Scientific Atlanta
A CISCO COMPANY

FEC on IP-output for video encoder

Maria Baltzer Pedersen

Technical University of Denmark
Kongens Lyngby 2006

IMM-B.ENG-2006-60

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
http://www.imm.dtu.dk

Summary

Broadcast systems based on IP technology is becoming a reality. At this very moment, there are already specific projects on the market, where the local infrastructure for a service provider is build around an IP network. Furthermore many projects are in the making, based on the use of IP networks for end delivery, focusing on the consumer.

It is expected that the market will demand Forward Error Correction on the output of a video encoder. Today error correction is not supported on Scientific Atlanta encoders, and while the current hardware in the SD encoders manufactured by Scientific Atlanta can not be extended with a Forward Error Correction solution, the HD encoders are presumed to be sufficient for experimental implementation.

The project was initiated by an analysis in which different types of applications and technical solutions including Forward Error Correction (FEC) on the IP output of an encoder. The result of the analysis is a suggestion to a FEC to be implemented in future encoders.

The questions answered in the analysis include the customers' needs and briefly touches the subject of other manufactures of encoders, and which FEC solutions they have implemented. Because of the many different types of FEC solutions, the decision of which algorithm to implement is taken after the analysis.

In the analysis, the transmission layers are analyzed with focus on the encoder side. This identifies the layer in which the FEC scheme is to be implemented or when in the encoding process the FEC is appended to the program stream that is. A program stream is a series of transport stream packets (TS packets) to which the appropriate headers are appended to form an IP packet. More specifically, the analysis shows if the FEC should be appended to the transport stream packets or to the IP packets.

The chosen FEC algorithm is implemented in hardware, and some simulations have been performed, but these are not included in the report.

Resumé

Broadcastsystemer, der baserer sig på IP-teknologi, er ved at blive en realitet. Allerede nu er der konkrete projekter fremme, hvor infrastrukturen for en ”service provider” lokalt er bygget op omkring IP. Tilsvarende er flere projekter i støbeskeen hvor IP tages i anvendelse til slutleverance rettet mod seerne.

Det forventes, at markedet vil kræve Forward Error Correction (FEC) for IP outputtet fra en videoenkoder. I dag understøttes ikke fejlkorrigering for IP i enkodere fra Scientific-Atlanta. Den nuværende hardware for firmaets SD enkodere kan ikke udbygges med FEC, mens HD enkoderne antages at kunne tages i brug til en eksperimentel implementering.

Projektet blev indledt med en afdækning af applikationer og foreslåede tekniske løsninger for FEC til brug for et IP output på en encoder. Undersøgelsen mundede ud i et forslag til FEC i nye enkodere.

Spørgsmålene der besvares i undersøgelsen omfatter hvad kunderne vil have, hvad behovene er og hvilke løsninger andre producenter har implementeret i deres enkodere. Der er flere forskellige muligheder for FEC-algoritmer der kan vælges, baseret både på behov, efterspørgsel og anbefalinger, hvorfor det først er besluttet efter de indledende undersøgelser hvilken type FEC der skulle implementeres.

I projektet analyseres transmissionslagene, med fokus på enkodersiden.Det undersøges hvilket lag FEC’en skal implementeres i, det vil sige hvor i enkodningsprocessen FEC’en skal tilføjes til programstrømmen. En programstrøm til IP-output er en række transportstrømspakker der bliver samlet og tilføjet nogle headers, så det resulterer i en IP-pakke. Konkret viser undersøgelsen derfor, om FEC skal tilføjes på transportstrømspakke-niveau eller på IP-pakke niveau.

Den valgte metode og den fejlkorrigerende hardware er implementeret ved hjælp af VHDL, og der er foretaget nogen simulering, men resultaterne af denne simulering er ikke medtaget i rapporten.

Preface

This report is the result of my exam project, with which I complete the Bachelor of Engineering from the Institute for Informatics and Mathematical Modelling at the Technical University of Denmark.

The project was carried out in cooperation with Scientific Atlanta Denmark A/S, A Cisco Company, and the project period was spent at their premises in Søborg, Denmark.

In the process of the development I have enjoyed the help of the employees at Scientific Atlanta who have all been curious and helpful during the project period. For this I thank everyone, especially the project group concerned with the D9054 h.264 Encoder. During the entire project I functioned as a part of this group, and everyone welcomed me and helped me in any way they could.

My supervisor at Scientific Atlanta, Ole Stender Nielsen, had faith in my skills and my interest in the subject of the project from the very beginning, and for this I thank him.

At the Technical University of Denmark, I had the pleasure of having Ole Remmer as my supervisor. The meetings we had were always joyful and uplifting.

My biggest thanks go to Bo Dyssegard of Scientific Atlanta for never getting tired of my constant questions and for supplying me with tons of useful information concerning the existing hardware in the target encoder. Without his help I would have had many more frustrating moments during the project period, when nothing seemed to go my way.

Maria Baltzer Pedersen
July 4th - 2006

Contents

Table of figures

List of Abbreviations

ATM	Asynchronous Transfer Mode
BCH	Bose, Ray-Chaudhuri, Hocquenghem
CoP	Code of Practice
ES	Elementary Stream
BIFS	Binary Format for Scenes
FEC	Forward Error Correction
FIFO	First In First Out-queue
FPGA	Field Programmable Gate Array
HD	High Definition
IEC	International Electro-technical Commission
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPTV	Internet Protocol Television
ISO	International Organization for Standardization
ISOC	Internet Society
LAN	Local Area Network
MALLOC	Memory Allocation
MPEG	Moving Picture Experts Group
PCI	Peripheral Component Interconnect
PT	Payload Type
RFC	Request for Comments
RS	Reed-Solomon
RTCP	RTP Control Protocol
RTP	Real-Time Transport Protocol
S-A	Scientific Atlanta
SD	Standard Definition
SDH	Synchronous Digital Hierarchy
SDRAM	Synchronous Dynamic Random Access Memory
SN	Sequence Number
TDC	Tele-Denmark Communications
TS	Transport Stream
TV	Television
UDP	User Datagram Protocol
WAN	Wide Area Network
XOR	Exclusive Or

CHAPTER 1

Analysis

1.1 Video streaming

Video signals are based on analogue technology. They are carried via expensive transmission circuits. We now, however, live in a digital world, and through advancements in digital video compression composite audio and video signals can now be carried over typical network circuits both on the LAN and across the WAN, and even over the Internet. Video over IP or IP Streaming Video are newer technologies that allow video signals to be captured, digitized, streamed and managed over IP networks, networks that already exist and hereby enabling almost everyone to receive digital television. That means easy distribution of the video and audio since the network to distribute it on is often already present.

Video contribution and distribution over IP is a network-based one-way or two-way transmission of audio visual file content. Usually the endpoint is merely a passive viewer with no control over the session, but other applications do exist where content is returned.

1.1.1 MPEG-4 over IP

The MPEG-4 standard is very extensive and the details of the encoding and decoding are of little importance to the project, since the level of abstraction is high compared to the technical details of the standard. A brief description of the standard follows.

The Moving Picture Experts Group (MPEG) a working group of ISO/IEC is in charge of the development of standards for coded representation of digital audio and video. The group was established in 1988, and has since then produced MPEG-1, the standard on which such products as Video CD and MP3 are based, MPEG-2, the standard on which such products as Digital Television set top boxes and DVD's are based, MPEG-4, the standard for multimedia for the fixed and mobile web, MPEG-7, the standard for description and search of audio and visual content and MPEG-21, the Multimedia Framework[1].

The MPEG-4 standard which is the basis for the content used in this project is the first standard to actually provide guidelines for the transmission of encoded multimedia over some sort of network. The standard mostly follow the MPEG-2 standard but with some improvements on the image processing side.

The actual compressing of the media is not at all relevant for this project and will not be described, whereas the outline of the compression procedure is given.

The standard gives a way to represent multiple audiovisual elements, such as natural and artificial content, in one final scene, presented to the user. The different elements of the stream are encoded separately in their own elementary stream (ES) and the spatial and temporal locations of the elements are encoded in a scene description (BIFS – Binary Format for Scenes).

The result of the encoding is a transport stream (TS) consisting of TS packets with a size of either 188 or 204 bytes and it is at this point in the encoding the Forward Error Correction is interesting. This is the point where the IP packets for transmission over an IP network is constructed, and this level of the encoding is the lowest level of abstraction in the following chapters [1][2].

[1] http://www.chiariglione.org/mpeg/

1.2 Scope

The analysis and development of this application are limited to applications handling contribution, the transmission of audio visual content between for example studios and TV network stations, and primary distribution, which is the transmission of content between for example the television network stations and the actual service providers or satellite transmitters (head ends). This means that the network in which the data is transmitted is of a certain quality and can be measured as to examine the error rate and bandwidth to ensure some minimum in the quality of the transmission.

The terms are related to their surroundings in the figure below:

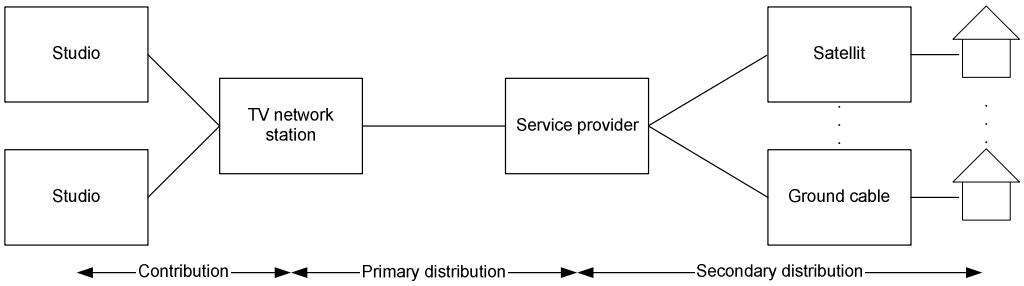


Figure 1.1 - Scope

As the figure shows, the networks on which this analysis is based, is only between the studios and the television network stations, contribution, and between the television network stations and the actual service provider, primary contribution. Some parts of the secondary distribution can be covered by this analysis, but only if the network is secure and dedicated to the purpose. This could for example be the network connection from the service provider to the actual antenna tower transmitting the signal for satellite distribution (head end).

1.3 Forward Error Correction

Whenever data is transmitted, via cable, radio, or satellite link, the transmitted signal will be subjected to distortions and noise, and so the received signal will differ from the transmitted signal, and transmission errors will have been introduced.

These errors could come from:

- signal degradation over the network medium,
- noise on the transmission channel,
- oversaturated network links,
- corrupted packets rejected in-transit by intermediary nodes or the destination node, or
- Faulty networking hardware.

Packet loss or bit errors are usually in the form of burst loss where a number of consecutive packets or bits are lost or random loss where as the name indicates only single random packets or bit in the stream are lost.

With uncompressed video, a single bit error would result in the loss of a single pixel - a small element which is barely noticeable. Compressed pictures are described using fewer bits of information, and consequently, each bit takes on a much greater meaning. Thus a bit error in a compressed picture can result in large areas of the picture being corrupted[2]. The effect is also very noticeable when dealing with audio, where a small error can have a great effect on the output. This is not acceptable by the end users who will experience pronounced irregularities in the signal received [3].

Forward Error Correction (FEC) is a method for minimizing these irregularities by allowing the receiver to reconstruct a number of lost packets in the stream, without a retransmission of data. The reconstruction is done by applying some FEC algorithm to a set of packets, at the sender, which results in additional IP packets. These packets are transmitted over the IP network as well as the stream but for another destination port so that it does not interfere with the original stream. These FEC packets can now be used by the receiver to reconstruct lost packets or can be discarded if there has been no packet loss in the transmission.

[2] Block errors, striping errors etc.

The general idea is to enforce the use of averaging noise which works by the principle that every bit or packet in the original stream affects one or more other bits or packets. This makes it possible to extract the original data from the other data which also depends on the lost data. A simple example of this is the XOR-code which is described in section 1.4.3 where one lost packet can be reconstructed by using the other data packets and the added FEC packet.

1.4 FEC algorithms

It is difficult to choose one FEC algorithm rather than another because there are so many to choose from, and they all perform differently and have different characteristics. One algorithm can be very efficient in reconstructing bit errors in a transmitted stream, another algorithm may have the ability to correct packet loss in the stream and finally also the computing time for the algorithms is of great importance. Therefore, the choice of algorithm has to take the type of errors, the number of errors and the resulting delay caused by the encoding and decoding of the content into consideration.

1.4.1 Types of algorithms

There are several types of FEC algorithms, each with different advantages and disadvantages. Algorithms that include the original information in the encoded output are classified as systematic whereas an algorithm that does not include the original information is referred to as non-systematic. Each of the two types of codes has different properties. The advantage of a systematic code is that the original data can be used directly without any alterations or any computing if no errors are present, whereas with the non-systematic codes, the receiver has to decode the received packets before using them.

There are two major types of error correction algorithms; block codes and convolutional codes. The difference between these two types of codes is that that block-code applies to transmission of fixed size data for example IP packets, and the convolutional codes apply to streams of arbitrary length. Since the FEC is to work on packets of fixed length, the convolutional codes are not considered for the project.

The block codes considered include the Reed-Solomon algorithm, and the Golay and Hamming codes.

A simple and typical overview of how the FEC is applied to the data is shown below:

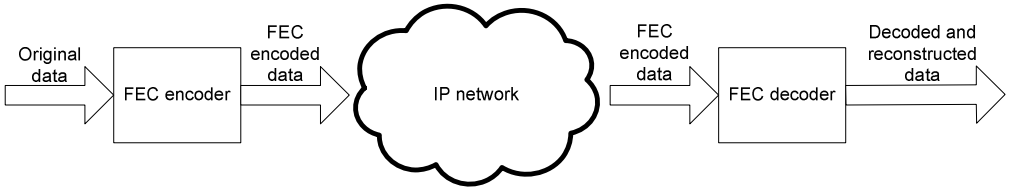


Figure 1.2 - FEC overview

The original data is fed into the encoder, which of course encodes the data and then applies a FEC code. The FEC code can consist of extra packets sent, or extra data in the packets and so on. Then the packets including the FEC is sent over an IP network and received in a decoder at the other end. The decoder makes sure that any lost packets or errors are corrected based on the FEC code transmitted with the original data. Also the decoder transforms the received data into an audio visual signal which can be shown on for example a television.

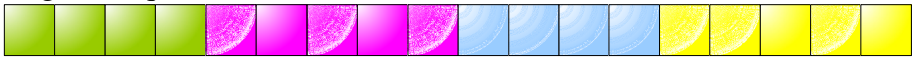
The encoding of the original data and the encoding of the FEC can be separated into two separate terms. The first is called source coding and at this stage the original data is encoded and with this, as much redundant information is removed from the signal as possible. The second stage is also called channel encoding, and at this state a controlled amount of redundancy is added to the compressed signal to allow for error correction at the receiving end. The corresponding decoding stages are of course present at the decoding end [4].

1.4.2 Interleaving

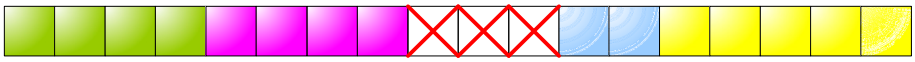
Interleaving is a simple method for error protection which can effectively be used with other FEC solutions. The use of interleaving enhances the ability to correct especially burst errors. The principle in interleaving in the case where it is used with FEC is to increase the chances of correcting the errors. There is always a limit to how many consecutive errors an error correcting code can correct. Of course there is also a limit to how many random errors the code can correct, but it is difficult to work around this.

The interleaving is basically the idea of not sending all information in the order it is to be used. You divide the data into blocks and interleave them with each other. The principle is shown below.

The original sequence to be transmitted:

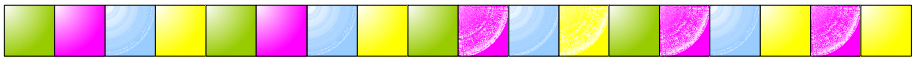


A burst loss occurs in the transmission and three blocks are lost or damaged. That is packets, bits, or bytes.



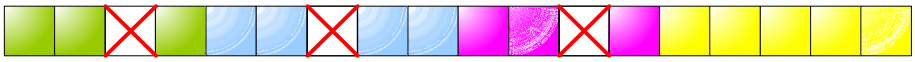
It is impossible for the receiver to know whether the lost or altered blocks should be red or blue from the sequence since the receiver does not know how many red and blue blocks there are supposed to be.

If the blocks were interleaved and sent one at the time in each colour, then the pattern sent would look like this:



Based on the same burst loss on the transmission the receiver would see this:

After reordering them in the right order, the received pattern looks like this:



From this pattern it is easy to see the corrupted blocks and to colour them in the right colour again.

For use with other error correcting solutions the example shown above would mean that a burst error would suddenly look like random errors in the transmitted signal, and would then in many cases be easier to correct.

1.4.3 Exclusive or

This type of error correction is fairly simple, since it only consists of an exclusive or (XOR) operation applied to a number of consecutive packets. There is several ways to implement this solution both in terms of the number of packets to include in each FEC-packet and the number of dimensions to apply the operation to.

1.4.3.1 One dimension

When the operation is applied to a number of consecutive packets it can be regarded as one dimensional.

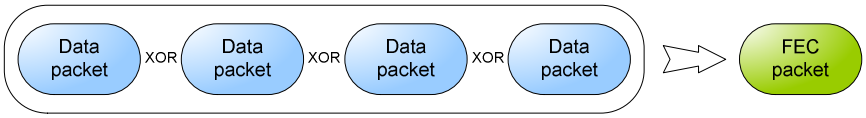


Figure 1.3 - One-dimensional FEC

The XOR operation is a simple bitwise exclusive or, and the resulting packet is the FEC packet which is sent along with the other packets. This means that there will be some overhead on the transmission of the packets which has to be taken into consideration when deciding on the design of the solution.

In the case shown above, four packets are sent with one FEC packet. This produces 20 % overhead in the transmission, which of course can be lowered by generating the FEC packet over a larger number of packets, but this also produces a problem; the one dimensional XOR solution can only reconstruct one lost packet per FEC packet. This means that if two packets belonging to the same FEC packet are lost in the transmission, there is no way to reconstruct them.

1.4.3.2 Two dimensions

Adding another dimension to the solution, and hereby making it two dimensional, will increase the number of packets that can be reconstructed in case of packet loss. The data packets are arranged in a matrix and each row and column produces one FEC packet.

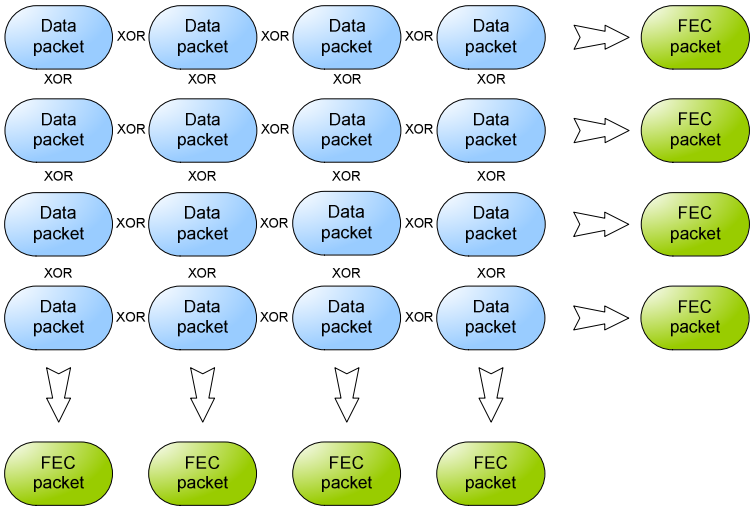
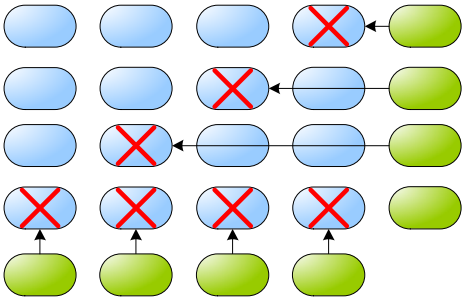


Figure 1.4 - Two-dimensional FEC

With this arrangement there is still some overhead on the transmission but the ability to reconstruct multiple lost packets has increased significantly. The transmission overhead with this matrix is 33% but can again be lowered by choosing another number of data packets in the rows and columns.

On the other hand, the number of packets that can be reconstructed has increased from one packet in the one dimensional arrangement to a maximum of seven packets by correcting the packets in the right order. This means that it is possible to correct a packet loss of at most 43% of the original data packets if the packets are lost "correctly" in the matrix, and minimum four packets corresponding to 25 %.



The row FEC enables the recovery of non consecutive packet loss whereas the column FEC enables the recovery of consecutive packet loss of at most the number of packets in a row.

Research results [5] have shown that one-dimensional parity FEC offers good error correction for MPEG-4 only when the loss rate is in the range of a few percent. Otherwise a combination of FEC and other methods for minimizing packet loss, for example interleaving, is necessary.

	Loss rate	0.5%	1%	3%	10%
FEC	*Overhead*				
3	33%	100%	100%	96%	86%
7	14%	100%	90%	85%	50%
15	7%	100%	89%	70%	21%

Figure 1.5 - Research results for the recovery rate, one-dimensional FEC

The results shows that only networks with very low error rates or little requirements for the consistency of the received data should use a one dimensional FEC, unless it is combined with for example interleaving.

1.4.4 Reed Solomon

The Reed-Solomon coder (RS) is defined as a systematic block code [6][7], since it handles data in blocks of a fixed size and the original data is sent untouched but with added data for the error correction. These blocks could be IP packet payloads to be sent on an IP network. Like the XOR code, the RS code adds some redundant data to the information to be sent. This redundant data can help correct the errors that may occur in the course of the transmission.

The algorithm is a subset of BCH codes and is defined as linear block codes. The algorithm is specified as RS (n, k) with m bit symbols, where k is the original message length in symbols of m bit. N is the code length in symbols after the redundant data have been added which can have the size of up to 2^m-1. This means that for each coded block there is r = n-k check symbols of m bits. The algorithm can effectively correct up to t symbols that are erroneous in the transmitted data, where t is ½(n-k).

