_extract* module introduces three clock cycles of latency to the video stream. The *anc_demux* module delays all the video timing signals by three clock cycles to match the delay added to the video stream by *anc_extract*.

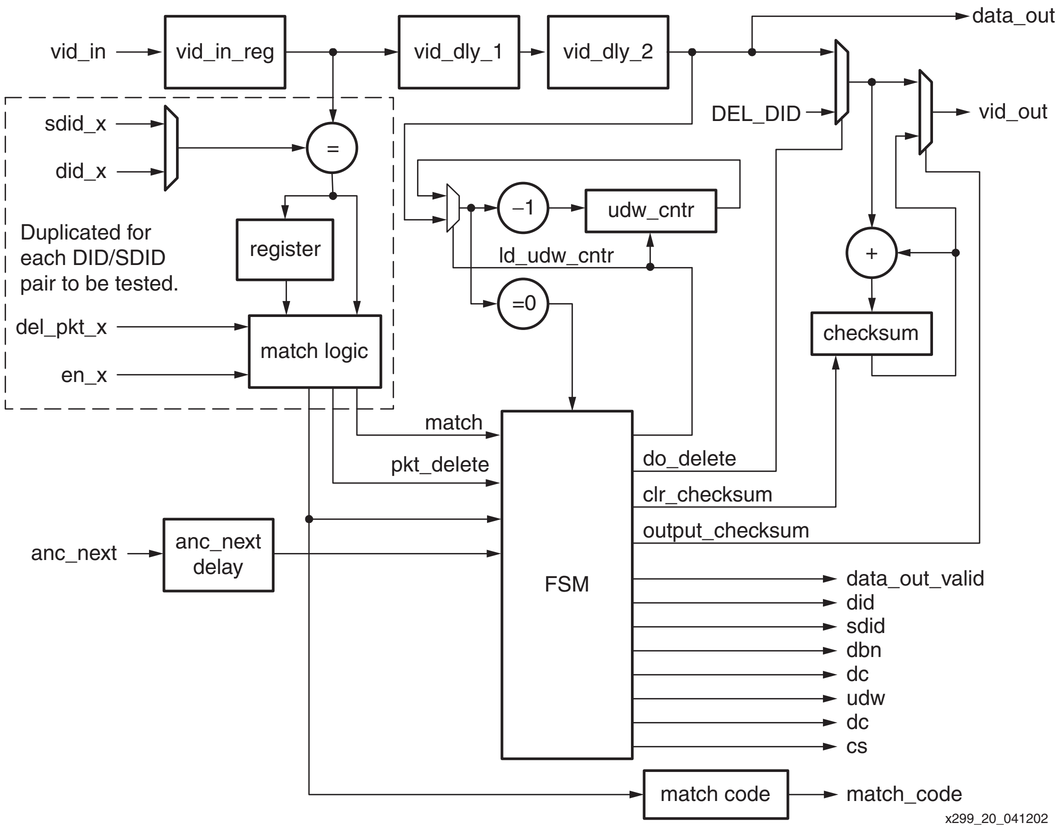


Figure 29: ***anc_extract*** **Block Diagram**

The *anc_demux* module can search for and demultiplex up to four different ANC packet types. There are four sets of inputs allowing the ANC packet types to be specified. Each set contains a DID value, a SDID value, an enable signal, and a *del_pkt* signal. If the DID value indicates a Type 1 ANC packet, the SDID value is ignored. If the DID value indicates a Type 2 ANC packet, the SDID value is also used to find matching packets. If the enable signal for the set is Low, the DID and SDID input set are not used by the module when searching for matching ANC packets. If the *del_pkt* input is asserted High, any matching packets are marked for deletion as they are demultiplexed.

The *anc_demux* module provides the demultiplexed data on the *data_out* port. The *data_out_valid* signal is asserted when the *data_out* port contains valid ANC data. This signal is asserted starting with the DID word and stays asserted through the CS word of the packet. It is not asserted for the three words of the ADF that marks the beginning of the packet. The *did*, *sdid*, *dbn*, *dc*, *udw*, and *cs* outputs of the module are asserted when the corresponding parts of the demultiplexed ANC packet are present on the *data_out* port. The module also places a value on the *match_code* output port to indicate which of the four input DID/SDID sets matched the packet: "00" for the A set, "01" for the B set, "10" for the C set, and "11" for the D set.

The *anc_extract* module calculates a new checksum for the ANC packet and inserts it into the ANC packet in the video stream. This is required because the module might modify the ANC packet if the packet is marked for deletion. The newly calculated checksum always replaces the CS word in every ANC packet, regardless of whether the packet is marked for deletion or not. If this behavior is not desired, the module must be modified to only replace the CS word in packets it marks for deletion.

If an application requires the *anc_demux* module to search for and demultiplex less than four different ANC packet types, the unused DID/SDID input sets must be disabled using the enable signal associated with each pair. However, the decoders and logic associated with the unused sets will still be synthesized. Unused input sets can be removed from the *anc_demux* module code to save space.