The value of n can vary, but can not be greater than 2^m-1. If a smaller value is chosen then both the encoder and decoder should append the number of zero's corresponding to (2^m-1) – n.

It should be noted that a large difference between n and k will result in correction of a large number of errors, but also it requires a large amount of computational power. These two things are mutually exclusive and one should chose carefully which property should weigh more; little computational power with little error recovery or the opposite.

The code is produced by constructing a polynomial from the bits in the data. This polynomial is then sampled, and even over-sampled to produce sufficient data to correct errors from the rest of the data. When the polynomial is over-sampled you simply evaluate it at more points than actually necessary, and from these extra points you can correct the data.

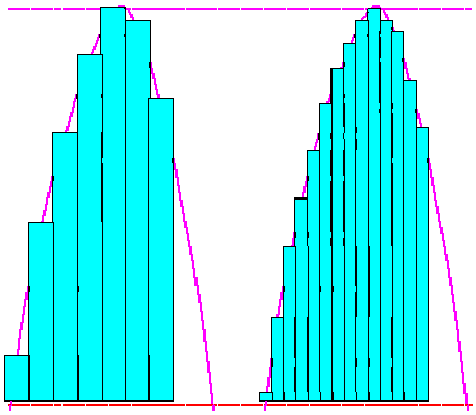


Figure 1.6 - Over sampling

Here, the left hand side has just the right amounts of samples whereas the right hand side polynomial is over-sampled and thus the loss of one value in the right hand side signal would not result in the wrong polynomial.

This algorithm is most effective on burst loss but only if the bursts are less than t consecutive bytes. Since bursts can vary a lot in length, the RS code could be combined with interleaving which would increase the number of consecutive errors that can be corrected. The principle is shown below:

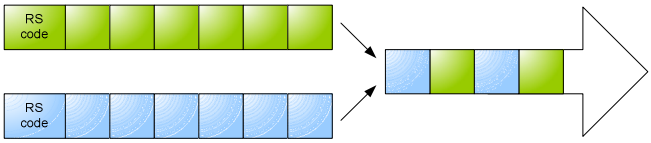


Figure 1.7 - Interleaving

Here the individual blocks are interleaved, in the example above they only interleave by a factor 1:2 but the interleaving could be larger. The bigger interleaving the more robust the code is in case of burst errors. If you have a code that can correct for example eight consecutive byte errors then an interleaving of 1:2 as shown above would enable the correction of sixteen consecutive errors.

1.4.5 Golay and Hamming code

These two error correcting codes both operate on bit level [6][7], which makes them unsuitable for this project. All data is handled as packets, be it TS packets or IP packets. The network which the analysis is based upon, described in section 1.8.1 handles bit errors in the transmission by dropping the IP packet. Therefore an error correcting code based on bit is not preferred.

1.5 RFC 2733

The RFC 2733 [3] (RFC) is a proposed standard released by the Internet Engineering Task Force (IETF) developed by the Network Working Group, all a part of The Internet Society[3]. The RFC's are notes with Requests for Comments and play an important role in the process of defining internet protocols and standards.

The RFC is designed as a generic error correction payload format which means that the protocol satisfies the following properties:

1) Independent of the nature of the media being protected, be it audio, video, or otherwise
2) Flexible enough to support a wide variety of FEC mechanisms
3) Designed for adaptivity so that the FEC technique can be modified easily without of band signalling
4) Supportive of a number of different mechanisms for transporting the FEC packets.

[3] http://www.isoc.org/

For the generation of the FEC packets, the RFC proposes the use of the XOR or parity FEC previously described, but marks that any FEC code could be used. The RFC proposes several different schemes for the implementation of the XOR FEC which are all described superficially. The general scheme shown below is the one used through out the note:

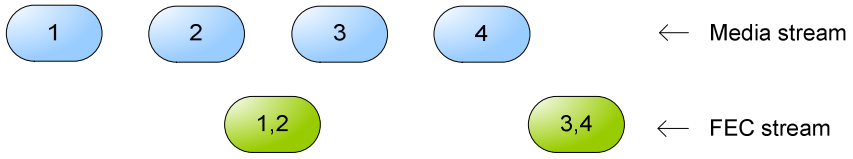


Figure 1.8 - Proposed FEC

This scheme is rather simple but efficient. It is a one dimensional FEC scheme as previously described. The other schemes shown below are all a little bit more complicated and will be slightly more difficult to implement.

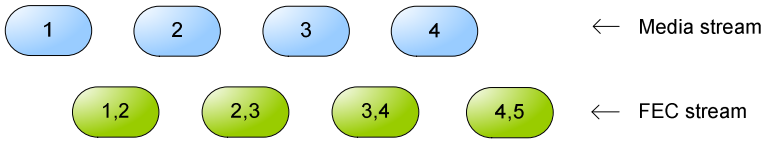


Figure 1.9 - FEC scheme 1

The first scheme is very simple and introduces quite an amount of overhead. The scheme allows for the recovery of up to two consecutive packets.

Figure 1.10 - FEC scheme 2

This specific scheme differs from the others by only transmitting FEC packets. The scheme has less transmission overhead than the scheme above, but is able to correct burst errors of up to three consecutive packets depending on the timing.

The recovery is quite time consuming since one packet affect four FEC packets, and therefore many packets has to be received before decisions can be made.

Figure 1.11 - FEC scheme 3

The last of the three schemes can recover from up to three consecutive packet losses but in addition the receiver has to wait for four packets after the lost packet.

The FEC packets of the RFC are constructed by forming a FEC header and a payload for the packet. The details of the format are described in a later section.

1.6 Pro-MPEG

The Pro-MPEG Forum, formed in 1998, is an association of broadcasters and program makers, equipment manufacturers and component suppliers with interests in realizing interoperability of professional television equipment, according to the implementation requirements of broadcasters and other end-users.

The Forum was formed to support open standards for emerging new professional television applications and has over 130 members from all over the world.[4]

The Pro-MPEG forum has amongst other things developed a recommendation to the transmission of MPEG-2 streams over IP networks [8], also called Pro-MPEG Code of Practice #3 release 2 (CoP). This includes some recommendations as to the use of forward error correction when transmitting media streams. The CoP is targeted towards MPEG-2 but there is no reason to why it would be any different when handling MPEG-4.

[4] http://www.pro-mpeg.org/

These recommendations are very important for the developers of both encoders and decoders. This is because the manufacturers will want as many as possible to be compatible with their product. The more manufacturers that follow some code of practice, the more compatibility they have with other manufacturers.

The CoP is basically an extension to the RFC 2733 described in the previous section, with some alterations to enhance the ability to correct errors. The FEC scheme they propose is a two dimensional XOR algorithm with several possibilities as to the size of the matrix. Their minimum implementation is a one dimensional FEC, but done across columns instead of rows, as the RFC 2733 proposes. The constraint for the Pro-MPEG scheme is that there has to be at least one column and at most 20 and there can be at least four rows and at most 20. Also the total number of packets in one matrix can not exceed 100 packets.

The only thing not included in the CoP version of the FEC solution which is found in the RFC is that the RFC FEC applies the FEC operation on some of the RTP headers as well as the payload which the CoP does not.

The CoP notes that it is important that the packets are not fragmented during transmission and therefore it is very important to pay close attention to the IP headers. Furthermore the recommendations require the use of the RTP header for the transmission, but this only makes sense for all FEC solutions.

The CoP recommends that equipment supports IP packets containing one, four, and seven TS packets but can of course support other sizes, and the equipment has to support 188 bytes in each TS packet, but it is optional to implement compatibility with the 204 bytes format.

The RFC 2733 RTP header is suggested as the header format, which contains the ordinary RTP header, a FEC header and then the payload, but since it only supports 24 consecutive packets to each FEC packet, an extension is proposed. The extension handles this problem by adding 32 additional bits to the FEC header which makes up several fields where some are saved for further extensions.

In the building of the FEC packets there are some rules, which include several concepts. The matrix is arranged with a width L and a depth D which corresponds to the width and depth of the matrix in packets. The values of L and D have the following constraints:

$$L \cdot D \leq 100$$
$$1 \leq L \leq 20$$
$$4 \leq D \leq 20$$

With these constraints it is optional weather to only calculate a column FEC or both the column and the row FEC. If both column and row FEC are constructed, then L must be equal to or larger than four, otherwise only column FEC should be calculated.

In the description of the proposed solution, the interpretation can be discussed. The CoP describes that when only sending the column FEC, then the FEC is calculated based on D consecutive packets, and therefore only one packet loss per FEC can be corrected. This can be interpreted in two ways, one that suggests that when calculating only row FEC the matrix is also one dimensional, and one that suggests that the matrix can also be two dimensional and is therefore able to correct burst errors. This would mean that a packet loss of up to L consecutive packets can be corrected.

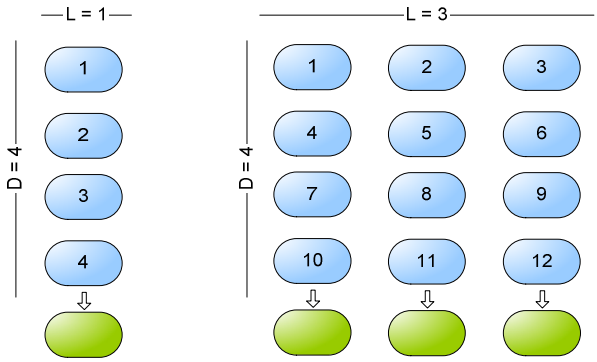


Figure 1.12 - Pro-MPEG matrix arrangement

With the matrix above with L = 3 and D = 4, three consecutive packets can be corrected. This also questions the CoP in their claims that one dimensional FEC is effective when you experience random errors but only the two dimensional is effective in the case of burst loss. Actually the ability to correct burst loss is determined only by the L variable.

When transmitting the original packets and the FEC packets it is suggested that three ports are used to receive the streams. Of course the original port to receive the original packets and then two ports each to receive either the column FEC or the row FEC. The reason for using two ports for the reception of the FEC stream is to support implementations using only one dimensional FEC.

The CoP includes a table showing the overhead, latency, recovery capacity, and the required buffer size. See the CoP [8] for the table. The CoP has been quite optimistic in their calculations and the following table shows some further calculations. There is a requirement to the proposed standard which states that the column FEC is sent no sooner than L packets after the last IP packet used to form the FEC packet, and no later than L·D packets after. This last criteria sets the size of the buffer in the decoder which has to be able to receive L·D packets before the last FEC packet since it is determined by the encoder, when the FEC packets are sent.

Also the last criteria determine the delay in the decoder when receiving the packets. The maximum delay is compared to the size of the buffer in the decoder and the transmission overhead caused by the FEC.

When looking at the receiving side, this requirement demands at least twice the buffer size in the decoder and the overhead calculated in the table is the overhead in the transmission; how many packets can be lost and still recovered that is. If the overhead in the transmission, meaning the extra bandwidth used by the FEC packets, is calculated, the picture is a bit different.

Table 1 - Transmission overhead and buffer size in the decoder with Pro-MPEG FEC

	Recovery (CoP)	Transmission Overhead	Buffer size, bytes. (CoP)	Buffer size in decoder, bytes.	Delay in decoder, ms.
XOR(5,10)	10 %	23 %	66400	195000	104
XOR(10,10)	10 %	16,7 %	132800	360000	192
XOR(20,5)	20 %	20 %	132800	375000	200
XOR(8,8)	12.5 %	20 %	84922	240000	128
XOR(10,5)	20 %	23 %	66400	195000	104
XOR(8,5)	20 %	24.5 %	53120	129000	68,8
XOR(5,5)	20 %	28.6 %	33200	120000	64
XOR(4,6)	16.5 %	29.4 %	31872	102000	54,4
XOR(6,4)	25 %	29.4 %	31872	102000	54,4

The delay is calculated by using a bandwidth of 15 Mbps which is a channel width often used for this type of transmission. From the bandwidth the number of packets received each second can be determined.

$$15Mbit \Leftrightarrow 1{,}875Mbps$$

$$\frac{1{,}875Mbps}{1500\,B/packet} = 1250\ packets/s$$

These calculations are very important since they describe how much memory is needed in the decoder and what affect the FEC has on the bandwidth. The buffer size in the decoder is calculated by the fact that the maximum number of packets between the received matrix of original packets, and the last FEC packet belonging to this matrix is L·D. This is actually a whole other matrix and it means that the memory in the decoder has to be twice as big as the matrix itself. For the sizes calculated a general packet size of 1500 bytes has been used as it is just to prove a point.

All calculations above are worst case scenarios, since the decoder has to be able to handle these situations. Of course there are cases where the delay is smaller depending on the time of arrival of the FEC packets and of course the number of lost packets.

The table shows that the memory use is not too big and that all the matrix sizes proposed can be handled with only 512 kB RAM.

Compared to the 512 kB RAM which could handle all the matrices, even 256 kB RAM would be able to handle all but two of the matrix sizes, and with 128 kB three matrix sizes would still be covered.

This means that the memory size might not be the biggest problem, even for consumer products. Instead the transmission overhead and the delay in the decoder cause bigger problems.

Some of the matrix sizes are fairly decent with the transmission overhead compared to the number of packets that can be recovered, but some of them are not too optimal. For example in the first matrix proposed, the recovery rate is 10 % but the overhead in the transmission is 23 %. That is 13 % overhead which has no direct effect. There are other matrix sizes which recover the same percentage of packets as their transmission overhead represents. In this case it would be easy to argue that the matrix is usable, but in the other cases it is unacceptable.

In the decoder there is a delay for collecting a whole frame of the audio visual signal of approximately 40 ms without the FEC applied to the signal. The delay depends on the frame rate, and the 40 ms is the standard for the PAL signal. Besides this delay, there is also a delay in the encoding process, depending on the encoding scheme. The total delay in the application is important especially when dealing with contribution purposes because two-way traffic occurs when for example returning the sound. For this a delay of the return sound of even less than a second would be very distracting for a speaker or the like.

When looking at the delay caused by the FEC, only the smallest matrices have acceptable delays compared to the delay already in the decoder. The large matrices have delays which are up to five times bigger than the delay in the decoder, and it would have a very noticeable effect on the signal passing through the decoder.

These delays are of course calculated based on a bit rate of 15 Mbps. The delay is directly dependent on the bit rate, with a low bit rate the delay increases. With a signal of 7 Mbps, a bit rate also used quite often, the delay would be ~429 ms which mean that the total delay in the decoder would be almost ½ s.

1.7 What do others do?

1.7.1 Tandberg

Tandberg is one of Scientific Atlanta's biggest competitors in the encoding and decoding industry. Scientific Atlanta has a few Tandberg encoders, and of these encoders only one has a FEC option. The output of this encoder has been examined to determine the type of FEC solution the encoder provides.

The encoder has several FEC setups and the IP output from these different options has been analyzed to try and bring some light on the FEC solution they have implemented.

The resulting IP packets from some of the settings in the encoder are included in appendix 1. The output has been generated from a completely black picture which is why all bytes have the value 0xFF. To the output signal are appended null-packets, which is a kind of filler.

When comparing the output of the four different outputs, no pattern can be detected, which is why the analysis of the Tandberg encoder was not of any use to the project. But also, it would be hard to detect other than a simple interleaving or the XOR solutions from the IP packets, and none of these seem to have been used. It is only fair to assume that Tandberg has implemented a proprietary solution.

1.7.2 TUT systems

TUT Systems claim on their internet site that they have implemented the CoP3 solution recommended by Pro-MPEG. The solution is superficially described in a datasheet and impossible to get any usable information from.

Since Scientific Atlanta has no encoders from TUT Systems, no research has been performed to find out which FEC solution is actually implemented in the encoder and if it in fact does as it is supposed to do.

1.8 What is the need at Scientific Atlanta?

All of the codes raise some questions when used for FEC on IP packets. The most obvious question is which code to implement, and the next question is: on which level should the code be implemented; on the IP packet, the IP payload or on the individual TS packets?

To start off by answering the second of the two questions, we can easily rule out at least one option. It would be useless to apply a FEC code on the whole IP packet since the IP headers are changed by routers in a network. This would mean that as soon as the packets meet a router, the packets are changed and the FEC codes can no longer be used to reconstruct lost or erroneous packets.

To rule out one of the two remaining options it is necessary to look at the different types of errors in an IP network. In section 1.3 are listed some of the most common errors that are likely to occur and the two major types are bit errors and packet loss.

If the FEC code is applied on TS packets then bit errors would cause no problems as long as it is within the limitations of the FEC code. This could be a good solution but the problems arise when packet loss is taken into consideration. If a whole packet is lost then there is no possibility of recovering the lost TS packets.

This problem though, could be solved by interleaving the TS packets. If an IP packet was lost then the other IP packets with TS packets belonging to the same FEC code, could be used to reconstruct the lost TS packets. And the lost TS packets would each belong to different FEC codes. It should be noted that this solution depends on the FEC solution implemented.

To answer the first question it is necessary to construct a table showing the characteristics of each FEC solution.

Table 2 - FEC codes

Errors / FEC code	Bit errors	Packet loss
Reed-Solomon	✓	✓
XOR	✓	✓
BCH	✓	×
Golay	✓	×
Hamming	✓	×

If the Reed Solomon algorithm was to be used, the block size would have to be the size of a whole IP packet or the payload, which is the data part of an IP packet. If less, then only errors within the packet could be corrected. This is not enough in case of a packet loss.

If the block size corresponds to an IP payload, then the number of blocks in the code should be reduced, to reduce the computation. The Reed Solomon code requires quite some calculations which could be bought as a chip and embedded in the encoder. This is pretty expensive, in both processing time and money.

The XOR algorithm is far simpler and very fast. It is very easy to implement in hardware and offers approximately the same error correction. Also it has to be taken into account that there are both the RFC and the suggestion to a FEC solution by Pro-MPEG. This has something to say since recommendations like for example the Pro-MPEG is developed by manufacturers in the same industry, and there is reason to believe that these FEC schemes will be more popular than others.

The choice, of which algorithm to implement, is therefore a one-dimensional FEC solution, based on the RFC, but constructed so that it also is compatible with the constraints listed by the CoP. This means that the FEC solution is actually a CoP FEC scheme, but since it only has one dimension, it equals to the FEC proposed by the RFC.

1.8.1 Applications

To determine the applications where a FEC solution would be needed, several people were contacted within Scientific-Atlanta and Cisco[5]. In the search for an application where error correcting was needed, it became clear that no one had any suggestions for this. The problem with the applications are that the ones presented all included either very little physical or very secure network between the encoder in which the FEC is implemented and other nodes in the application, all of which either stripped the IP header off the packets and applied their own header, or included extensive FEC solutions for further transmission. This would mean that the FEC code would be stripped off and never used or just be overwritten by new FEC data and therefore never used.

Also as Rudi Van de Genachte, Application Engineering Manager, Hybrid Fiber Coax Networks, from Scientific-Atlanta in Belgium, describes it:

> *"From my experience in IPTV, I have to admit that PRO-MPEG FEC isn't much of demand. The reason for this is that IP networks can be easily be scale to carry the traffic needed for the TV services. So, packet drops and error packets can be avoid by engineering correctly these networks."*

The point of this is that there are ways of sending data in an erroneous network, without losing too much data, by simply scaling down the transmitted data according to the channel. The chance of an error in a signal of only 50 % or less of the bandwidth of the actual network is small. Therefore many companies only use 40-60 % of the actual bandwidth available to avoid data loss in the transmission. This is of course not preferable since it does not utilize the connection and the result of this is that money is lost on the unutilized part of the connection.

[5] Correspondences in appendix 2

It was necessary to provide a hypothetic network which could be a part of a plausible application with an erroneous network, and therefore would need the FEC. This network was found as a business network solution provided by the Danish telecommunication company, Tele-Denmark Communications A/S (TDC), and will be used as worst case scenario in reference to the error rate of the connection.

Figure 1.13 - EtherLink network

The network found is considered to be a fair representative since the scope determines that the networks on which the FEC solution is based are dedicated and of a certain quality. The TDC network is a point-to-point network with a bandwidth of 40 Mbps, which is very plausible.

The network could be a part of an application from for example a studio, where the signal is encoded and then transmitted to the TV network station, where the signal is decoded and subtitles and or other artificial or real objects can be added to the signal. The TDC network would then represent the network connection between the studio and the network station.

In reality, it is more plausible with even more dedicated networks like fiber or the like. These have even less errors than this hypothetical network.

To determine the appropriate size of the XOR algorithm; that is the number of errors the code would have to be able to correct, TDC was contacted to try to get some information on the error rate of some of their communication lines.

TDC claims to perform better than 10^{-12} on their SDH lines but has only measured a performance better than 10^{-10} because of the duration of the test period. The two error rates result in:

$$\frac{1}{40\,Mb/s \cdot 10^{-12}\ errors/bit} = 25000\,s/error \approx 7\,h/error$$

$$\frac{1}{40\,Mb/s \cdot 10^{-10}\ errors/bit} = 250\,s/error \approx 4\,m/error$$

The results mean that there is one error per seventh hour and 1 error per fourth minute respectively. The test is based on ten errors to rate the network and ten errors in a network with 10^{-12} errors results in a test period of more than 70 hours. The errors occurring in the network are random errors. The error rate is based on the EtherLink point-to-point network provided by TDC[6] and this is the network used as a hypothetical network for the FEC. This means that the error rate of the EtherLink network is what the performance of the FEC is compared to.

In the datasheet for the EtherLink network the error rate is not mentioned and the consequences of an error is not mentioned either. TDC has therefore been contacted and one of their technical consultants, Bjarke Skoldstrup, was able to ensure the error rates. Also he explained what happens in case of a bit error in the transmission in which case the error detection on the SDH network detects the error in the packet by calculating a check sum an if there is an error then the packet is dropped.

The fact that the packet is dropped by the Ethernet Bridge is of great importance because it limits the choice of FEC algorithm to those which can correct packet loss.

When trying to bring some light on the applications in which the implemented FEC could be used, a measure came up, which is very important to the project. It should though be noted that the measure was only mentioned by a product manager as a rule of thumb, and is not official in Scientific Atlanta or Cisco. The measure was that in broadcasting television, less than one "visible artefact", some visual or audible disturbance in the received signal, per hour would be acceptable in the signal.

To make sure that this measure is met by the FEC algorithm, every bit error counts as a visual artefact and should therefore be corrected. This might not always be true.

[6] http://download.tdc.dk/pub/tdc/erhverv/faktablade/faktablad0890.pdf

This loose measure compared to the EtherLink network provided by TDC means that if all the bandwidth is used for transmitting a signal then the FEC would have to be able to correct:

$$0.004\,errors/s \cdot 3{,}600 = 14.4\,errors/h$$

For at typical contribution transmission, the bandwidth needed is about 15 Mbps and this will be the basis of the calculation of the performance of the FEC. Also the error rate will be set to 10^{-10} for the network which is guaranteed by TDC. With 15 Mbps signal and 10^{-10} errors, there is an error rate of:

$$\frac{1}{15\,Mb/s \cdot 10^{-10}\,errors/bit} = 666.\overline{6}\,s/error \approx 11\,m/error$$

$$\Updownarrow$$

$$0.0015\,errors/s \cdot 3{,}600 = 5.4\,errors/h$$

So the FEC has to be able to correct at least 5.4 errors per hour.

1.8.2 Trade offs

The optimal solution is to only correct the 5.4 packets described in section 1.8.1 which would mean that six packets per hour should be corrected, corresponding to six FEC packets sent per hour. When using this as a measure then the calculation for a one-dimensional XOR would be:

$$15Mbit \Leftrightarrow 1{,}875\,MB/s$$

$$\frac{1{,}875\,MB/s}{1{,}500\,B/packet} = 1.250\,packets/s$$

$$1{,}250\,packets/s \cdot 3{,}600 = 4{,}500{,}000\,packets/h$$

$$\frac{4{,}500{,}000\,packets/h}{6\,FEC/h} = 750{,}000\,packets/FEC$$

This would correct the exact amount of lost packets in the stream in a perfect system with perfectly distributed errors on the network. This however is not the

case in most networks where errors might not be evenly distributed over time. Another problem is the decoder, which would then have to buffer 750,000 packets before being able to correct a lost packet. This would bee too long since it would introduce a huge delay while the decoder collects the packets.

The choice of the size of the FEC is therefore a trade off between the delay in the decoder, the amount of memory needed in the decoder for buffering, the number of packets to correct caused by errors in the network, and the redundant FEC data, added to the signal to be transmitted.

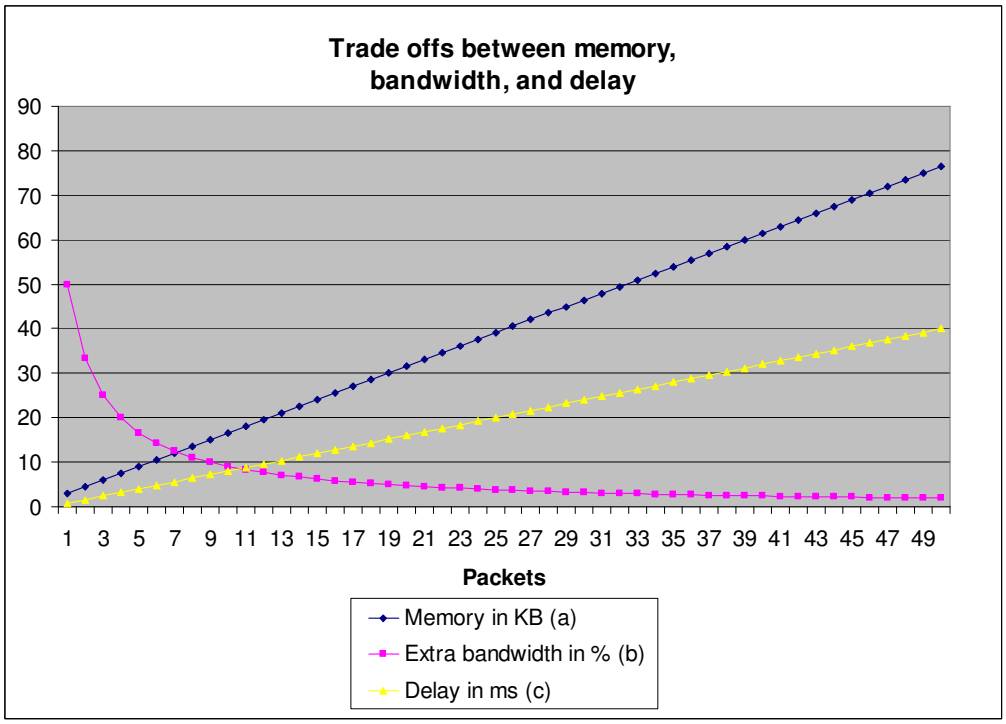


Figure 1.14 - Trade offs

It is important to describe the series both as individuals but also take into consideration, the influence they have on each other since this is what a conclusion as to the width of the FEC should be is based upon.

The series (a) shows the memory usage in the decoder and this is a very important property to take into consideration since memory is very expensive, and the decoder should be held at an absolute minimum. The memory usage shown in the graph though is not big, which is why this property does not set the limit to the size of the FEC. The (b) series describe the amount of

bandwidth used to transmit the FEC packets over the network. The bandwidth should also be kept at a minimum, since the service provider or the contributor pays for the network, and each FEC packet transmitted and never used is redundant and costs money for the transmitting end. Finally the (c) series describe the delay in the decoder caused because the received packets has to be buffered until all packages belonging to a single FEC packet, and the FEC packet itself has to be received and potentially used to correct a packet loss. The reason for keeping the delay in the decoder low was described in section 1.6.

The table below shows the different properties at different delays and widths of the FEC, only some of the FEC widths are shown since the properties are quite close to each other by only adding one packet to the width. For further results please see appendix 4 and the chart above, which shows the trade offs compared to each other.

Table 3 - FEC distributions

FEC width	Delay	Corrected packets	Extra bandwidth	Bandwidth total	Corrected packets/hour	Decoder memory
1	0,8 ms.	100,00%	50,00%	22,50 Mbit	4500000	3 KB
5	4 ms.	20,00%	16,67%	17,50 Mbit	900000	9 KB
10	8 ms.	10,00%	9,09%	16,36 Mbit	450000	16,5 KB
15	12 ms.	6,67%	6,25%	15,94 Mbit	300000	24 KB
20	16 ms.	5,00%	4,76%	15,71 Mbit	225000	31,5 KB
25	20 ms.	4,00%	3,85%	15,58 Mbit	180000	39 KB
30	24 ms.	3,33%	3,23%	15,48 Mbit	150000	46,5 KB
35	28 ms.	2,86%	2,78%	15,42 Mbit	128571	54 KB
40	32 ms.	2,50%	2,44%	15,37 Mbit	112500	61,5 KB
45	36 ms.	2,22%	2,17%	15,33 Mbit	100000	69 KB
50	40 ms.	2,00%	1,96%	15,29 Mbit	90000	76,5 KB

With ten packets to construct one FEC packet, 450000 lost packets can be corrected per hour. This corresponds to an error rate of more than 10^{-6} errors/s. This is a big improvement compared to the actual error rate of 10^{-10} which the network from TDC could provide. This means that the actual network which supports the FEC can have an error rate of 10^{-6}.

1.8.3 Transmission

The previously described IETF standard [3] the transmission of FEC packets on an IP network using the RTP protocol is considered. The standard describes the correct RTP payload format and should be used for the transmission of FEC packets. The standard describes the use of the XOR FEC, but the header is generic, the only limitation being that it allows for no more than 24 consecutive packets to form one FEC packet.

The standard ensures that the receiver knows which packets have been used to generate the FEC packet in the case where the FEC packet arrives out of order by utilizing the RTP header.

The FEC packets are sent as a separate stream, and the original IP packets are sent as if there were no FEC.

In the general format described by the RFC, a FEC header as well as the FEC payload is placed inside the RTP payload:

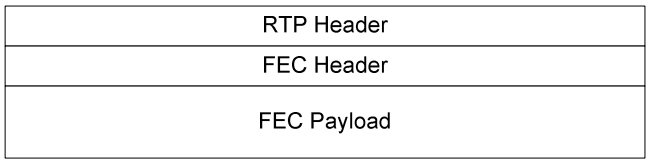


Figure 1.15 - RTP packet with FEC payload

The RTP header consists of the following fields:

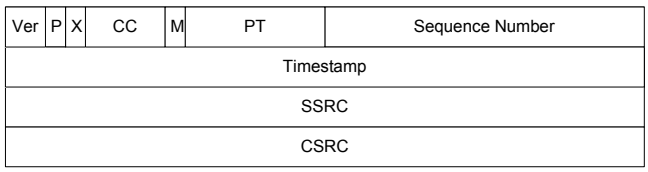


Figure 1.16 - RTP header

Most of the values in the RTP header are given by the RFC and the CoP and are just listed. When the two papers disagree, the CoP has precedence, since the chance of interoperability with other manufacturers is greater.

- The Version field is always set to 2.
- The Padding bit, the marker bit, and the Extension bit are computed via the protection operation according to the RFC, but are set to zero in the CoP.
- In the RFC, the SSRC value will generally be the same as the SSRC value of the media stream it protects, but according to the CoP, the field is not used, and can therefore have any value.
- The CC value is also set to zero in the CoP, defining that there are no contributing sources – which ultimately means that the CSRC field is not included.
- The sequence number has the standard definition: it must be one higher than the sequence number in the previously transmitted FEC packet.
- The timestamp is not used by the CoP and can be set to any desired value.

The FEC header described by the RFC has the following fields:

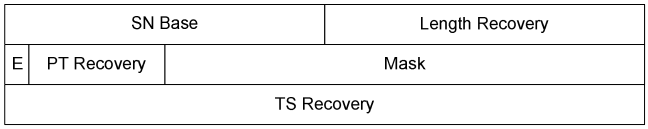


Figure 1.17 - RFC FEC header

The following fields are the same for both the RFC and the CoP FEC headers [3][8]:

- SNBase low bits: minimum sequence number of the packets associated to the FEC packet. For MPEG2 transport streams 16 bit sequence numbers are sufficient, so this parameter shall contain the entire sequence number. For protocols with longer sequence numbers this field will contain the least significant 16 bits of the sequence number.
- Length Recovery: this field is used to determine the length of any media packets associated with the FEC packet.
- PT recovery: this field is used to determine the Payload Type of any media packets associated with the FEC packet.
- TS Recovery: this field is used to recover the timestamp of any media packets associated with the FEC packet.

The Mask field is the most important reason for creating an extension to the RFC FEC header. The field is 24 bits and if bit i in the mask is set to 1, then the media packet with sequence number N + i is associated with this FEC packet. N is the sequence number of the first packet, and this limits the use of media packets used in the FEC to 24 (0 to 23).

To enable the FEC to cover more than 24 consecutive packets, the CoP has proposed the following extension to the RFC FEC header:

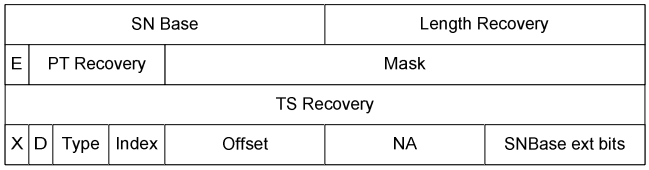


Figure 1.18 - CoP extended FEC header

The following fields are defined by the CoP to have different values than the RFC or have been added to the header – the definition of the fields is taken directly from the CoP [8]:

- E: In RFC2733 this shall be set to '0', in this code of practice this shall be set to '1' to indicate that the header is extended.
- Mask: In RFC2733 this is used to select which packets the FEC packet is applied to. The definition of the mask allows for a complex relationship between data packets and FEC packets, but this adds to implementation complexity. For simplicity, the mask field will be set to zero for implementations supporting this code of practice, and the NA field will be used instead. Handling of Mask requires special care due to the change of use from CoP #3 January 2003.
- X: This bit is reserved for future header extensions and must be set to zero to conform to this version of the FEC header.
- D: This bit is provided as an additional means of determining which FEC stream the packets belong. It must be set to 0 for FEC packets computed on columns and set to 1 for FEC packets computed on rows.
- Type: This field indicates which error-correcting code is chosen. It can be XOR (type=0), Hamming (type=1), Reed-Solomon (type=2). More encoding techniques can be used. For this version of the Code of Practice equipment shall only use the XOR type.
- Index: This field is used for more complex error protection codes. For the XOR method, only one FEC packet protects each group of media packets and hence the index field will always contain 0.

- Offset: This 1-byte field is the period chosen to select the media packets associated with this FEC packet, and corresponds exactly to the L parameter above for packets computed over columns (the first FEC stream). For packets computed over rows (the second FEC stream) this parameter shall always be one. This field should be kept constant during a session for each FEC stream.
- NA: This 1-byte field indicates the number of media packets associated with this FEC packet, and corresponds exactly to the D parameter above for packets belonging to the first FEC stream, and should correspond to the L parameter for packets belonging to the second FEC stream. This field should be kept constant during a session for each FEC stream.
- SNBase ext bits: This field is reserved for use with protocols which require extended sequence numbers longer than 16 bits. For MPEG2 transport streams 16 bit sequence numbers are sufficient, so this parameter shall be set to zero.

Furthermore the RTP packets should be encapsulated in both the UDP and the IP header. The UDP header provides a checksum which makes it possible to detect bit errors in the stream.

The header looks like this:

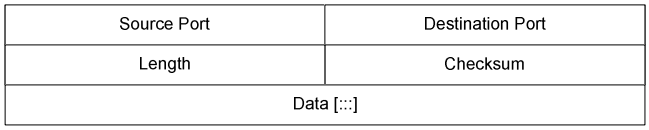


Figure 1.19 - UDP header

Finally everything is encapsulated in the IP header, which will not be described here. The header does not provide any special properties to the FEC, but provides a transport protocol for the IP network on which the packets are to be sent.

The resulting encapsulation of the FEC packets is shown below, where a FEC packet corresponds to several TS packets XOR'd together to form the FEC packet.

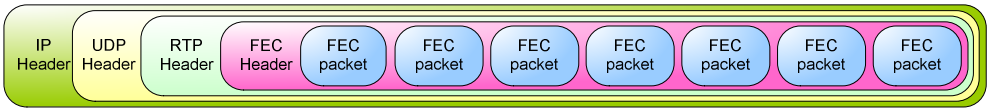


Figure 1.20 - FEC encapsulation

1.9 Summary

From the analysis performed the conclusion is to implement the one dimensional XOR solution using a distribution of 10 packets to form each FEC packet. This means that there will be an overhead on the transmission of 9.09 % and one packets in ten packets sent can be reconstructed. The solution is able to correct a sufficient amount of errors when basing the error rate on the presented hypothetical network as described by TDC, which could be a part of a plausible application.

The solution only discusses the memory use and the resulting delay in the decoder, but is not based upon the actual values, although both properties are very important. The delay is small but might be smaller at the cost of bandwidth usage, and the memory depends on size of the FEC matrix.

Furthermore the solution is based on the RFC described, and actually satisfies the Pro-MPEG CoP suggestion which means that any decoder supporting the Pro-MPEG FEC will support this implementation of FEC. It also more than covers the need for error correction in most types of networks with the ability to correct errors of a rate of more than 10^{-6}. It will use the standard format proposed by the RFC and extended by Pro-MPEG for the payload.

In the network described, packets are dropped if any errors are detected in the headers; therefore there is no reason for applying the FEC to the headers as well as the payloads.

CHAPTER 2

Design

This section covers the design of the system for the D9054 HD h.264 Encoder, which will be the basis for the implementation, including the interface with the surrounding system, error signals, and interfaces with memory and queues and so on and so forth.

The block diagrams of the individual entities are provided and described, as well as the whole system.

2.1 Basic architecture

The solution to be implemented is of course only a small part of a huge design forming D9054 Encoder. In theory the FEC code just receives the TS packets and does the calculations and operations on these packets.

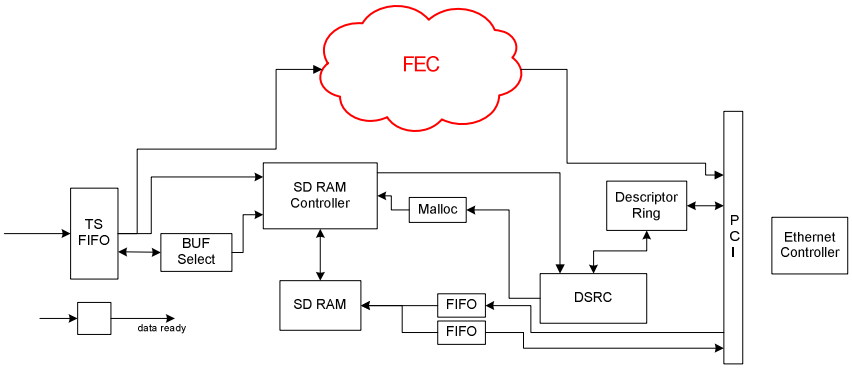


Figure 2.1 - Surrounding system

Data to the FEC entity is received from a first in, first out queue (FIFO), TS FIFO in the surrounding hardware and is put in an input FIFO inside the entity upon arrival. The FEC system operates in one clock domain, which is described further in the chapter concerning the implementation.

Data processed and generated in the FEC mechanism is fetched by an Ethernet controller via the PCI bus when the data is needed and not necessarily when data is ready. Therefore data has to be stored, since there is no guarantee that it is fetched before the next packet needs to be built. The obvious choice would be a FIFO queue since this eliminates the need for addressing; the first data stored in it would be the first data to be fetched on the other side. The Ethernet controller fetching the data would then keep tack of the number of words fetched from the FIFO but this would also be the case with other memory structures.

Despite the obvious and simplest choice of a FIFO queue, the memory storing the data produced in the FEC entity is an addressable memory based only upon the fact that it has to interface with the existing system, and this requires for addressable memory, more specifically dual port ram. Of course the design has to fit in with the existing system.

The FEC entity handles all calculations and operations concerning the generation of FEC packets and delivers them to a bus to be transmitted. Since the analysis resulted in the choice of a one dimensional XOR implementation, there is a need for some storage of the partially generated FEC packets.

To form the FEC packets, only two registers are needed, one to hold the payload which is used to calculate the FEC packet, and one for the actual FEC packet.

2.2 Clock domains

The solution operates in only one clock domain which has a frequency of 74.25 MHz and 74.25 MHz/1.001 according to which clock is used. There are other clock domains in the surrounding hardware but the FEC implementation does not need any signals belonging to these other domains.

The encoder supports four frame rates, each with a corresponding line count which dictates the clock domain for the processing in the encoder. These four formats are:

Table 4 - The relationship between the frame rate, frame format, and the clock frequency

Format[7]	E*	F*	M*	3**
Lines per frame	1125	1125	750	750
Words per line	2200	2640	1650	1980
Frame rate (Hz)	30/1.001	25	60/1.001	50
Resulting clock (MHz)	74.25/1.001	74.25	74.25/1.001	74.25

The source data is presented at the given frame rate and since the encoder does not change the frame rate, the resulting output must be presented at the same rate.

When either of the four frame modes is chosen in the user interface, the internal clock is set to 74.25 MHz or 74.25 MHz/1.001 accordingly.

Since the fastest clock is the 74.25 MHz clock, this will be used for calculations throughout the implementation. If a slower clock is used, it should have no effect on the design.

[7] * Formats are described in appendix 3
** Format is described in appendix 3

2.3 Entities

The FEC solution consists of a number of entities which will be described in the following sections.

2.3.1 IN-FIFO

This FIFO handles the reception of the TS packets from the surrounding system. The entity will have the following characteristics:

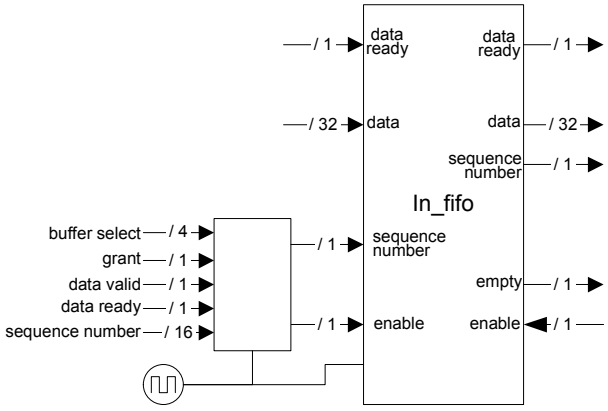


Figure 2.2 - In FIFO

data
Since it is a FIFO, there is a data bus going through the entity or the equivalent. Data is stored in the FIFO until the processing entity is ready at which point it signals to the FIFO and data is transferred. This data bus is 32 bits wide which is the same width as the surrounding system uses and is therefore the data width used in the FEC mechanism too.

Data is received from the TS FIFO in the existing system. The first word of data is ready for more than one clock cycle since the first byte of the word contains the id of the buffer to which is should be added. The reason for using the first byte is that the first byte of every TS packet is the same, 0x47. The buffer id is overwritten by this value after it has been read. The reason the first word of a TS packet is ready for more than one clock cycle is that when the first word is initially ready, the buffer id has to be decoded and fed to the

buffer select entity, which produces a little delay. This process takes more than one cycle and the second word can not be transmitted before the buffer id has been determined.

The delay is shown in the figure below:

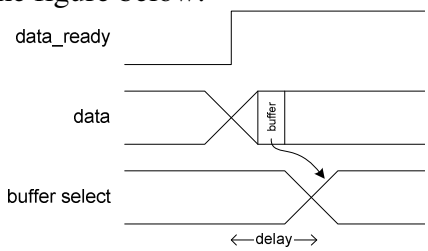


Figure 2.3 - Buffer select delay

enable
Besides the data bus, there is an enable signal to signal to the FIFO when data is ready on the bus. Data is fetched from the output of another FIFO, the TS FIFO and there are some conditions when to fetch data since not all data is intended for the FEC mechanism. Therefore the enable signal is a combination of a number of other signals; data ready, buffer select, data valid and grant. These signals together, result in a signal, which tells the in FIFO to fetch data from the data bus.