If an application requires demultiplexing of more than four different ANC packet types, the *anc_demux* module can be modified to provide more DID/SDID input sets. Or, multiple *anc_demux* modules can be cascaded. Unlike the *anc_mux* module, the *anc_demux* has pipeline registers, so cascading *anc_demux* modules should not present any timing problems.

Figure 30 shows the state diagram for the finite state machine for the *anc_extract* module.

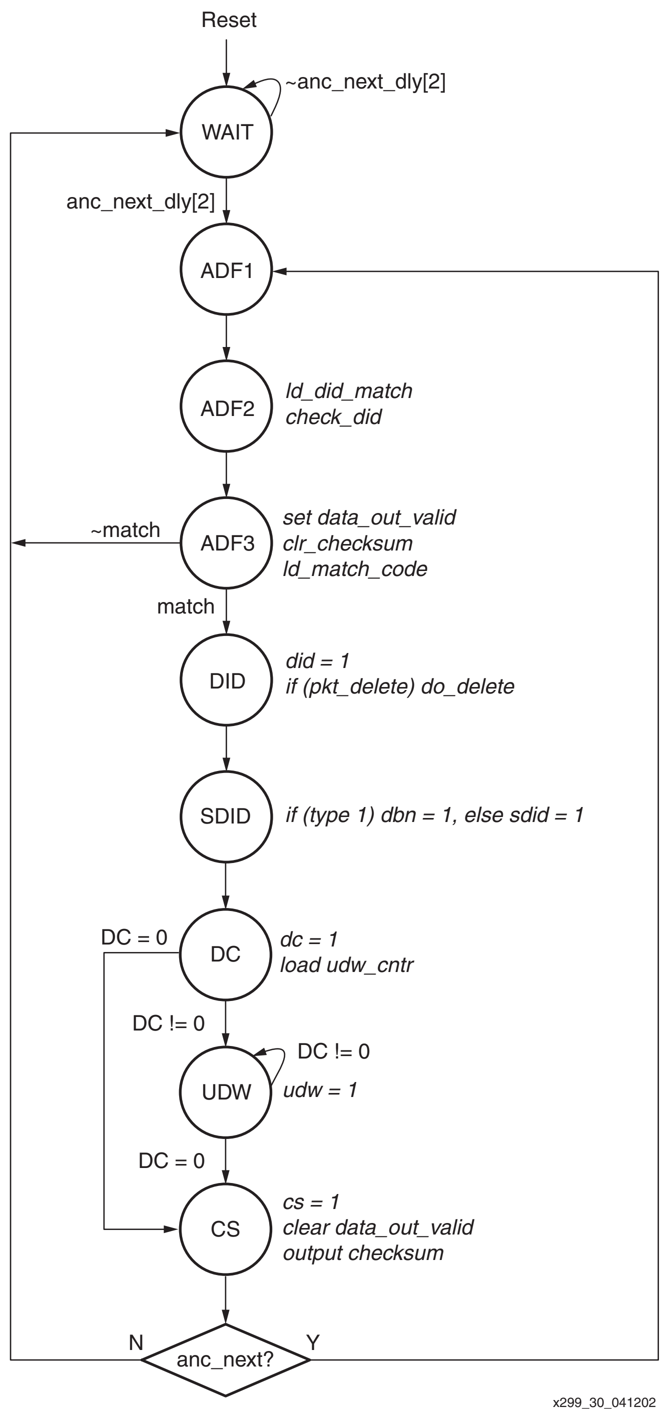


Figure 30: ***anc_extract*** **State Diagram**

anc_edh_processor **Module**

The *anc_edh_processor* implements a complete ANC and EDH processor design. It implements **XAPP625** video decoding, input EDH packet processing, ANC multiplexing, ANC demultiplexing, and output EDH packet generation. The design can be easily modified to remove either the *anc_mux* or *anc_demux* modules or to cascade multiple *anc_mux* or *anc_demux* modules.

edh_gen Module

The *edh_gen* module (Figure 31) is used by the *anc_edh_processor* module. It calculates the CRC checkwords and generates a new EDH packet that is inserted into the outgoing video stream.

The *edh_gen* module instances the *edh_crc* module to calculate the CRC checkwords and the *edh_tx* module to generate the EDH packets. It also provides an output register for the video path and the various video timing signals.