The data ready signal tells when data is ready in the TS FIFO but is not enough to act as an enable signal to the FIFO, since it is not all TS packets which are intended for the FEC. The buffer select signal determines the channel for which the data is intended. This is the signal that determines if the TS packet should also be fetched for the input FIFO in the FEC. When data is ready for the specific channel on which the FEC is applied, a copy of the data is fetched by the FIFO.

The grant signal is set by the SDRAM which buffers the TS packets while the IP packets are created. When grant is high, it tells the FIFO that the buffer select is updated as well as all other management signals. Then the only signal missing is the data valid. This signal is also set by the SDRAM and toggles between high and low depending on the status of the SDRAM. If the memory needs to change bank, the data valid is low, since the memory needs time for pointer household.

The enable signal received from the right side of the system, within the FEC mechanism, tells the FIFO when to deliver data to the outgoing data bus.

empty
The empty signal is used for signalling to the rest of the system when the FIFO is empty. The signal is active low which means that when the signal is 1, the FIFO is not empty and vice versa. When the FIFO is empty no enable signalling can be done since no packets can be transmitted.

The enable signalling is done based on this signal and the last word counter, described later. The reason for not basing the enable only on the empty signal is that there is a possibility that data arrives in bursts from the TS FIFO and into the in FIFO. This happens if the surrounding system does not fetch data from the TS FIFO at the exact time they are ready, then data stacks up in the FIFO and when fetched, it comes in bursts. This also sets some demands for the FIFO since it has to be able to handle the bursts without overflowing.

The size of the FIFO is what corresponds to one block ram in the FPGA, and this is the same size as the TS FIFO which is why there should be no problems with overflowing the FIFO in case of bursts. The FEC system should though process the IP packets faster than they arrive. Only the TS packets arrive in bursts, so the system has to be able to handle IP-"interpacket"-bursts.

data ready
There is one data ready signal going directly through the FIFO which is intended for a counter entity. The signal is passed through the FIFO in a 33rd bit of the data bus, the most significant bit. The signal is used to synchronize the last word counter, described in the section concerning the XOR entity. When the signal changes from zero to one, it marks the beginning of a new packet. The signal is held for some time, and only the low to high transition will be monitored.

sequence number
The signal will be used according to RFC2733 and the standard proposed by Pro-MPEG and indicates the first packet belonging to a FEC packet. The sequence number is retrieved from the SD ram controller when a new FEC packet is about to be build. Since the controller might not know the sequence number at the exact time the first TS packet of a new FEC packet arrives at the processing entity, and because the FIFO can hold several TS packets at a time, the sequence number might not be the valid one.

To cope with this problem the sequence number is sampled when data is going into the FIFO with the data ready signal as a reference to the beginning of each packet. After the data ready signal has changed from zero to one, the sequence number can be sampled after receiving a few words.

The sequence number can have up to 24 bits according to Pro-MPEG, but adding 24 bits to the data signal would mean that two block rams would be needed instead of one, since the limit for the width of the FIFO is 36 bits. To limit the use of block ram the sequence number is transmitted through the FIFO one bit at the time, together with the original data and the data ready signal. This would mean a 34th bit in the data bus. Also only 16 bit sequence numbers will be used, since the implemented FEC is very limited. If larger FEC solutions are implemented, the sequence number can easily be extended to 24 bit.

The first bit of the sequence number would be placed with the x-th word and then one bit with the next 24 words.

2.3.2 Processing

This entity is the primary one, which handles the data processing. The entity received the TS packets from the in FIFO and performs the XOR operation. Since data arrives 32 bits at the time, this is also the amount of data handled at the time.

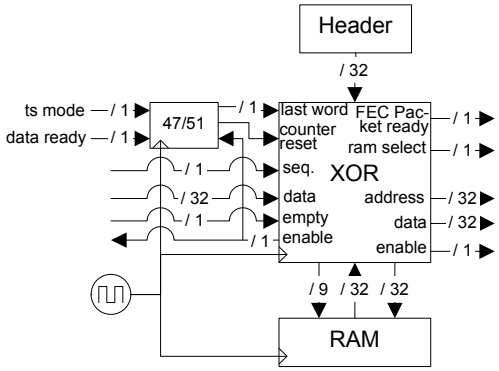


Figure 2.4 - Processing entity

XOR
The XOR entity is the general processing unit of the FEC mechanism. The entity receives data from the in FIFO when ready and builds a FEC packet from these.

When data is ready in the FIFO, empty is set to zero and data can be fetched. This is done by signalling enable to the FIFO for one clock period and fetching data from the bus. As described in the previous section, all data transmissions have a width of 32 bits.

The state diagram for the fetching of data is as follows:

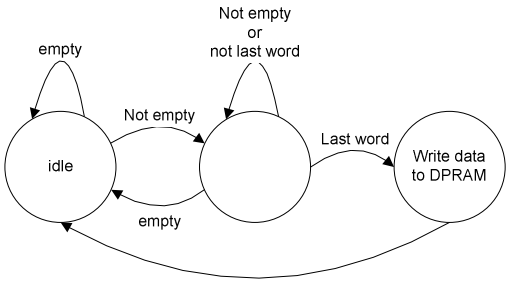


Figure 2.5 - States for fetching data from the in-FIFO

The FEC packet is build in the memory module and also has a width of 32 bits which means that addressing is done with nine bits. The way the packet is build is by fetching 32 bits at a time and performing an XOR operation with these bits and the received 32 bits. The result is stored at the same address in the memory.

When the last 32 bits of the last TS packet is received (signalled by the counter by raising the last word signal) and processed, the resulting FEC packet is transmitted to a memory containing the finished FEC packet including the IP header.

The memory select signal is used by the Ethernet controller in the surrounding system to address the correct data in the memory containing the finished FEC packet.

Header
This entity is a simple block that contains an IP header. The header is updated by the PCI bus with the information set in the user interface. This is properties such as: destination address and destination port, also the more indirect parameters are set by the user, such as the FEC setting, resulting in a FEC matrix, the individual parameters in the FEC matrix should not be available to the user, but depend on the FEC setting chosen. When the header is needed for a complete FEC packet, it is simply fetched and forwarded along with the FEC

payload. The sequence number is received in the processing unit as a signal and is inserted into the header when needed.

The header used for the FEC is the one specified by Pro-MPEG which is an extension to the one, proposed by the RFC, both described in previous sections.

Ram
This module is a memory module which is 32 bits wide, since all data busses are 32 bit, and can therefore be addressed by 9 bit addresses. The depth of the module is minimum one IP payload consisting of 7 TS packets which can have a size of either 51 or 47 32 bit words. To cover the largest payload of seven TS packets of 51 32 bit words, the memory must be at least 51 x 7 = 357, 32 bit words and is only limited by the hardware on which it is implemented. The excess memory not used can not be used elsewhere which is why there is no upper limit to the size.

Counter
The counter is a simple block which is used by the XOR entity to keep track of when the last packet for the FEC is received. This packet marks the time for finishing the FEC packet and transferring it to the memory containing the finished FEC packets.

The TS mode signal determines the number of bytes in a TS packet, 188 or 204, and this is used for counting the number of 32 bit words fetched by the XOR entity. Each FEC packet is the length of seven TS packets and is generated over ten TS packets. When 70 TS packets have been received and processed, one FEC packet is ready and the timeslot between the TS packets can be used to forward the packet.

This timeslot is determined by the surrounding system where the TS FIFO mentioned earlier gathers a complete TS packet before signalling that it is ready. This timeslot, where the TS packet is gathered in the TS FIFO is the timeslot available to the FEC entity.

The FIFO is limited to 10 TS packets determined by the size of the block ram Since data arrives one byte at the time in the TS FIFO at a frequency of 15 MHz. it takes either 188 or 204 clock cycles, depending on the size of the TS packets, at this frequency to collect one TS packet in the TS FIFO. The minimum time is 188 clock cycles which corresponds to

$188 \cdot 74.25/15 = 930.6 \approx 930$ [8] clock cycles when transforming it into the 74.25 MHz. clock domain; this again corresponds to 19 clock cycles to process each word. Since this is the smallest amount of time, the design has to meet this constraint.

If each word can be processed faster than the 19 clock cycles, no data overflow will occur in the in FIFO.

delay
The constraints for the sending of the FEC packets according to Pro-MPEG are that a FEC packets should be sent at a minimum of L packets after the last media packet, and a maximum of L·D packets after the last media packet protected. This means that there has to be introduces some delay mechanism in the system to ensure that these constraints are maintained.

The processing of the FEC packets from the TS FIFO until they are ready is presumably much faster than the processing of the TS packets in the rest of the system, so the delay has to be adjusted to the system. If no delay was introduced, the FEC packets would be sent much before the TS packets used to generate them. This is not preferable in any decoder since it collides with the proposed format for the FEC.

The delay is also managed by the processing entity by comparing sequence numbers.

[8] $\frac{1}{15MHz} * 188cc = 12.5\bar{3}m\sec \qquad \frac{1}{74.25MHz} = 13.47\mu\sec/cc$

$\Rightarrow \frac{12.5\bar{3}m\sec}{13.47\mu\sec/cc} = 930.6cc \qquad cc = clock\ cycles$

2.3.3 Transmitting

This entity is the last of the entities the FEC packet will pass through. It consists of a dual port ram in which two packets can be stored.

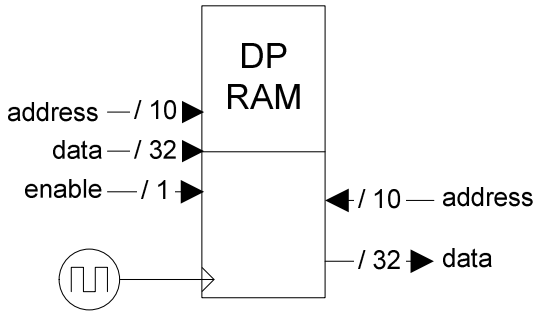


Figure 2.6 - Dual port RAM

The memory is divided into two parts, one to hold the packets which are transferred to the memory and one to hold the previous packet. The reason for the division of the memory is that the processing entity has no knowledge about when the surrounding system will fetch the packet from the memory. If the packet has not been fetched when the new packet is ready to be transferred into the memory the packet will be overwritten and the data lost. This means that the timing for the DP RAM is hard, data has to be fetched or they will be overwritten.

The mechanism handling the extraction from the memory is an Ethernet controller which receives a pointer to the right address space and the length of the data from a descriptor ring controller. Then data is transmitted over the PCI bus and forwarded in the system.

The reason for using a dual port ram instead of the more obvious choice of a FIFO is that the interface to the PCI bus is given by the existing system and is not optional. This means that the right side, the one facing the PCI bus, of any entity is determined already, and only the left side, facing the FEC system, is optional. The result of this is a dual port RAM which satisfies the needs from the existing system.

When generating the memory it is important to pay close attention to the interface to the PCI bus.

The timing of the ram is as follows:

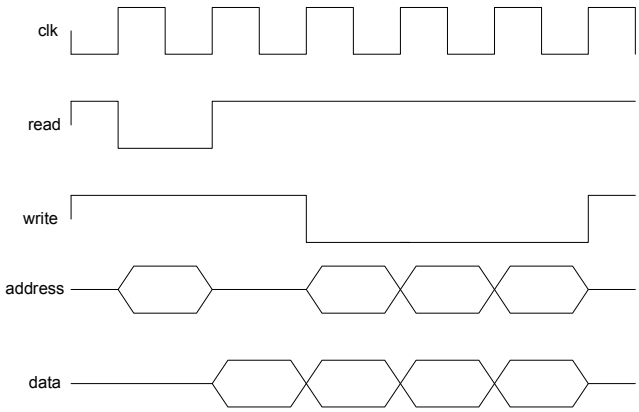


Figure 2.7 - Ram timing

In the timing diagram above, no considerations has been taken as to whether the enable signals are active high or low. There are some constraints to the timing of the memory which is that in order to make the timing shown above work, the clock signal, shared by all the entities, must reach the memory and the XOR entity at the same time. The XOR entity is the one setting the address and if the address is changed before the memory reads it, data is written at the wrong location. As the diagram shows, all signals for the write operation is set simultaneously, and can be changed at each rising edge of the clock, only if the timing constraint for the clock is maintained.

2.3.4 Supervising

There is one last entity which is a small supervising entity between the signalling from the FEC entity to the descriptor ring controller. The reason for having this block is for signalling a buffer overflow in the dual port ram.

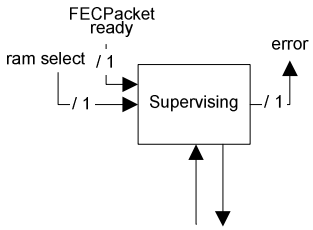


Figure 2.8 – Delay and supervising

When the FECPacket ready signal is high, it means that a FEC packet is ready to be transmitted, and the signal is passed to the descriptor ring controller, which then adds the appropriate pointer, depending on the memory select signal, to the descriptor ring, which the Ethernet controller services.

The error signal can be used to keep management of the used memory. If there is a situation where the packets are not fetched from the memory, the delay block would know it and send an output memory overflow-signal. The two unnamed signals are the signalling to and response from the descriptor ring controller. The signals are used to tell the descriptor ring that a FEC packet is ready in the memory and then the delay entity received some kind of acknowledge from the controller when the data has been read and new data can be written.

This means that the block acts as a supervisor of the dual port ram where the finished packets are stored. In case of a buffer overflow, the block sends an error signal.

2.3.5 Complete system

The entities described in the previous sections are all combined and integrated in the surrounding system. The signals used by the FEC system is the buffer select signal, data ready signal, and of course a data bus. The FEC system results in a data bus from the memory, containing the finished FEC packets, and a DP RAM select signal. The pointers to the DP RAM are hard coded in the descriptor ring controller and the DP RAM selector just defines which of the two memory spaces holds the finished packet. The complete system in which the FEC system is integrated:

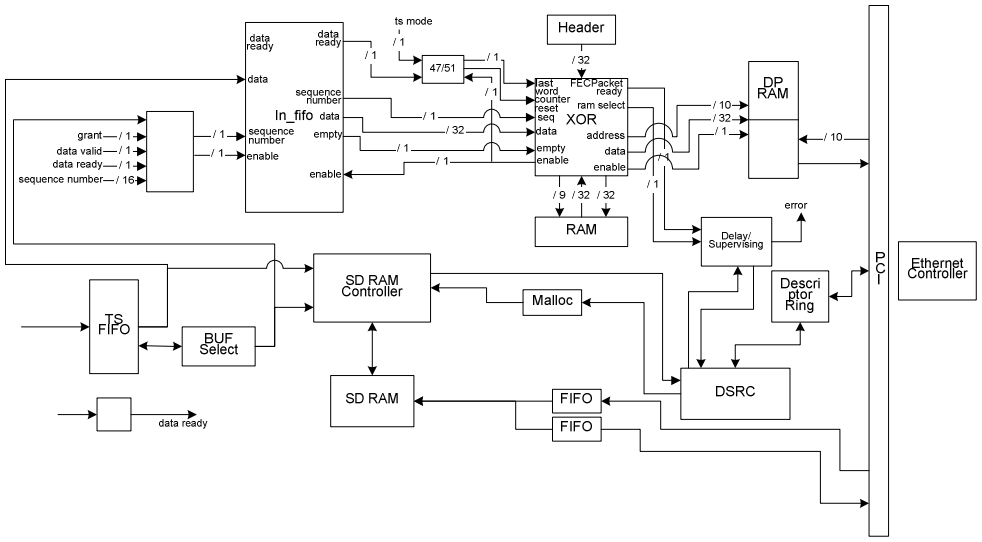


Figure 2.9 - Complete block diagram of the FEC system

The surrounding system consists of a TS FIFO described in the previous sections. The FIFO received data in blocks of eight bits at the time and these are gathered in the FIFO to build up a whole TS packet. As soon as the packet is in the FIFO, the data ready signal is set and data is collected from the FIFO by means of a 32 bit bus. The buffer select determines which buffer in the SD ram the current TS packet is destined for.

When the buffer select points to the buffer on which the FEC should be applied, the in FIFO of the FEC system collects the data while it is transferred onwards in the system. The data of each TS packet is put in an appropriate buffer in the SD ram module where IP packets are built. The SD ram controller handles the header insertion and the destinations of the TS packets. Also in the SD ram, other maintenance packets are build, for example responds to ARP requests.

These data for these packets is received in the two FIFOs which acts as receive and transmitting FIFOs respectively.

Furthermore there is a descriptor ring controller (DSRC) which keeps track of the pointers to the memory using a ring buffer. These pointers are used by the Ethernet controller for transmitting data over the PCI bus.

The malloc module in the middle is used to keep control of the memory used in the SD ram. The malloc module signals to the memory controller where to place the buffers for the IP packets.

Depending on the need, there is a control register placed outside the FPGA connected to a port. This means that signals can be routed to a multiplexer choosing between all debug signals from the other entities in the system. The entities debug signals are chosen by specifying an entity to the multiplexer by means of the PCI bus.

CHAPTER 3

Implementation

This section describes the individual entities in the system. Also the test bench used to verify the behaviour of the code. All source code is included in appendix 5.

3.1 Xilinx Core Generator

Many entities in the existing design has been generated by use of the Xilinx Core Generator, which is a tool for generating the most common VHDL modules such as memory, adders, multipliers and multiplexers.

Some of the entities of this project are generated by use of this tool too, since it would be a waste of time to implement it from scratch. Furthermore the Core Generator can optimize the modules more than would be immediately possible.

The Core Generator provides datasheets for all modules, the relevant sheets is included in appendix 6.

3.2 The complete system

The FEC system is routed together in one entity, a top entity, which only depicts the input and output pint to the complete system. The other signals are internal to the system. The top entity has the following inputs and outputs, where the pins in the top are the inputs and the ones in the bottom are the outputs:

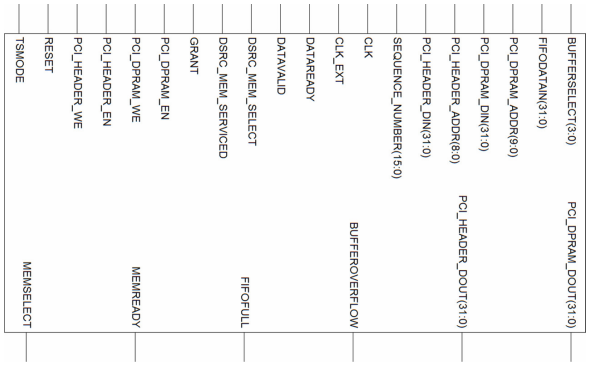


Figure 3.1 – Complete system entity

There are two clocks as inputs to the system, which is because the timing of the PCI bus is different from the timing of the FEC system and the surrounding system.

3.2.1 Input FIFO

This FIFO was generated by the Core Generator with the appropriate input and output signals. The signals for the FIFO were defined in the design chapter and the generation of the FIFO was straight forward. The figure below shows the generated FIFO with the input signals, output signals and flags.

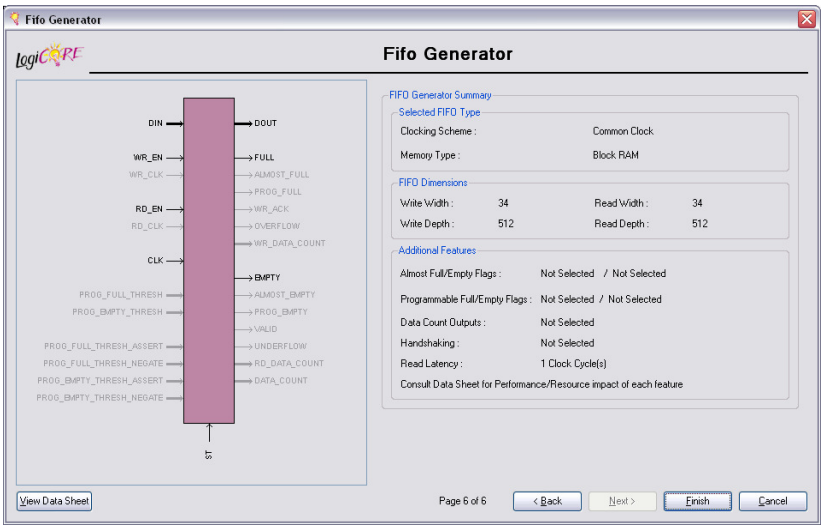


Figure 3.2 – Core generated input FIFO

The complete description of the FIFO is included as an appendix.

What is not described in the appendix is the levels of the FIFO flags; full and empty. The full flag is active high, which means that the flag signals '1' when the FIFO is full. The empty flag is active low, which means that the FIFO is empty when the signal is '1' and empty when it is '0'. This means that data can only be read from the FIFO when the empty flag is '1' but on the other hand, data can only be written to the FIFO when the full flag is '0'.

Also; after resetting the FIFO, it is important to wait for a few clock cycles because the flags can not be trusted to reflect the status of the FIFO. This is not described in the datasheet, but was experienced in simulations during the implementation.

3.2.1.1 FIFO enable

This entity has changed during the implementation since some signal behaved differently from first anticipated. The correct behaviour was not known before the implementation had commenced.

The inputs and outputs of the entity are shown below, where the signals on the left side are input signals and signals on the right are output signals.

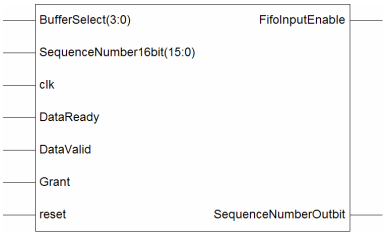


Figure 3.3 – FIFOEnable entity

The FIFO enable signal is generated from several other signals to ensure the FIFO is enabled only when data is ready on the data bus, and is disabled when data is no longer valid.

The entity is basically a small state machine which samples the input signals. The state diagram is shown below, and describes how the state transitions happens.

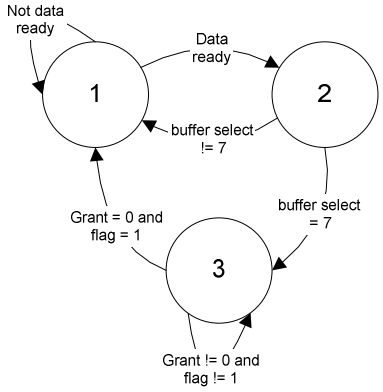


Figure 3.4 – FIFOEnable state diagram

When in state one, the state machine waits for the data ready signal to make a transition from low to high. When this happens it means that data is ready in

the TS FIFO, but the first word is held for more than one clock cycle to let the surrounding system handle buffer allocation, channel decoding and other kinds of housekeeping. When the surrounding system is ready, data ready is set, and the state machine changes to state two. In this state, the buffer select is sampled to make sure the data ready on the data bus is actually for the FEC system. This is the case when the buffer select is "0111" corresponding to channel seven.

If the channel is the right one, the state machine changes to state three, otherwise it returns to state one, where it waits for another transition in the data ready signal. In state three the machine waits for the grant signal which is the signal telling the rest of the system to start fetching data from the TS FIFO. If grant is high, it stays in state three, and if is low when reaching state three it also stays. When the signal has transitioned once, a flag is set to indicate this. After this, if the signal is low, the flag is dropped, and the state machine returns to state one.

Furthermore the entity adds the sequence number to the data bus. The signal is received as a 16 bit signal and can only be passed through the FIFO as a one bit signal. This is done by sampling the sequence number after a few words, to make sure the sequence number belongs to the current TS packet. Then the bits are added one by one to the data bus.

3.3 Processing

This entity is implemented as a state machine, as described in the design chapter. The state machine described there consisted of only four states, which was in no way enough, since many write and read operations take more than one clock cycle and therefore needs more than one state.

The resulting state diagram is shown below:

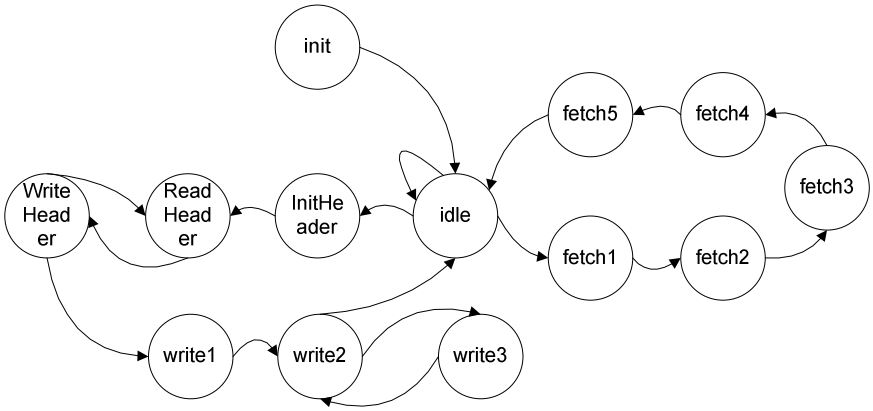


Figure 3.5 – FEC processing state diagram

All decisions as to which state or process should happen next are taken in the idle state.

The state machine initiates with an init state in which some variables are reset. From the init state the idle state is chosen. In this state it is decided in this state. If the FIFO contains data, and a whole IP packet has not been constructed, more TS packets should be fetched, and the state transitions to fetch1. Then a whole series of events happen, which include the fetching of data from the FIFO and from the storage memory, calculation of the FEC, and writing the result into the memory again.

If the FIFO is empty and the FEC still needs more TS packets before it is complete, the state machine stays in the idle state until the FIFO can deliver data.

When a FEC packet is complete, the headers are written into the dual port memory from which the packet is fetched by the PCI bus. This is also initiated from the idle state but runs until completion before continuing to the states where the actual FEC packet is transferred from the storage memory to the dual port memory.

The reason for having so many states for each part of the process is because some functions take more than one clock cycle. For example the header reading and writing consists of three states. The first state initiates the header storage with the appropriate signals for reading an address. After this the operation only transitions between two states where is alternates between reading the header from the header memory and writing it to the dual port ram.

Also the bits in the sequence number received from the FIFO as an extra bit in the data bus, has to be fetched and calculated to form the 16 bit sequence number.

This is done by finding the right word in the TS packet, and then simply bit by bit placing bit in the vector. The placement of the first bit in the sequence number is determined by the entity transforming the 16 bit sequence number to a stream of single bits, the entity enabling the input FIFO for reading data, and of course the two entities has to be synchronized, or the sequence number will be faulty.

```
if WordCount >= 5 and WordCount <= 20 then
  SequenceNumber(WordCount-5) <= SequenceNumber1Bit;
end if;
```

Figure 3.6 – Sequence number calculation

When a FEC packet is complete, the processing entity has to wait for some time before sending the FEC packet. The delay is defined by the standard, and has to be at least for the transmission of one ordinary IP packet. The way of calculating the delay is by saving the sequence number of the first IP packet used by the FEC. Then all following sequence numbers are compared to this, and when the difference is 11, it means that the ten packets for the FEC has been sent and then one more, and it is time to send the FEC packet.

The signalling is done by raising the FECPacketReady signal for one clock period. Then the supervising entity handles the rest of the process.

The entity has the following input and output pins, where input pins are on the top of the entity, and the output pins are in the bottom of the entity:

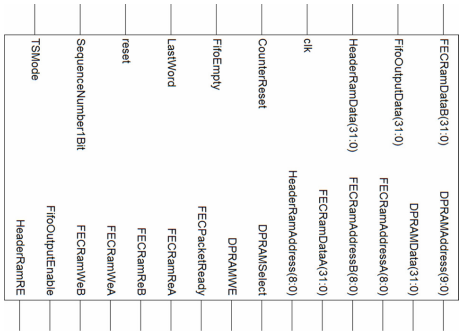


Figure 3.7 – FEC processing entity

3.3.1 Counter

This simple entity counts the number of TS packets fetched from the input FIFO. The reason for counting them is to keep track of the number of packets received in the processing entity. The entity is shown below, with the input pins and output pins.

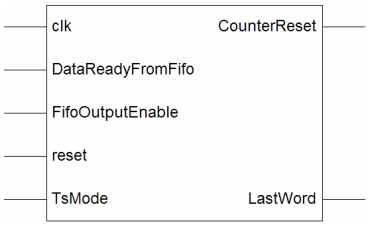


Figure 3.8 – Counter entity

The entity counts the packets triggered on the enable signal from the processing entity. When it enables the input FIFO then another word has been received for processing. Depending on the mode, 51 or 47 words in each TS packet, the signal LastWord is raised, and the processing entity knows that another TS packet has been received and processed.

If the counter encounters a DataReady signal which is one of the bits in the data bus from the input FIFO, the CounterReset signal is raised, since it means that the system is out of synchronization. If it happens, the processing entity should reset itself and drop the current FEC packet since it may by erroneous.

3.3.2 Header storage

This memory module was generated with the core generator, and looks very similar to the memory containing the finished IP packets. The values for the header are loaded via the PCI bus and therefore the memory module should have an interface fitting the PCI interface. The entity is shown below.

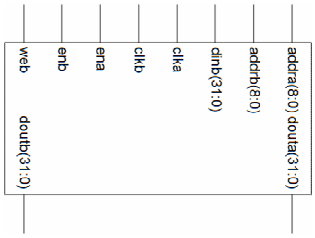


Figure 3.9 – Header storage entity

The entity is pretty simple, and can be written to and read from by use of the PCI bus. The fields of the headers are written to the memory to reflect the FEC solution chosen in the user interface of the encoder.

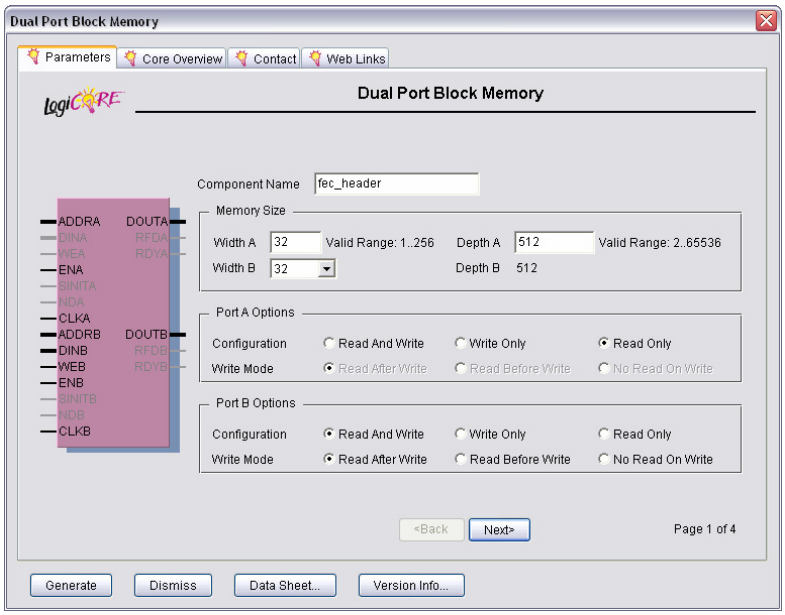


Figure 3.10 – Core generated RAM for header storage with PCI interface

Data can of course also be read by the processing entity when a header is added to a complete FEC packet. The B side of the memory is used by the PCI bus, whereas the A side is used by the FEC system.

3.3.3 FEC storage

As the FEC packet is created from the received TS packets, the results are stored in a memory module, FEC storage. This memory module is only used by the processing unit, and is of course used for both reading and writing.

The module is generated as a dual port memory, where writing is done on the A side, and reading is done from the B side of the memory.

It is generated, like the other memory modules, by use of the core generator, and the entity is shown below:

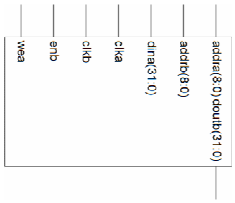


Figure 3.11 – FEC storage entity

The module was created with the following settings:

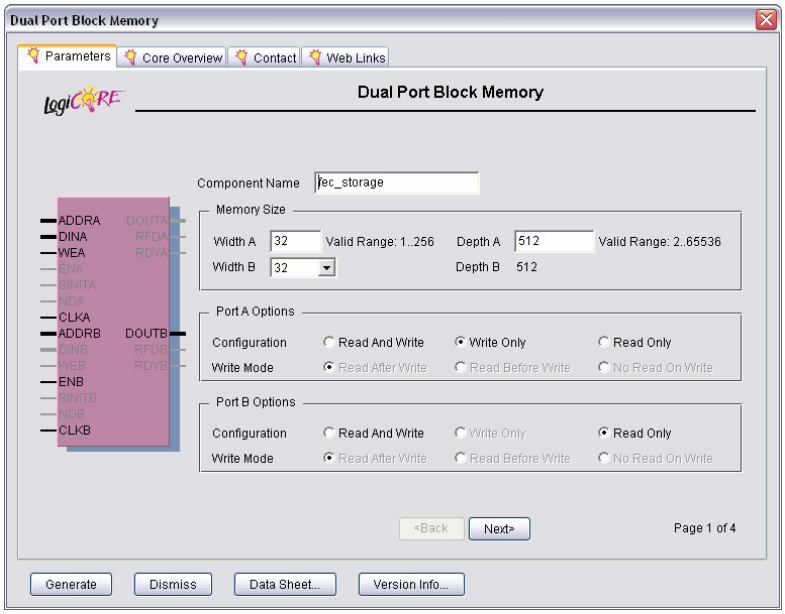


Figure 3.12 – Core generated RAM for FEC storage

3.4 Transmitting

The transmitting entity is basically a dual port RAM as described in the design chapter.

The interface to the entity on one side faces the PCI bus, and has to fit this existing interface. This is done by generating the memory module based on another module already existing in the design. Then there is no doubts about the interface, and will fit right in with the existing system.

The figure below shows the settings for the PCI side of the memory, which should not be changed. The other side could be fitted to the FEC system with the signals needed.

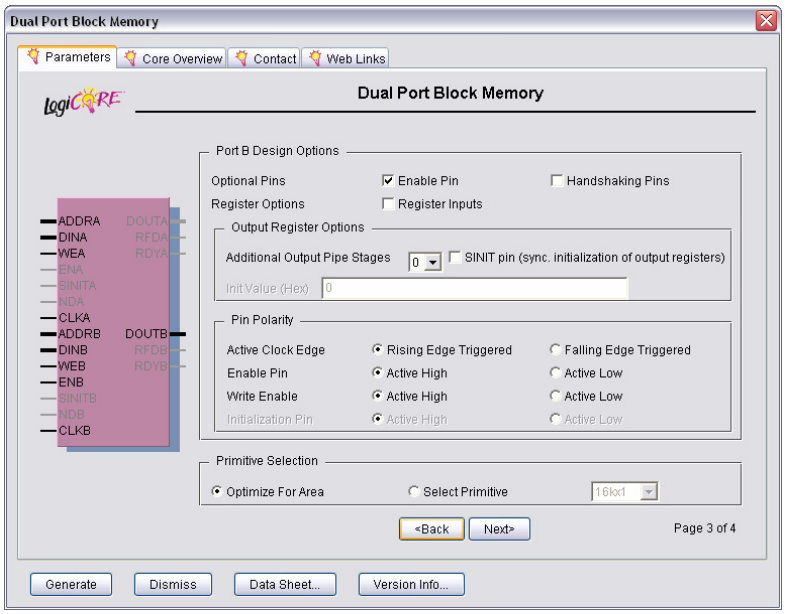


Figure 3.13 – Core generated RAM for FEC output with PCI interface

The memory block uses one block ram, which is more than needed, but it does not make sense to generate a memory block which is smaller than one block ram. Besides, there are more than enough block rams available on the FPGA as the system is implemented now.

The FEC system only needs to write to the memory, there will not be a situation where the ability to read data from the memory is needed. If the situation should occur, it is easy to reconfigure the dual port ram the reflect this.

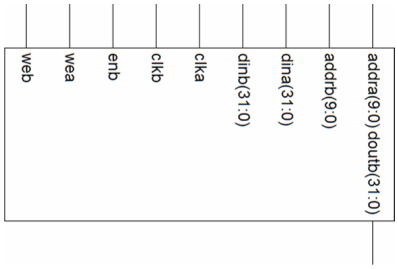


Figure 3.14 – Output RAM entity

3.5 Supervising

This entity handles the supervising of the dual port memory used for transmitting the finished FEC packets to the PCI bus.

The supervising is done by managing two signals representing the two buffers or banks in the DPRAM. When the FECPacketReady signal transitions from low to high, the DPRAMSelect signal is sampled, and the appropriate buffer is updated. If the Select signal is 0, bank one is updated, and if it is one, then bank two is updated.

The update consists of a single bit which is set to one. If the bit is already one, it means that the bank already contains a complete FEC packet, and has not been serviced by the Ethernet Controller. In this case the BufferOverflow signal is set to one, to signal to the rest of the system that an overflow has occurred. As for the FEC system, it continues, even if it means that data is lost in the memory.

The banks are cleared by the signals from the descriptor ring controller DSRCMemServices and DSRCMemSelect in the same way the banks are set.

The signals for setting the banks are passed directly through the entity, to the descriptor ring controller.

The entity is shown below, with input and output signals.

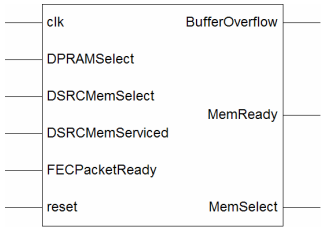


Figure 3.15 – Supervising entity

3.6 Test bench

A test bench was created to aid in the error correcting of the system. The test bench is not very extensive and does not cover all the possible incidents the FEC system can encounter, and can therefore not be used for testing the system.

Time did not permit testing the system, and this will be done subsequent to handing in the report.

3.7 Reset

When verifying the implemented entities with ModelSim, some peculiar behaviour was noticed. One of these behaviours were the effect of the reset signal which is a system reset, meaning that it is the same reset for all entities in the system.

When the reset signal was toggled all signals is reset to some initial value, and the system should be operable when the reset signal is zero again, but this is not the case.

The signals from the input FIFO, namely the full flag can not be trusted until three clock cycles after the reset. It simply signals '1' for three cycles, which means that the FIFO is full, but no data has been filled into it.

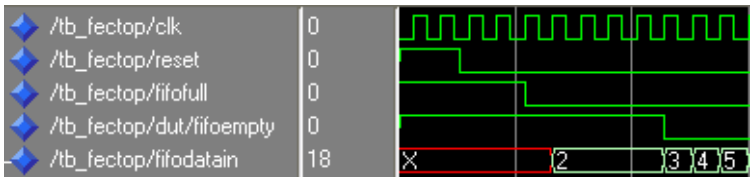


Figure 3.16 – Wave form of input FIFO reset

The wave form above shows the behaviour of the full and empty flags of the input FIFO. As described, the full flag, fifofull, is one for three clock cycles after the reset reaches zero, which means that if data is written into the FIFO, or attempted to be written will be lost.

The reason for the X'es on the fifodatain bus is because nothing is written on the bus until the FIFO full flag is zero. A part of the code from the test bench is shown below. The process waits for the reset to reach zero, then waits for three clock cycles and then data is put on the busses and signals.

```
FifoSide : process
begin
  wait until RESET = '0';

  wait for 3*PERIOD;
  for h in 34952 to 100000 loop
    for i in 0 to 6 loop
            .
            .
            .
    end loop;
  end loop;
end process FifoSide;
```

Figure 3.17 – Test bench with reference to the timing of the input FIFO

The reason for the late update of the flag is unknown, but very valuable to know, to avoid lost data. Nothing is mentioned about this in the datasheet for the FIFO, included in appendix 6.

It is therefore recommended that an idle state or initialization state is implemented on the top level, to ensure the correct behaviour of all the entities and their flags and signals. The state would have to run for at least 5-10 clock cycles, before commencing the processing in the system.

This recommendation for the system is actually only to ensure that the surrounding system is functioning correct. The FEC system is not effected, since there will always be a delay of at least 930 clock cycles from when the reset is zero to the arrival of data to the input FIFO. This delay is introduced because a whole TS packet has to be collected in the TS FIFO before it can be transmitted to the FEC system. This takes at least 930 clock cycles, in which time all the flags and signals of the entities should be stabile.

3.8 Detecting errors

As described in section 1.8.1, the network which the analysis is based upon, drops a whole package if a bit error is discovered within it. Because of this there is no need to have the ability to correct bit errors in packages since they never occur. But in the case of a network which does not drop the erroneous packages, it is crucial to know which package contains an error, since this is the packet to be corrected.

If this should be made possible, it would be necessary to use the check sum field in the UDP protocol with which the packages are sent. The check sum field enables the receiver to calculate the checksum of the received package and compare it to the checksum received. If there is inconsistency between the two, then the package is erroneous.

As the encoder is programmed today, the field is simply set to zero which indicates that the field is not used.

The solution implemented requires some kind of counter to let the receiver know if a package has been dropped, in which case two packages arriving one after the other would be missing a package in between. There is no continuity counter in the UDP protocol but there is one in the RTP protocol, and this protocol must be used in order to correct lost packets. The RTP header is not used in the encoder at the time, but in order to use the FEC option, it will be necessary.

CHAPTER 4

Conclusion

The project was initiated with an extensive analysis into the needs and possibilities for implementing a forward error correction scheme in Scientific Atlanta's video encoders.

The project has resulted in the implementation of a one dimensional Forward Error Correction solution, which is based on the RFC 2733 but fulfils the criteria described by Pro-MPEG. The latter has great importance for the future of encoders since the market is looking for interoperable encoders and decoders which work with other manufacturers products. Since no one else has come with any suggestions to an error correcting scheme for MPEG encoded streams, the Code of Practice created by Pro-MPEG has become a buzzword for the customers, who has no knowledge as to the actual need for error correction in their applications.

The problem with the scheme is that it is far too extensive and corrects more errors in a transmission than will probably ever be a reality, since the market is heading towards fiber networks for distributing media content. The fiber networks have error rates which are much smaller than ordinary copper lines, and therefore the Code of Practice is overdone as for the ability to correct errors, but since no one else has any suggestions, this is what the market currently demands.

The positive angle to the Pro-MPEG suggestion is that it is easy for all manufacturers of encoders and decoders to follow the same scheme and therefore be interoperable with others. The reason for implementing a small version of the Pro-MPEG scheme in Scientific Atlanta encoders is to win market shares on the fact that the buzzword scheme is supported.

The downside of the proposed scheme is that it introduces a comparatively long delay in the decoder which is intolerable in some applications.

The initial worry with the FEC scheme proposed by Pro-MPEG was the memory use, but after completing the analysis, it became clear that the memory use of the algorithm is smaller than anticipated, and will not cause a noticeable increase in the cost of an encoder or decoder.

As mentioned, the future of media streaming in IP networks is in fibers. The networks are faster and more reliable. The network is based on the ATM protocol in which some error correction is already implemented and the resulting errors are very few, and will only include errors that should occur on the cable connecting the encoder with the ATM box.

The proposed scheme from Pro-MPEG is too extensive, and should be adjusted to the actual needs of the industry. Since the forum is a collection of representatives from companies in the industry, it would be wise for Scientific Atlanta to join the forum or at least make their opinions clear as for the future of the FEC scheme which is still a suggestion but might become a standard.

For Scientific Atlanta, an implementation of an error correcting scheme would require some adjustments in the existing design. The obvious is of course the actual FEC system which will interact with the existing system, but also some changes are crucial in the existing design.

It is required that the RTP protocol is appended to the output, to enable the use of sequence numbers in order to identify the lost packets in a stream. Furthermore the use of the checksum field in the UDP header is required in order to detect bit errors in the transmitted signal.