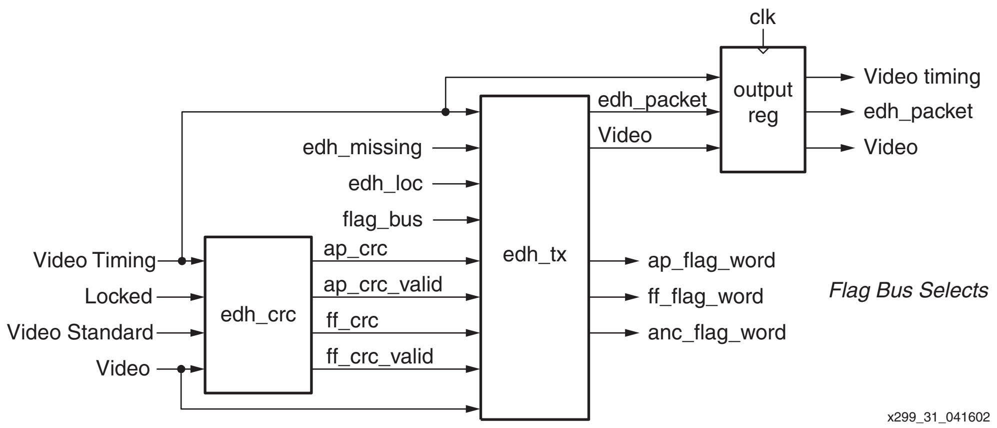


Figure 31: ***edh_gen*** **Block Diagram**

edh_tx Module

The *edh_tx* module (Figure 32) generates new EDH packets and inserts them into the outgoing video stream. This module is used directly by the *edh_processor* design. The *anc_edh_processor* instances an *edh_gen* module. The *edh_gen* module instances the *edh_tx* module.

The *edh_tx* module's finite state machine (state diagram shown in Figure 33) waits for the *edh_next* signal to be asserted. This signal is usually generated by an *edh_loc* module and signals the *edh_tx* module to output the first word of the EDH packet during the next clock cycle. The FSM contains a state for each word of the EDH packet and controls a big MUX to output the words of the EDH packet in the correct sequence.

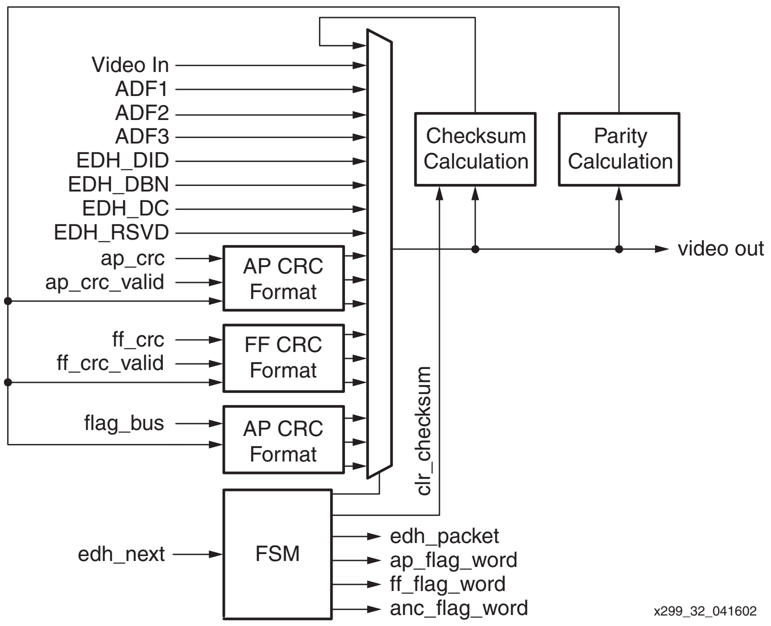


Figure 32: ***edh_tx*** **Block Diagram**

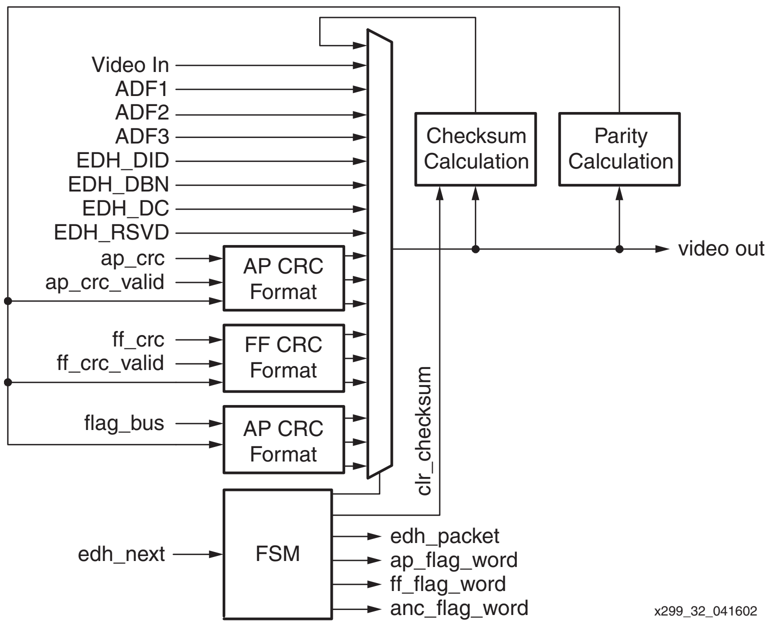


Figure 33: ***edh_tx*** **State Diagram**

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
05/16/02	1.0	Initial Xilinx release.