As for the memory use and logic use in the FPGA, a simple FEC system would not be any problem to implement. The implemented system is a very limited solution, but some larger solutions would fit into the existing design.

CHAPTER 5

Other suggestions

The main point of this project has been to discuss the need for error protection on Scientific Atlanta's encoders, in particular the D9054 HD h.264 Encoder. The conclusion has been that for the networks considered for the actual applications, very weak or no FEC is needed since the error rate in the networks inflicted has little or no significance.

If the media stream were intended for other types of networks for example internet connections, the FEC would be very useful and the suggested algorithms might not even be sufficient for the correction of errors.

To handle this, an additional following FEC scheme is suggested; a three dimensional FEC matrix.

The FEC would consist of n, l·m matrices forming a cube. In this description l, m, and n is set to three, so the matrix is 3·3·3 matrix with a total of 27 original packets.

Each of the three matrices would have an ordinary row and column FEC calculated, as described in the analysis chapter. Then an additional 3rd dimensional FEC is applied, which calculates the FEC packets over the first packet in the three matrices, the second in al three and so on and so forth over al nine packets in three matrices. This results in 9 additional FEC packets, which add up to 27 FEC packets including the 18 packets from the 2-dimensional matrices.

The overhead would then be 50 % since 27 original packets are sent, and in addition to these, 27 FEC packets are sent. But the gain of this scheme is an ability to correct up to 70.37 % of the original packets if no FEC packets are lost in transmission. In given example of a 3·3 matrix, 19 of the 27 packets can be recreated.

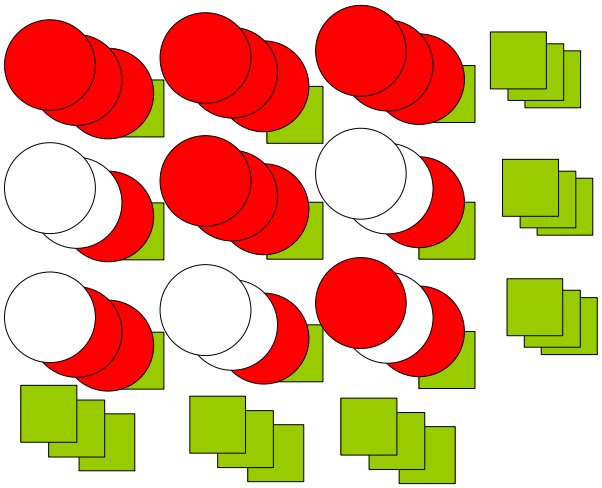


Figure 5.1 - Three dimensional FEC

The overhead in the transmission is quite big, but is greatly compensated by the ability to correct errors. The proposed FEC is a very strong solution with strength in both the correction of burst errors and random errors.

Literature

[1] The MPEG-4 Book
F. Pereira, T. Ebrahimi
2002, Prentice Hall, Pearson Education
ISBN: 0-13-061621-4

[2] Broadcasters guide to MPEG
O. S. Nielsen, N. Eriksen
1999, BARCO A/S.

[3] An RTP Payload Format for Generic Forward Error Correction
J. Rosenberg, H. Schulzrinne
Network Working Group
1999, Request for Comments: 2733

[4] MPEG-II Video Coding For Noisy Channels - A dissertation submitted to the University of Cambridge for the degree of Doctor of Philosophy
Robert Swann, Trinity College, March 11, 1998
Signal Processing and Communications Laboratory.
Department of Engineering.

[5] Transport of MPEG-4 over IP/RTP
A. Basso, S. Varakliotis
0-7803-6536-4/00 2000 IEEE, p. 1067-1070

[6] Error Control Coding – An Introduction
P. Sweeney
1991, Prentice Hall
ISBN: 0-13-284118-5

[7] The Theory and Practice of Error Control Codes
R. E. Blahut
1983, Addison Wesley
ISBN: 0-201-10102-5

[8] Transmission of Professional MPEG-2 Transport Streams over IP Networks
Pro-MPEG Code of Practice #3 release 2 July 2004

Scientific Atlanta
A CISCO COMPANY

FEC on IP-output for video encoder

Appendices

Maria Baltzer Pedersen

Technical University of Denmark
Kongens Lyngby 2006

IMM-B.ENG-2006-60

Contents

1 Appendix

1.1 Output from Tandberg Encoder

1.1.1 No FEC is added, and the transmission protocol is RTP

290306_NOFEC_RTP_black.pcap

```
00000000: D4C3B2A1 02000400 00000000 00000000 ÔÃ²¡••••••••••••
00000010: FFFF0000 01000000 EA0D0000 CA730600 ÿÿ••••••ê•••Ês••
00000020: 5E050000 5E050000 0008D48A 03F700E0 ^•••^•••••Ô•÷•à
00000030: 2A010E5F 08004500 054C1DFB 40000711 *••_••E••L•û@•••
00000040: CD52C0A8 0102C0A8 01017FBC 157D0538 ÍRÀ¨••À¨•••¼•}•8
00000050: 00008021 ECF212CB 629D7CC6 EC6D471F •••!ìò•Ëb•|ÆìmG
00000060: FF100000 00000000 00000000 00000000 ÿ•••••••••••••••
00000070: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000080: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000090: 00000000 00000000 00000000 00000000 ••••••••••••••••
000000A0: 00000000 00000000 00000000 00000000 ••••••••••••••••
000000B0: 00000000 00000000 00000000 00000000 ••••••••••••••••
000000C0: 00000000 00000000 00000000 00000000 ••••••••••••••••
000000D0: 00000000 00000000 00000000 00000000 ••••••••••••••••
000000E0: 00000000 00000000 00000000 00000000 ••••••••••••••••
000000F0: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000100: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000110: 00000000 00000000 0000470C 4414FFFF ••••••••••G•D•ÿÿ
00000120: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000130: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000140: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000150: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000160: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000170: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000180: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000190: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000001A0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000001B0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000001C0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000001D0: FFFFFFFF FFFF471F FF100000 00000000 ÿÿÿÿÿÿG•ÿ••••••••
000001E0: 00000000 00000000 00000000 00000000 ••••••••••••••••
000001F0: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000200: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000210: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000220: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000230: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000240: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000250: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000260: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000270: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000280: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000290: 0000470C 4415FFFF FFFFFFFF FFFFFFFF ••G•D•ÿÿÿÿÿÿÿÿ
000002A0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000002B0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000002C0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000002D0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000002E0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000002F0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000300: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000310: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000320: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000330: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000340: FFFFFFFF FFFFFFFF FFFFFFFF FFFF471F ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿG
00000350: FF100000 00000000 00000000 00000000 ÿ•••••••••••••••
00000360: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000370: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000380: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000390: 00000000 00000000 00000000 00000000 ••••••••••••••••
000003A0: 00000000 00000000 00000000 00000000 ••••••••••••••••
000003B0: 00000000 00000000 00000000 00000000 ••••••••••••••••
000003C0: 00000000 00000000 00000000 00000000 ••••••••••••••••
000003D0: 00000000 00000000 00000000 00000000 ••••••••••••••••
000003E0: 00000000 00000000 00000000 00000000 ••••••••••••••••
000003F0: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000400: 00000000 00000000 0000471F FF100000 ••••••••••G•ÿ•••
00000410: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000420: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000430: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000440: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000450: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000460: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000470: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000480: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000490: 00000000 00000000 00000000 00000000 ••••••••••••••••
```

1 of 14

290306_NOFEC_RTP_black.pcap

```
000004A0: 00000000 00000000 00000000 00000000 ................
000004B0: 00000000 00000000 00000000 00000000 ................
000004C0: 00000000 0000470C 4416FFFF FFFFFFFF ......G.D.ÿÿÿÿÿÿÿÿ
000004D0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000004E0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000004F0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000500: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000510: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000520: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000530: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000540: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000550: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000560: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000570: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000580: FFFF629A 3934EA0D 0000A37A 06005E05 ÿÿb.94ê....£z..^.
00000590: 00005E05 00000008 D48A03F7 00E02A01 ..^.....Ô.÷.à*.
000005A0: 0B5F0800 4500054C 1DFC4000 0711CD51 ._..E..L.ü@...ÍQ
000005B0: C0A80102 C0A80101 7FBC157D 05380000 À¨..À¨...¼.}.8..
000005C0: 8021ECF3 12CB633B 7CC6EC6D 471FFF10 .!ìó.Ëc;|ÆimG ÿ.
000005D0: 00000000 00000000 00000000 00000000 ................
000005E0: 00000000 00000000 00000000 00000000 ................
000005F0: 00000000 00000000 00000000 00000000 ................
00000600: 00000000 00000000 00000000 00000000 ................
00000610: 00000000 00000000 00000000 00000000 ................
00000620: 00000000 00000000 00000000 00000000 ................
00000630: 00000000 00000000 00000000 00000000 ................
00000640: 00000000 00000000 00000000 00000000 ................
00000650: 00000000 00000000 00000000 00000000 ................
00000660: 00000000 00000000 00000000 00000000 ................
00000670: 00000000 00000000 00000000 00000000 ................
00000680: 00000000 00000000 470C4437 07102020 ........G.D7..
00000690: 86E0587E 3DFFFFFF FFFFFFFF FFFFFFFF †àX~=ÿÿÿÿÿÿÿÿÿÿÿÿ
000006A0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000006B0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000006C0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000006D0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000006E0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000006F0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000700: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000710: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000720: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000730: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000740: FFFFFFFF FF471FFF 10000000 00000000 ÿÿÿÿÿG ÿ........
00000750: 00000000 00000000 00000000 00000000 ................
00000760: 00000000 00000000 00000000 00000000 ................
00000770: 00000000 00000000 00000000 00000000 ................
00000780: 00000000 00000000 00000000 00000000 ................
00000790: 00000000 00000000 00000000 00000000 ................
000007A0: 00000000 00000000 00000000 00000000 ................
000007B0: 00000000 00000000 00000000 00000000 ................
000007C0: 00000000 00000000 00000000 00000000 ................
000007D0: 00000000 00000000 00000000 00000000 ................
000007E0: 00000000 00000000 00000000 00000000 ................
000007F0: 00000000 00000000 00000000 00000000 ................
00000800: 00471FFF 10000000 00000000 00000000 .G ÿ............
00000810: 00000000 00000000 00000000 00000000 ................
00000820: 00000000 00000000 00000000 00000000 ................
00000830: 00000000 00000000 00000000 00000000 ................
00000840: 00000000 00000000 00000000 00000000 ................
00000850: 00000000 00000000 00000000 00000000 ................
00000860: 00000000 00000000 00000000 00000000 ................
00000870: 00000000 00000000 00000000 00000000 ................
00000880: 00000000 00000000 00000000 00000000 ................
00000890: 00000000 00000000 00000000 00000000 ................
000008A0: 00000000 00000000 00000000 00000000 ................
000008B0: 00000000 00000000 00000000 00471FFF ............G ÿ
000008C0: 10000000 00000000 00000000 00000000 ................
000008D0: 00000000 00000000 00000000 00000000 ................
000008E0: 00000000 00000000 00000000 00000000 ................
000008F0: 00000000 00000000 00000000 00000000 ................
00000900: 00000000 00000000 00000000 00000000 ................
00000910: 00000000 00000000 00000000 00000000 ................
00000920: 00000000 00000000 00000000 00000000 ................
00000930: 00000000 00000000 00000000 00000000 ................
```

1.1.2 No FEC is added, and the transmission protocol is UDP

290306_NOFEC_UDP_black.pcap

```
00000000: D4C3B2A1 02000400 00000000 00000000 ÔÃ²¡••••••••••••
00000010: FFFF0000 01000000 2F0E0000 5E5D0E00 ÿÿ••••••/•••^]••
00000020: 52050000 52050000 0008D48A 03F700E0 R•••R•••••ÔŠ•÷•à
00000030: 2A010B5F 08004500 0540B8BF 40000711 *••_••E••@¸¿@•••
00000040: 329AC0A8 0102C0A8 01017FBC 157D052C 2šÀ¨••À¨•••¼•}•,
00000050: 0000470C 441AFFFF FFFFFFFF FFFFFFFF ••G•D•ÿÿÿÿÿÿÿÿ
00000060: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000070: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000080: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000090: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000000A0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000000B0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000000C0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000000D0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000000E0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000000F0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000100: FFFFFFFF FFFFFFFF FFFFFFFF FFFF471F ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿG
00000110: FF100000 00000000 00000000 00000000 ÿ•••••••••••••••
00000120: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000130: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000140: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000150: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000160: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000170: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000180: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000190: 00000000 00000000 00000000 00000000 ••••••••••••••••
000001A0: 00000000 00000000 00000000 00000000 ••••••••••••••••
000001B0: 00000000 00000000 00000000 00000000 ••••••••••••••••
000001C0: 00000000 00000000 0000470C 441AFFFF ••••••••••G•D•ÿÿ
000001D0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000001E0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000001F0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000200: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000210: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000220: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000230: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000240: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000250: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000260: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000270: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000280: FFFFFFFF FFFF471F FF100000 00000000 ÿÿÿÿÿÿÿG ÿ•••••••
00000290: 00000000 00000000 00000000 00000000 ••••••••••••••••
000002A0: 00000000 00000000 00000000 00000000 ••••••••••••••••
000002B0: 00000000 00000000 00000000 00000000 ••••••••••••••••
000002C0: 00000000 00000000 00000000 00000000 ••••••••••••••••
000002D0: 00000000 00000000 00000000 00000000 ••••••••••••••••
000002E0: 00000000 00000000 00000000 00000000 ••••••••••••••••
000002F0: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000300: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000310: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000320: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000330: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000340: 0000470C 441CFFFF FFFFFFFF FFFFFFFF ••G•D•ÿÿÿÿÿÿÿÿ
00000350: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000360: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000370: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000380: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000390: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000003A0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000003B0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000003C0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000003D0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000003E0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000003F0: FFFFFFFF FFFFFFFF FFFFFFFF FFFF471F ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿG
00000400: FF100000 00000000 00000000 00000000 ÿ•••••••••••••••
00000410: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000420: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000430: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000440: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000450: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000460: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000470: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000480: 00000000 00000000 00000000 00000000 ••••••••••••••••
00000490: 00000000 00000000 00000000 00000000 ••••••••••••••••
```

1 of 4

290306_NOFEC_UDP_black.pcap

```
000004A0: 00000000 00000000 00000000 00000000 ................
000004B0: 00000000 00000000 0000470C 441DFFFF ..........G.D.ÿÿ
000004C0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000004D0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000004E0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000004F0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000500: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000510: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000520: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000530: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000540: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000550: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000560: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000570: FFFFFFFF FFFF9DF2 EC502F0E 00003864 ÿÿÿÿÿÿ.òìP/...8d
00000580: 0E005205 00005205 00000008 D48A03F7 ..R...R.....ÔŠ.÷
00000590: 00E02A01 0B5F0800 45000540 B8C04000 .à*..._..E..@¸À@.
000005A0: 07113299 C0A80102 C0A80101 7FBC157D ..2™À¨..À¨...¼.}
000005B0: 052C0000 471FFF10 00000000 00000000 .,..G ÿ.........
000005C0: 00000000 00000000 00000000 00000000 ................
000005D0: 00000000 00000000 00000000 00000000 ................
000005E0: 00000000 00000000 00000000 00000000 ................
000005F0: 00000000 00000000 00000000 00000000 ................
00000600: 00000000 00000000 00000000 00000000 ................
00000610: 00000000 00000000 00000000 00000000 ................
00000620: 00000000 00000000 00000000 00000000 ................
00000630: 00000000 00000000 00000000 00000000 ................
00000640: 00000000 00000000 00000000 00000000 ................
00000650: 00000000 00000000 00000000 00000000 ................
00000660: 00000000 00000000 00000000 00000000 ................
00000670: 470C441E FFFFFFFF FFFFFFFF FFFFFFFF G.D.ÿÿÿÿÿÿÿÿÿÿÿÿ
00000680: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000690: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000006A0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000006B0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000006C0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000006D0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000006E0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000006F0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000700: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000710: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000720: FFFFFFFF FFFFFFFF FFFFFFFF 471FFF10 ÿÿÿÿÿÿÿÿÿÿÿÿG ÿ.
00000730: 00000000 00000000 00000000 00000000 ................
00000740: 00000000 00000000 00000000 00000000 ................
00000750: 00000000 00000000 00000000 00000000 ................
00000760: 00000000 00000000 00000000 00000000 ................
00000770: 00000000 00000000 00000000 00000000 ................
00000780: 00000000 00000000 00000000 00000000 ................
00000790: 00000000 00000000 00000000 00000000 ................
000007A0: 00000000 00000000 00000000 00000000 ................
000007B0: 00000000 00000000 00000000 00000000 ................
000007C0: 00000000 00000000 00000000 00000000 ................
000007D0: 00000000 00000000 00000000 00000000 ................
000007E0: 00000000 00000000 471FFF10 00000000 ........G ÿ.....
000007F0: 00000000 00000000 00000000 00000000 ................
00000800: 00000000 00000000 00000000 00000000 ................
00000810: 00000000 00000000 00000000 00000000 ................
00000820: 00000000 00000000 00000000 00000000 ................
00000830: 00000000 00000000 00000000 00000000 ................
00000840: 00000000 00000000 00000000 00000000 ................
00000850: 00000000 00000000 00000000 00000000 ................
00000860: 00000000 00000000 00000000 00000000 ................
00000870: 00000000 00000000 00000000 00000000 ................
00000880: 00000000 00000000 00000000 00000000 ................
00000890: 00000000 00000000 00000000 00000000 ................
000008A0: 00000000 471FFF10 00000000 00000000 ....G ÿ.........
000008B0: 00000000 00000000 00000000 00000000 ................
000008C0: 00000000 00000000 00000000 00000000 ................
000008D0: 00000000 00000000 00000000 00000000 ................
000008E0: 00000000 00000000 00000000 00000000 ................
000008F0: 00000000 00000000 00000000 00000000 ................
00000900: 00000000 00000000 00000000 00000000 ................
00000910: 00000000 00000000 00000000 00000000 ................
00000920: 00000000 00000000 00000000 00000000 ................
00000930: 00000000 00000000 00000000 00000000 ................
```

1.1.3 FEC setting 1, and the transmission protocol is UDP

290306_FEC01_black.pcap

```
00000000: D4C3B2A1 02000400 00000000 00000000 ÔÃ²¡••••••••••••
00000010: FFFF0000 01000000 7F0D0000 B5340700 ÿÿ••••••••••µ4••
00000020: D2050000 D2050000 0008D48A 03F700E0 Ò•••Ò•••••ÔŠ•÷•à
00000030: 2A010B5F 08004500 05C03039 40000711 *••_••E••À09@•••
00000040: BAA0C0A8 0102C0A8 01017FBC 157D05AC ° À¨••À¨•••¼•}•¬
00000050: 00008064 FF301238 81FE7CC6 EC6DCC51 •••dÿ0•8•þ|ÆìmÌQ
00000060: 8801471F 44100000 FF00FF00 FF00FF00 ^•G D•••ÿ•ÿ•ÿ•ÿ•
00000070: 00FF00FF 00FF00FF 00FF00FF 00FF00FF •ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ
00000080: 000000FF 00FF00FF 00FF00FF 00FF00FF •••ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ
00000090: 00FF00FF 00000000 FF00FF00 FF00FF00 •ÿ•ÿ••••ÿ•ÿ•ÿ•ÿ•
000000A0: FF00FF00 FF00FF00 FF000000 FF000000 ÿ•ÿ•ÿ•ÿ•ÿ•••ÿ•••
000000B0: 00000000 FF00FF00 0000FF00 00FF00FF ••••ÿ•ÿ•••ÿ••ÿ•ÿ
000000C0: 00FF00FF 00FF0000 FF00FF00 0000FF00 •ÿ•ÿ•ÿ•••ÿ•ÿ•••ÿ•
000000D0: FF00FF00 FF0000FF 00FF00FF 00FF00FF ÿ•ÿ•ÿ••ÿ•ÿ•ÿ•ÿ•ÿ
000000E0: 00FF00FF 00FF0000 00FF00FF 00FF00FF •ÿ•ÿ•ÿ•••ÿ•ÿ•ÿ•ÿ
000000F0: 00FF00FF 00FF00FF 00FF0000 0000FF00 •ÿ•ÿ•ÿ•ÿ•ÿ••••ÿ•
00000100: FF00FF00 FF00FF00 FF00FF00 FF00FF00 ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•
00000110: 0000FF00 00000000 FF000300 FF007D2F ••ÿ•••••ÿ•••ÿ•}/
00000120: 98796FE3 0F722103 7FD41990 4A67470C ˜yoã•r!••Ô••JgG•
00000130: FF1C0000 00FF00FF 00FF00FF 0000FF00 ÿ••••ÿ•ÿ•ÿ•ÿ••ÿ•
00000140: FF00FF00 FF00FF00 FF00FF00 FF000000 ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•••
00000150: FF00FF00 FF00FF00 FF00FF00 FF00FF00 ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•
00000160: FF000000 00FF00FF 00FF00FF 00FF00FF ÿ••••ÿ•ÿ•ÿ•ÿ•ÿ•ÿ
00000170: 00FF00FF 00FF0000 00FF0000 00000000 •ÿ•ÿ•ÿ•••ÿ••••••
00000180: 00FF00FF 000000FF 0000FF00 FF00FF00 •ÿ•ÿ•••ÿ••ÿ•ÿ•ÿ•
00000190: FF00FF00 00FF00FF 000000FF 00FF00FF ÿ•ÿ••ÿ•ÿ•ÿ•••ÿ•ÿ•ÿ
000001A0: 00FF0000 FF00FF00 FF00FF00 FF00FF00 •ÿ•••ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•
000001B0: FF00FF00 0000FF00 FF00FF00 FF00FF00 ÿ•ÿ•••ÿ•ÿ•ÿ•ÿ•ÿ•
000001C0: FF00FF00 FF00FF00 000000FF 00FF00FF ÿ•ÿ•ÿ•ÿ•••ÿ•ÿ•ÿ
000001D0: 00FF00FF 00FF00FF 00FF00FF 000000FF •ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•••ÿ
000001E0: 00000000 00FF0000 00FF7D2F A6BD6F9A •••••ÿ•••Ÿ}/¦½oš
000001F0: D273EDDA F0DC6DDF B367471F 4410FF00 ÒsíÚðÜmß³gG D•ÿ•
00000200: 0000FF00 FF00FF00 FF0000FF 00FF00FF ••ÿ•ÿ•ÿ•ÿ••ÿ•ÿ•ÿ
00000210: 00FF00FF 00FF00FF 00FF0000 00FF00FF •ÿ•ÿ•ÿ•ÿ•ÿ•••ÿ•ÿ
00000220: 00FF00FF 00FF00FF 00FF00FF 00FF0000 •ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ••
00000230: 0000FF00 FF00FF00 FF00FF00 FF00FF00 ••ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•
00000240: FF00FF00 0000FF00 00000000 0100FF00 ÿ•ÿ•••ÿ•••••••ÿ•
00000250: FF000000 FF0000FF 00FF00FF 00FF00FF ÿ•••ÿ••ÿ•ÿ•ÿ•ÿ•ÿ
00000260: FF00FF00 FF000000 FF00FF00 FF00FF00 ÿ•ÿ•ÿ•••ÿ•ÿ•ÿ•ÿ•
00000270: 00FF00FF 00FF00FF 00FF00FF 00FF00FF •ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ
00000280: 000000FF 00FF00FF 00FF00FF 00FF00FF •••ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ
00000290: 00FF00FF 00000000 FF00FF00 FF00FF00 •ÿ•ÿ••••ÿ•ÿ•ÿ•ÿ•
000002A0: FF00FF00 FF00FF00 FF000000 FF000000 ÿ•ÿ•ÿ•ÿ•ÿ•••ÿ•••
000002B0: 0000FF00 0000F52F A679C49A 0F142161 ••ÿ•••õ/¦yÄš••!a
000002C0: 7FF51967 4AF1470C FF1D00FF 000000FF •õ•qJñG•ÿ••ÿ•••ÿ
000002D0: 00FF00FF 00FF0000 FF00FF00 FF00FF00 •ÿ•ÿ•ÿ•••ÿ•ÿ•ÿ•ÿ
000002E0: FF00FF00 FF00FF00 0000FF00 FF00FF00 ÿ•ÿ•ÿ•ÿ•••ÿ•ÿ•ÿ•
000002F0: FF00FF00 FF00FF00 FF00FF00 000000FF ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•••ÿ
00000300: 00FF00FF 00FF00FF 00FF00FF 00FF00FF •ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ
00000310: 000000FF 00000000 000C00FF 00FF0000 •••ÿ•••••••ÿ•ÿ••
00000320: 00FF0000 FF00FF00 FF00FF00 00FF00FF •ÿ••ÿ•ÿ•ÿ•ÿ••ÿ•ÿ
00000330: 00FF0000 00FF00FF 00FF00FF 0000FF00 •ÿ•••ÿ•ÿ•ÿ•ÿ••ÿ•
00000340: FF00FF00 FF00FF00 FF00FF00 FF000000 ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•••
00000350: FF00FF00 FF00FF00 FF00FF00 FF00FF00 ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•
00000360: FF000000 00FF00FF 00FF00FF 00FF00FF ÿ•••ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•
00000370: 00FF00FF 00FF0000 00FF0000 000000FF •ÿ•ÿ•ÿ•••ÿ•••••ÿ
00000380: 00037D9A A6796F01 0F73AEDA 16DCC2DF ••}š¦yo••s®Ú•ÜÂß
00000390: 7A67471F 4410FF00 FF000000 FF00FF00 zqG D•ÿ•ÿ•••ÿ•ÿ•
000003A0: FF00FF00 00FF00FF 00FF00FF 00FF00FF ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ
000003B0: 00FF00FF 000000FF 00FF00FF 00FF00FF •ÿ•ÿ•••ÿ•ÿ•ÿ•ÿ•ÿ
000003C0: 00FF00FF 00FF00FF 00000000 FF00FF00 •ÿ•ÿ•ÿ•ÿ••••ÿ•ÿ•
000003D0: FF00FF00 FF00FF00 FF00FF00 FF000000 ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•••
000003E0: FF000000 0000FF00 FF00FF00 0000FF00 ÿ•••••ÿ•ÿ•ÿ•••ÿ•
000003F0: 00FF00FF 00FF00FF FF00FF00 FF00FF00 •ÿ•ÿ•ÿ•ÿÿÿ•ÿ•ÿ•ÿ•
00000400: 0000FF00 FF00FF00 FF0000FF 00FF00FF ••ÿ•ÿ•ÿ•ÿ••ÿ•ÿ•ÿ
00000410: 00FF00FF 00FF00FF 00FF0000 00FF00FF •ÿ•ÿ•ÿ•ÿ•ÿ•••ÿ•ÿ
00000420: 00FF00FF 00FF00FF 00FF00FF 00FF0000 •ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ••
00000430: 0000FF00 FF00FF00 FF00FF00 FF00FF00 ••ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•
00000440: FF00FF00 0000FF00 00000000 FF00A52F ÿ•ÿ•••ÿ•••••ÿ•¥/
00000450: 57796F9A 38732165 7FEA1963 4A5E470C Wyoš8s!e•ê•cJ^G•
00000460: FF3E00FF 00FF0000 00FF00FF 00FF00FF ÿ>•ÿ•ÿ•••ÿ•ÿ•ÿ•ÿ
00000470: 0000FF00 FF00FF00 FF00FF00 FF00FF00 ••ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•
00000480: FF000000 FF00FF00 FF00FF00 FF00FF00 ÿ•••ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•
00000490: FF00FF00 FF000000 00FF00FF 00FF00FF ÿ•ÿ•ÿ••••ÿ•ÿ•ÿ•ÿ
```

290306_FEC01_black.pcap

```
000004A0: 00FF00FF 00FF00FF 00FF0000 00FF0000 •ÿ•ÿ•ÿ•ÿ•ÿ•••ÿ••
000004B0: 000000FF 00FF00FF 000000FF 0000FF00 •••ÿ•ÿ•ÿ•••ÿ••ÿ•
000004C0: FF00FF00 00FF00FF 00FF00FF 000000FF ÿ•ÿ••ÿ•ÿ•ÿ•ÿ•••ÿ
000004D0: 00FF00FF 00FF0000 FF00FF00 FF00FF00 •ÿ•ÿ•ÿ••ÿ•ÿ•ÿ•ÿ•
000004E0: FF00FF00 FF00FF00 0000FF00 FF00FF00 ÿ•ÿ•ÿ•ÿ•••ÿ•ÿ•ÿ•
000004F0: FF00FF00 FF00FF00 FF00FF00 000000FF ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•••ÿ
00000500: 00FF00FF 00FF00FF 00FF00FF 00FF00FF •ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ
00000510: 000000FF 00000000 00FF7DA1 A62B6F9A •••ÿ••••ÿ}¡¦+oš
00000520: 0F1E21DA FDDCA7DF 3C67471F 44100700 ••!ÚýÜ§ß<gG D•••
00000530: FF00FF00 0000FF00 FF00FF00 FF0000FF ÿ•ÿ•••ÿ•ÿ•ÿ•ÿ•ÿ
00000540: 00FF00FF 00FF00FF 00FF00FF 00FF0000 •ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ••
00000550: 00FF00FF 00FF00FF 00FF00FF 00FF00FF •ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ
00000560: 00FF0000 0000FF00 FF00FF00 FF00FF00 •ÿ••••ÿ•ÿ•ÿ•ÿ•ÿ•
00000570: FF00FF00 FF00FF00 0000FF00 00000000 ÿ•ÿ•ÿ•ÿ••••ÿ••••
00000580: FF00FF00 FF000000 FF0000FF 00FF00FF ÿ•ÿ•ÿ•••ÿ••ÿ•ÿ•ÿ
00000590: FF00FF00 FF00FF00 FF000000 FF00FF00 ÿ•ÿ•ÿ•ÿ•ÿ•••ÿ•ÿ•
000005A0: FF00FF00 00FF00FF 00FF00FF 00FF00FF ÿ•ÿ••ÿ•ÿ•ÿ•ÿ•ÿ•ÿ
000005B0: 00FF00FF 000000FF 00FF00FF 00FF00FF •ÿ•ÿ•••ÿ•ÿ•ÿ•ÿ•ÿ
000005C0: 00FF00FF 00FF00FF 00000000 FF00FF00 •ÿ•ÿ•ÿ•ÿ••••ÿ•ÿ•
000005D0: FF00FF00 FF00FF00 FF00FF00 FF000000 ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•••
000005E0: FF000000 0000E32F AF79D19A 0F7367DA ÿ••••ã/¯yÑš•sgÚ
000005F0: 7F4F1994 4AB2EA29 F3937F0D 0000903B •O•”J²ê)ó”•••••;
00000600: 0700D205 0000D205 00000008 D48A03F7 ••Ò•••Ò•••••ÔŠ•÷
00000610: 00E02A01 0B5F0800 450005C0 303A4000 •à*••_••E••À0:@•
00000620: 0711BA9F C0A80102 C0A80101 7FBC157D ••°ŸÀ¨••À¨•••¼•}
00000630: 05AC0000 8064FF31 1238829C 7CC6EC6D •¬•••dÿ1•8‚œ|Æìm
00000640: CC518F01 470CFF1F 001000FF 00FF0000 ÌQ••G•ÿ ••••ÿ••
00000650: 00FF00FF 00FF00FF 0000FF00 FF00FF00 •ÿ•ÿ•ÿ•ÿ••ÿ•ÿ•ÿ•
00000660: FF00FF00 FF00FF00 FF000000 FF00FF00 ÿ•ÿ•ÿ•ÿ•ÿ•••ÿ•ÿ•
00000670: FF00FF00 FF00FF00 FF00FF00 FF000000 ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•••
00000680: 00FF00FF 00FF00FF 00FF00FF 00FF00FF •ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ
00000690: 00FF0000 00FF0000 000000FF 00FF00FF •ÿ•••ÿ•••••ÿ•ÿ•ÿ
000006A0: 000000FF 0000FF00 FF0000FF 00FF00FF •••ÿ••ÿ•ÿ••ÿ•ÿ•ÿ
000006B0: 00FF00FF 000000FF 00FF00FF 00FF0000 •ÿ•ÿ•••ÿ•ÿ•ÿ•ÿ••
000006C0: FF00FF00 FF00FF00 FF00FF00 FF00FF00 ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•
000006D0: 0000FF00 FF00FF00 FF00FF00 FF00FF00 ••ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•
000006E0: FF00FF00 000000FF 00FF00FF 00FF00FF ÿ•ÿ••••ÿ•ÿ•ÿ•ÿ•ÿ
000006F0: 00FF00FF 00FF00FF 000000FF 00000000 •ÿ•ÿ•ÿ•ÿ•••ÿ••••
00000700: 7DECA6DA 6F3A0F73 21077FDC 8EDFF567 }ì¦Úo:•s!••ÜŽßÖg
00000710: 471F4410 FF002020 20202020 00FF00FF G D•ÿ•       •ÿ•ÿ
00000720: 000000FF 00FF00FF 00FF0000 FF00FF00 •••ÿ•ÿ•ÿ•ÿ••ÿ•ÿ•
00000730: FF00FF00 FF00FF00 FF00FF00 0000FF00 ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•••ÿ•
00000740: FF00FF00 FF00FF00 FF00FF00 FF00FF00 ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•
00000750: 000000FF 00FF00FF 00FF00FF 00FF00FF •••ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ
00000760: 00FF00FF 000000FF 00000000 00FF00FF •ÿ•ÿ•••ÿ••••ÿ•ÿ•
00000770: 00FF0000 00FF0000 FF00FFFF 00FF00FF •ÿ•••ÿ••ÿ•ÿÿ•ÿ•ÿ
00000780: 00FF00FF 00FF0000 00FF00FF 00FF00FF •ÿ•ÿ•ÿ•••ÿ•ÿ•ÿ•ÿ
00000790: 0000FF00 FF00FF00 FF00FF00 FF00FF00 ••ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•
000007A0: FF000000 FF00FF00 FF00FF00 FF00FF00 ÿ•••ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•
000007B0: FF00FF00 FF000000 00FF00FF 00FF00FF ÿ•ÿ•ÿ••••ÿ•ÿ•ÿ•ÿ
000007C0: 00FF00FF 00FF00FF 00FF0000 00FF0000 •ÿ•ÿ•ÿ•ÿ•ÿ•••ÿ••
000007D0: 007D2F31 79859A1B 7321DA1B DC19804A •}/1y…š•s!Ú•Ü••J
000007E0: 1D470CFF 1000FF00 3D00FF00 FF000000 •G•ÿ••ÿ•=•ÿ•ÿ•••
000007F0: FF00FF00 FF00FF00 00FF00FF 00FF00FF ÿ•ÿ•ÿ•ÿ••ÿ•ÿ•ÿ•ÿ
00000800: 00FF00FF 00FF00FF 000000FF 00FF00FF •ÿ•ÿ•ÿ•ÿ•••ÿ•ÿ•ÿ
00000810: 00FF00FF 00FF00FF 00FF00FF 00000000 •ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•••• 
00000820: FF00FF00 FF00FF00 FF00FF00 FF00FF00 ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•
00000830: FF000000 FF000000 0000FF00 FF00FF00 ÿ•••ÿ••••ÿ•ÿ•ÿ•
00000840: 0000FF00 00FF0000 FF00FF00 FF00FF00 ••ÿ•••ÿ••ÿ•ÿ•ÿ•ÿ•
00000850: FF00FF00 0000FF00 FF00FF00 FF0000FF ÿ•ÿ•••ÿ•ÿ•ÿ•ÿ••ÿ
00000860: 00FF00FF 00FF00FF 00FF00FF 00FF0000 •ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ••
00000870: 00FF00FF 00FF00FF 00FF00FF 00FF00FF •ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ
00000880: 00FF0000 0000FF00 FF00FF00 FF00FF00 •ÿ••••ÿ•ÿ•ÿ•ÿ•ÿ•
00000890: FF00FF00 FF00FF00 0000FF00 007D2FA6 ÿ•ÿ•ÿ•ÿ•••ÿ••}/¦
000008A0: CA6F840F 0021DA7F 5019DFB6 67471F44 Êo„••!Ú•P•ß¶gG D
000008B0: 10FF00FF 006F00FF 00FF0000 00FF00FF •ÿ•ÿ•o•ÿ•ÿ•••ÿ•ÿ
000008C0: 00FF00FF 0000FF00 FF00FF00 FF00FF00 •ÿ•ÿ••ÿ•ÿ•ÿ•ÿ•ÿ•
000008D0: FF00FF00 FF000000 FF00FF00 FF00FF00 ÿ•ÿ•ÿ•••ÿ•ÿ•ÿ•ÿ•
000008E0: FF00FF00 FF00FF00 FF000000 00FF00FF ÿ•ÿ•ÿ•ÿ•ÿ•••ÿ•ÿ
000008F0: 00FF00FF 00FF00FF 00FF00FF 00FF0000 •ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ••
00000900: 00FF0000 000000FF 00FF00FF 000000FF •ÿ•••••ÿ•ÿ•ÿ•••ÿ
00000910: 0000FFFF 00FF00FF 00FF00FF 00FF00FF ••ÿÿ•ÿ•ÿ•ÿ•ÿ•ÿ•ÿ
00000920: 000000FF 00FF00FF 00FF0000 FF00FF00 •••ÿ•ÿ•ÿ•ÿ••ÿ•ÿ•
00000930: FF00FF00 FF00FF00 FF00FF00 0000FF00 ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•••ÿ•
```

1.1.4 FEC setting 4, and the transmission protocol is UDP

290306_FEC04_black.pcap

```
00000000: D4C3B2A1 02000400 00000000 00000000 ÔÃ²¡••••••••••••
00000010: FFFF0000 01000000 AD0D0000 C8460A00 ÿÿ••••••-•••ÈF••
00000020: D2050000 D2050000 0008D48A 03F700E0 Ò•••Ò•••••ÔŠ•÷•à
00000030: 2A010B5F 08004500 05C096BB 40000711 *••_••E••À–»@•••
00000040: 541EC0A8 0102C0A8 01017FBC 157D05AC T•À¨••À¨•••¼•}•¬
00000050: 00008064 65B21277 F4F57CC6 EC6DCF1F •••de²•w´õ|ÆimÍ
00000060: 1604470C FF1D00FF 0000FF00 FF00FF00 ••G•ÿ••ÿ••ÿ•ÿ•ÿ•
00000070: FF00FF00 00FF00FF 00FF0000 00FF00FF ÿ•ÿ••ÿ•ÿ•ÿ•••ÿ•ÿ
00000080: 0000FF00 FF00FF00 FF00FF00 FFFF00FF ••ÿ•ÿ•ÿ•ÿ•ÿ•ÿÿ•ÿ
00000090: 00FF00FF 00FF00FF 00000000 FF00FF00 •ÿ•ÿ•ÿ•ÿ••••ÿ•ÿ•
000000A0: FF00FF00 FFFF00FF 00FF00FF 00FF0000 ÿ•ÿ•ÿÿ•ÿ•ÿ•ÿ•••
000000B0: 0000FF00 FF00FF00 FF00FFFF FF0000FF ••ÿ•ÿ•ÿ•ÿ•ÿÿÿ••ÿ
000000C0: 000000FF FFFF00FF 0000FF00 FF00FF00 •••ÿÿÿ•ÿ••ÿ•ÿ•ÿ•
000000D0: FF00FF00 FF00FFFF 00FF00FF 00FF00FF ÿ•ÿ•ÿ•ÿ•ÿÿ•ÿ•ÿ•ÿ
000000E0: 000000FF 0000FF00 0000FF00 FF0000FF •••ÿ••ÿ•••ÿ•ÿ••ÿ
000000F0: 000000FF 00FF00FF 00FF0000 FF00FF00 •••ÿ•ÿ•ÿ•ÿ••ÿ•ÿ•
00000100: FF00FF00 FF00FFFF 00FF00FF 00FF0000 ÿ•ÿ•ÿ•ÿÿ•ÿ•ÿ•ÿ••
00000110: 00FF0000 FF00FF00 0000FF00 FF004091 •ÿ••ÿ•ÿ•••ÿ•ÿ•@'
00000120: A6686F3A 0F7221A5 7F89197F 3A67471F ¦ho:•r!¥•‰••:gG
00000130: 4410FF00 FFFF00FF 00FF00FF 00FF00FF D•ÿ•ÿÿ•ÿ•ÿ•ÿ•ÿ•ÿ
00000140: 0000FF00 FF000000 0000FF00 FFFF0000 ••ÿ•ÿ••••ÿ•ÿÿ••
00000150: 00FF00FF 00FF00FF 0000FF00 FF00FF00 •ÿ•ÿ•ÿ•ÿ•••ÿ•ÿ•ÿ•
00000160: FF00FF00 FF0000FF 00FF00FF 00FF00FF ÿ•ÿ•ÿ••ÿ•ÿ•ÿ•ÿ•ÿ
00000170: 0000FF00 FF000000 FF00FF00 9A0000FF ••ÿ•ÿ•••ÿ•ÿ•š••ÿ
00000180: 00FF00FF 00FF00FF 0000FF00 FFFFFF00 •ÿ•ÿ•ÿ•ÿ••ÿ•ÿÿÿ•
00000190: FF00FF00 FF0000FF 00FF00FF 000000FF ÿ•ÿ•ÿ••ÿ•ÿ•ÿ•••ÿ
000001A0: 00FF0000 0000FF00 0000FF00 FF00FFFF •ÿ••••ÿ•••ÿ•ÿ•ÿÿÿ
000001B0: FFFF00FF 00FF00FF 00FF0000 FF00FF00 ÿÿÿ•ÿ•ÿ•ÿ•ÿ••ÿ•ÿ•
000001C0: FF00FF00 000000FF 00FF00FF 00FF00FF ÿ•ÿ•••ÿ•ÿ•ÿ•ÿ•ÿ
000001D0: 00FF0000 FF00FF00 FF00FF00 FF00FF00 •ÿ••ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•
000001E0: 00FF00FF 0000003E 00FF7D2F A679FB9A •ÿ•ÿ•••>••ÿ}/¦yûš
000001F0: 387324DA 21DCE4DF 4A4F471F FF1E00FF 8s$Ú!ÜäßJOG ÿ••ÿ
00000200: 0000FF00 0000FF00 FF000000 FFFF0000 ••ÿ•••ÿ•ÿ•••ÿÿ••
00000210: 00FF0000 00FF00FF 0000FF00 FF00FF00 •ÿ•••ÿ•ÿ••ÿ•ÿ•ÿ•
00000220: FF000000 FF0000FF 00FF0000 00FF00FF ÿ•••ÿ••ÿ•ÿ•••ÿ•ÿ
00000230: 00FF0000 FF00FF00 FF000000 FF0000FF •ÿ••ÿ•ÿ•ÿ•••ÿ••ÿ
00000240: 00FF00FF 00FF00FF 0000FF00 FF00FF00 •ÿ•ÿ•ÿ•ÿ••ÿ•ÿ•ÿ•
00000250: 0000FF00 FF0000FF 00FF00FF 00FF00FF ••ÿ•ÿ••ÿ•ÿ•ÿ•ÿ•ÿ
00000260: 00FF0000 FF00FF00 FF00FF00 FF00FF00 •ÿ••ÿ•ÿ•ÿ•ÿ•ÿ•ÿ•
00000270: 00FF00FF 00FF00FF 00FF0000 0000FF00 •ÿ•ÿ•ÿ•ÿ•ÿ•••ÿ•
00000280: FF00FF00 FF00FFFF 00FF00FF 00FF00FF ÿ•ÿ•ÿ•ÿÿ•ÿ•ÿ•ÿ•ÿ
00000290: 00FF0000 FF00FF00 0000FF00 FF00FF00 •ÿ••ÿ•ÿ•••ÿ•ÿ•ÿ•
000002A0: 000000FF 00FF00FF 00FF00FF FF00FF00 •••ÿ•ÿ•ÿ•ÿ•ÿÿÿ•ÿ•
000002B0: FF00FF00 FF007D2F A6FE6FD8 0F1421DA ÿ•ÿ•ÿ•}/¦þoØ••!Ú
000002C0: 7F96196B 4A67471F 4410FF00 FF0000FF •–•kJgG D•ÿ•ÿ••ÿ
000002D0: 00FF00FF 00FF00FF 0000FF00 FF00FF00 •ÿ•ÿ•ÿ•ÿ••ÿ•ÿ•ÿ•
000002E0: 0000FF00 FF0000FF 00FF00FF 00FF00FF ••ÿ•ÿ••ÿ•ÿ•ÿ•ÿ•ÿ
000002F0: 00FFFF00 FF00FF00 FF00FF00 FF000000 •ÿÿ•ÿ•ÿ•ÿ•ÿ•ÿ•••
00000300: 00FF00FF 00FF00FF 00FFFF00 FF00FF00 •ÿ•ÿ•ÿ•ÿ•ÿÿ•ÿ•ÿ•
00000310: FF00FF00 000000FF 00FF00FF 00FF00FF ÿ•ÿ••••ÿ•ÿ•ÿ•ÿ•ÿ
00000320: FFFF0000 FF000000 FFFFFF00 FF0000FF ÿÿ••ÿ•••ÿÿÿ•ÿ••ÿ
00000330: 00FF00FF 00FF00FF 00FF00FF FF00FF00 •ÿ•ÿ•ÿ•ÿ•ÿ•ÿÿ•ÿ•
00000340: FF00FF00 FF000000 FF0000FF 000000FF ÿ•ÿ•ÿ•••ÿ••ÿ•••ÿ
00000350: 00FF0000 FF000000 FF00FF00 FF00FF00 •ÿ••ÿ•ÿ•••ÿ•ÿ•ÿ•ÿ•
00000360: 00FF00FF 00FF00FF 00FF00FF FF00FF00 •ÿ•ÿ•ÿ•ÿ•ÿ•ÿÿ•ÿ•
00000370: FF00FF00 0000FF00 00FF00FF 000000FF ÿ•ÿ•••ÿ••ÿ•ÿ•••ÿ
00000380: 00FF7D8C 1B79039A 1B73EDDA C7DC4BDF •ÿ}Œ•y•š•síÚÇÜKß
00000390: F3A3470C FF1F00FF 00FFFF00 FF00FF00 ó£G•ÿ •ÿ•ÿÿ•ÿ•ÿ•
000003A0: FF00FF00 FF0000FF 00FF0000 000300FF ÿ•ÿ•ÿ••ÿ•ÿ•••••ÿ
000003B0: 00FFFF00 0000FF00 FF00FF00 FF0000FF •ÿÿ•••ÿ•ÿ•ÿ•ÿ••ÿ
000003C0: 00FF00FF 00FF00FF 00FF0000 FF00FF00 •ÿ•ÿ•ÿ•ÿ•ÿ••ÿ•ÿ•
000003D0: FF00FF00 FF0000FF 00FF0000 00FF00FF ÿ•ÿ•ÿ••ÿ•ÿ•••ÿ•ÿ
000003E0: 00000000 FF00FF00 FF00FF00 FF0000FF ••••ÿ•ÿ•ÿ•ÿ•ÿ••ÿ
000003F0: 00FFFFFF 00FF00FF 00FF0000 FF00FF00 •ÿÿÿ•ÿ•ÿ•ÿ••ÿ•ÿ•
00000400: FF000000 FF00FF00 000000FF 000000FF ÿ•••ÿ•ÿ••••ÿ•••ÿ
00000410: 00FF00FF FFFFFF00 FF00FF00 FF00FF00 •ÿ•ÿÿÿÿ•ÿ•ÿ•ÿ•ÿ•
00000420: 00FF00FF 00FF00FF 00000000 FF00FF00 •ÿ•ÿ•ÿ•ÿ••••ÿ•ÿ•
00000430: FF00FF00 FF00FF00 00FF00FF 00FF00FF ÿ•ÿ•ÿ•ÿ••ÿ•ÿ•ÿ•ÿ
00000440: 00FF00FF 0000FF00 FF000000 65008F2F •ÿ•ÿ••ÿ•ÿ•••e••/
00000450: A6796F4C 0F1E2103 7FCB199C 4A67471F ¦yoL••!••Ë•œJgG
00000460: FF10FF00 FF0000FF 000000FF 00FF0000 ÿ•ÿ•ÿ••ÿ•••ÿ•ÿ••
00000470: 00FFFF00 0000FF00 0000FF00 FF0000FF •ÿÿ•••ÿ•••ÿ•ÿ•ÿ•ÿ
00000480: 00FF00FF 00FF0000 00FF0000 FF00FF00 •ÿ•ÿ•ÿ•ÿ•••ÿ••ÿ•ÿ•
00000490: 0000FF00 FF00FF00 00FF00FF 00FF0000 ••ÿ•ÿ•ÿ•••ÿ•ÿ•ÿ••
```

1 of 8

290306_FEC04_black.pcap

```
000004A0: 00FF0000 FF00FF00 FF00FF00 FF0000FF ·ÿ··ÿ·ÿ·ÿ·ÿ·ÿ··ÿ
000004B0: 00FF00FF 000000FF 00FF0000 FF00FF00 ·ÿ·ÿ···ÿ·ÿ··ÿ·ÿ·
000004C0: FF00FF00 FF00FF00 00FF00FF 00FF00FF ÿ·ÿ·ÿ·ÿ··ÿ·ÿ·ÿ·ÿ
000004D0: 00FF00FF 0000FF00 FF00FF00 FF00FF00 ·ÿ·ÿ··ÿ·ÿ·ÿ·ÿ·ÿ·
000004E0: 000000FF 00FF00FF 00FF00FF FF00FF00 ···ÿ·ÿ·ÿ·ÿ·ÿÿ·ÿ·
000004F0: FF00FF00 FF00FF00 00FF00FF 000000FF ÿ·ÿ·ÿ·ÿ···ÿ·ÿ···ÿ
00000500: 00FF00FF 00000000 FF00FF00 FF00FF00 ·ÿ·ÿ····ÿ·ÿ·ÿ·ÿ·
00000510: FFFF00FF 00FF00FF 00FF7D2F A679169A ÿÿ·ÿ·ÿ·ÿ·ÿ}/¦y·š
00000520: F173AEDA 7FDC2EDF B067470C FF1000FF ñs®Ú·Ü.ß°gG·ÿ··ÿ
00000530: 00FF0000 FF00FF00 FF00FF00 FF0000FF ·ÿ··ÿ·ÿ·ÿ·ÿ·ÿ··ÿ
00000540: 00FF00FF 000000FF 00FF0000 FF00FF00 ·ÿ·ÿ···ÿ·ÿ··ÿ·ÿ·
00000550: FF00FF00 FF00FFFF 00FF00FF 00FF00FF ÿ·ÿ·ÿ·ÿÿ·ÿ·ÿ·ÿ·ÿ
00000560: 00FF0000 0000FF00 FF00FF00 FF00FFFF ·ÿ····ÿ·ÿ·ÿ·ÿ·ÿÿ
00000570: 00FF00FF 00FF00FF 00000000 FF00FF00 ·ÿ·ÿ·ÿ·ÿ····ÿ·ÿ·
00000580: FF00FF00 FFFFFF00 00FF0000 00FFFFFF ÿ·ÿ·ÿÿÿ··ÿ···ÿÿÿ
00000590: 00FF0000 FF00FF00 FF00FF00 FF00FF00 ·ÿ··ÿ·ÿ·ÿ·ÿ·ÿ·ÿ·
000005A0: FFFF00FF 00FF00FF 00FF0000 00FF0000 ÿÿ·ÿ·ÿ·ÿ·ÿ···ÿ··
000005B0: FF000000 FF00FF00 00FF0000 00FF00FF ÿ···ÿ·ÿ··ÿ···ÿ·ÿ
000005C0: 00FF00FF 0000FF00 FF00FF00 FF00FF00 ·ÿ·ÿ··ÿ·ÿ·ÿ·ÿ·ÿ·
000005D0: FFFF00FF 00FF00FF 000000FF 0000FF00 ÿÿ·ÿ·ÿ·ÿ·ÿ···ÿ··ÿ·
000005E0: FF000000 FF00C32F CF8C6FAE 0F002161 ÿ···ÿ·Ã/ÏŒo®··!a
000005F0: 7FD41998 4A0CC87F 0534AD0D 0000A14D ·Ô·˜J·È··4-···¡M
00000600: 0A00D205 0000D205 00000008 D48A03F7 ··Ò····Ò·····ÔŠ·÷
00000610: 00E02A01 0B5F0800 450005C0 96BC4000 ·à*··_···E··À-¼@·
00000620: 0711541D C0A80102 C0A80101 7FBC157D ··T·À˚··À˚···¼·}
00000630: 05AC0000 806465B3 1277F593 7CC6EC6D ·¬···de³·w÷"|Æim
00000640: CF1F1D04 471F4410 FF00FF00 FFFF00FF Ï ··G D·ÿ·ÿ·ÿÿ·ÿ
00000650: 00FF00FF 00FF00FF 0000FF00 FF000000 ·ÿ·ÿ·ÿ·ÿ···ÿ·ÿ···
00000660: 0000FF00 FFFF0000 00FF00FF 00FF00FF ··ÿ·ÿÿ···ÿ·ÿ·ÿ·ÿ
00000670: 0000FF00 FF00FF00 FF00FF00 FF0000FF ··ÿ·ÿ·ÿ·ÿ·ÿ·ÿ··ÿ
00000680: 00FF00FF 00FF00FF 0000FF00 FF000000 ·ÿ·ÿ·ÿ·ÿ···ÿ·ÿ···
00000690: FF00FF00 000000FF 00FF00FF 00FF00FF ÿ·ÿ····ÿ·ÿ·ÿ·ÿ·ÿ
000006A0: 0000FF00 FFFFFF00 FF00FF00 FF0000FF ··ÿ·ÿÿÿ·ÿ·ÿ·ÿ··ÿ
000006B0: 00FF00FF 000000FF 00FF0000 0000FF00 ·ÿ·ÿ···ÿ·ÿ····ÿ·
000006C0: 0000FF00 FF00FFFF FFFF00FF 00FF00FF ··ÿ·ÿ·ÿÿÿÿ·ÿ·ÿ·ÿ
000006D0: 00FF0000 FF00FF00 FF00FF00 000000FF ·ÿ··ÿ·ÿ·ÿ·ÿ····ÿ
000006E0: 00FF00FF 00FF00FF 00FF0000 FF00FF00 ·ÿ·ÿ·ÿ·ÿ·ÿ··ÿ·ÿ·
000006F0: FF00FF00 FF00FF00 00FF00FF 00000096 ÿ·ÿ·ÿ·ÿ···ÿ·ÿ···-
00000700: 7D29A679 6F9A5D73 67DAF0DC 08DF7967 })¦yoš]sgÚðÜ·ßyg
00000710: 470CFF10 00FF00FF 0000FF00 0000FF00 G·ÿ··ÿ·ÿ·ÿ····ÿ·
00000720: FF000000 FFFF0000 00FF0000 00FF00FF ÿ···ÿÿ···ÿ···ÿ·ÿ
00000730: 0000FF00 FF00FF00 FF000000 FF0000FF ··ÿ·ÿ·ÿ·ÿ···ÿ··ÿ
00000740: 00FF0000 00FF00FF 00FF0000 FF00FF00 ·ÿ···ÿ·ÿ·ÿ··ÿ·ÿ·
00000750: FF000000 FF0000FF 00FF00FF 00FF00FF ÿ···ÿ··ÿ·ÿ·ÿ·ÿ·ÿ
00000760: 0000FF00 FF00FF00 0000FF00 FF0000FF ··ÿ·ÿ·ÿ···ÿ·ÿ··ÿ
00000770: 00FF00FF 00FF00FF 00FF0000 FF00FF00 ·ÿ·ÿ·ÿ·ÿ·ÿ··ÿ·ÿ·
00000780: FF00FF00 FF00FF00 00FF00FF 00FF00FF ÿ·ÿ·ÿ·ÿ··ÿ·ÿ·ÿ·ÿ
00000790: 00FF0000 0000FF00 FF00FF00 FF00FFFF ·ÿ····ÿ·ÿ·ÿ·ÿ·ÿÿ
000007A0: 00FF00FF 00FF00FF 00FF0000 FF00FF00 ·ÿ·ÿ·ÿ·ÿ·ÿ··ÿ·ÿ·
000007B0: 0000FF00 FF00FF00 000000FF 00FF00FF ··ÿ·ÿ·ÿ····ÿ·ÿ·ÿ
000007C0: 00FF00FF FF00FF00 FF00FF00 0C2FA679 ·ÿ·ÿÿ·ÿ·ÿ·ÿ··/¦y
000007D0: 6F950F0A 21657FDC 196F4A77 471F4410 o···!e·Ü·oJwG D·
000007E0: FF00FF00 FF0000FF 00FF00FF 00FF00FF ÿ·ÿ·ÿ··ÿ·ÿ·ÿ·ÿ·ÿ
000007F0: 0000FF00 FF00FF00 0000FF00 FF0000FF ··ÿ·ÿ·ÿ···ÿ·ÿ··ÿ
00000800: 00FF00FF 00FF00FF 00FFFF00 FF00FF00 ·ÿ·ÿ·ÿ·ÿ·ÿÿ·ÿ·ÿ·
00000810: FF00FF00 FF000000 00FF00FF 00FF00FF ÿ·ÿ·ÿ····ÿ·ÿ·ÿ·ÿ
00000820: 00FFFF00 FF00FF00 FF00FF00 000000FF ·ÿÿ·ÿ·ÿ·ÿ·ÿ····ÿ
00000830: 00FF00FF 00FF00FF FFFF0000 FF000000 ·ÿ·ÿ·ÿ·ÿÿÿ··ÿ···
00000840: FFFFFF00 FF0000FF 00FF00FF 00FF00FF ÿÿÿ·ÿ··ÿ·ÿ·ÿ·ÿ·ÿ
00000850: 00FF00FF FF00FF00 FF00FF00 FF000000 ·ÿ·ÿÿ·ÿ·ÿ·ÿ·ÿ···
00000860: FF0000FF 000000FF 00FF0000 FF000000 ÿ··ÿ····ÿ·ÿ··ÿ···
00000870: FF00FF00 FF00FF00 00FF00FF 00FF00FF ÿ·ÿ·ÿ·ÿ··ÿ·ÿ·ÿ·ÿ
00000880: 00FF00FF FF00FF00 FF00FF00 0000FF00 ·ÿ·ÿÿ·ÿ·ÿ·ÿ···ÿ·
00000890: 00FF00FF 000000FF 7DDBA678 EE9AB773 ·ÿ·ÿ···ÿ}Û¦xî š·s
000008A0: 21DA16DC 6DDF3F67 470CFF11 00FF00FF !Ú·Ümß?gG·ÿ··ÿ·ÿ
000008B0: 00FFFF00 FF00FF00 FF00FF00 FF0000FF ·ÿÿ·ÿ·ÿ·ÿ·ÿ·ÿ··ÿ
000008C0: 00FF0000 000000FF 00FFFF00 0000FF00 ·ÿ·····ÿ·ÿÿ···ÿ·
000008D0: FF00FF00 FF0000FF 00FF00FF 00FF00FF ÿ·ÿ·ÿ··ÿ·ÿ·ÿ·ÿ·ÿ
000008E0: 00FF0000 FF00FF00 FF00FF00 FF0000FF ·ÿ··ÿ·ÿ·ÿ·ÿ·ÿ··ÿ
000008F0: 00FF0000 00FF00FF 00000000 FF00FF00 ·ÿ···ÿ·ÿ····ÿ·ÿ·
00000900: FF00FF00 FF0000FF 00FFFFFF 00FF00FF ÿ·ÿ·ÿ··ÿ·ÿÿÿ·ÿ·ÿ
00000910: 00FF0000 FF00FF00 FF000000 FF00FF00 ·ÿ··ÿ·ÿ·ÿ···ÿ·ÿ·
00000920: 000000FF 000000FF 00FF00FF FFFFFF00 ···ÿ···ÿ·ÿ·ÿÿÿÿ·
00000930: FF00FF00 FF00FF00 00FF00FF 00FF00FF ÿ·ÿ·ÿ·ÿ··ÿ·ÿ·ÿ·ÿ
```

2 Appendix

2.1 Correspondences

2.1.1 Rudi Van de Genachte, 150506

Maria,

Sorry for the late response.

From my experience in IPTV, I have to admit that PRO-MPEG FEC isn't much of demand. The reason for this is that IP networks can be easily be scale to carry the traffic needed for the TV services. So, packet drops and error packets can be avoid by engineering correctly these networks. The problem can arise in the access part of the network like ADSL. But, the access technologies used have already some FEC methods implemented. Conclusion the need for PRO-MPEG FEC is limited for IPTV application. Nevertheless, it doesn't exclude that FEC can contribute to a better service level or can be of an advantage for IPTV operator. It is also true that Telecom companies like to stick close to the standards, so when FEC is or becomes a standard they will require it.

I believe the advantages of FEC will be more required in contribution application or primary distribution, where higher standards of errorless transmission are needed. Also typical for these application is that transmission capacity (bandwidth) will be leased for a certain period, and therefore the only impact that the sender has on the quality of the transmission is to use FEC.

I suggest also that you talk to our Market management department and possibly to Cisco service provider departments to help from your idea about FEC.

Best regards,

Rudi

-----Original Message-----
From: Baltzer, Maria
Sent: donderdag 4 mei 2006 12:43
To: Van de Genachte, Rudi
Subject: Forward Error Correction on IP output
Hi Rudi

The reason for me contacting you is that I was told that you might have some insight in the subject of Forward Error Correction, so here is a little about me and the cause of me contacting you.

I am currently a student at the Technological University of Denmark where I am finishing my degree in engineering, within the field of IT and electronics. This last period of the education includes ten weeks of practical experience in a company, working and preparing for my exam project. Following these ten weeks is the actual project which also spans over ten weeks. I am

writing the project in close cooperation with, surprisingly, Scientific Atlanta Denmark which explains the e-mail address.

The subject for my project and the research I am currently doing concerns Forward Error Correction, and more specifically the implementation of a FEC algorithm for the IP output in the D9054 encoder. This includes market analysis to determine what the clients are expecting, literary studies into the area of FEC and of course research into the suggestions made by e.g. the Pro-MPEG group as to which algorithm to implement.

The research and results of this analysis is very important for S-A since some FEC solution is to be implemented in the encoder and the more I find out, the easier it is for S-A to implement. Therefore, this is not just some imaginary project but a project of great relevance.

The problem I am facing right now is the decision of which solution to implement (XOR, Reed Solomon, interleaving…) and I have yet to meet someone who could describe an application where FEC is actually needed – maybe you have some ideas?

I hope you can help me or maybe direct me to someone who can.

Med venlig hilsen / Best regards

Maria Baltzer Pedersen
Project student
Scientific Atlanta, A Cisco Company
Tobaksvejen 23A
DK-2860 Søborg
mailto: maria.baltzer@sciatl.com
Tel.: +45 39 17 08 54

2.1.2 Richard Diaz, 240206

Hi Maria

I am curious as to why you picked FEC for your project?

I am working on system level designs for customers wishing to implement IPTV networks. One of the things I am finding is that there is very little understanding by customers on FEC.

The biggest issue I see with all FEC solutions is that they consume too much bandwidth. Bandwidth savings is the holy grail of IPTV networks. The way customers see it FEC consumes bandwidth but provides no cash income. If the switched network was clean enough then you wouldn't need FEC. Some switch vendors like Cisco and Alacatel do offer some error correction in the hardware.

The other issue with FEC is compatibility. Settop box vendors need to use the same FEC algorithm that the source is sending. That is not an issue when both the encoder and the settop use the same standard. So thats why we are looking at Pro-MPEG COP3 open solution to provide a common solution. However that being said the COP3 solution is not the best solution, it has made some compromises. In my opinion we, (SA), should create a solution that could be used when both our encoders and settops are used together but also offer COP3 for those installations that require integration with other settops.

Take a look at Tut Systems and Path1. Both of those companies offer FEC solutions in hardware.

Good Luck with your project

Rick

From: Baltzer, Maria
Sent: Friday, February 24, 2006 8:41 AM
To: Diaz, Richard
Cc: Pedersen, Lars; Egede, Niels; Nielsen, Ole Stender
Subject: Forward Error Correction on IP output
Hi Rick

The reason for this mail is that I have been told that maybe you could help me.

I am currently a student at the Technological University of Denmark where I am finishing my degree in engineering, within the field of IT and electronics. This last period of the education includes ten weeks of practical experience in a company, working and preparing for my exam project. Following these ten weeks is the actual project which also spans over ten weeks. I am writing the project in close cooperation with, surprisingly, Scientific Atlanta Denmark which explains the e-mail address.

The subject for my project and the research I am currently doing concerns Forward Error Correction, and more specifically the implementation of a FEC algorithm for the IP output in the D9054 encoder. This includes market analysis to determine what the clients are expecting, literary studies into the area of FEC and of course research into the suggestions made by e.g. the Pro-MPEG group as to which algorithm to implement.

This brings me to you and what I hope you can help me with. I was told by Lars Pedersen that you were currently working with applying some sort of FEC to the IP output of an encoder and I

was hoping to maybe I could benefit from your experience, and also if you have any recommendations or thoughts as to which, if any, algorithms to concentrate the research.

I hope that you can help me in my quest or maybe guide me in the direction of someone who can.

Best regards and have a nice weekend,

Maria Baltzer Pedersen
Project student
Scientific-Atlanta Denmark A/S
Tel.: +45 39 17 08 54
maria.baltzer@sciatl.com

2.1.3 Richard Diaz, 280206

Hi Maria

See Below

From: Baltzer, Maria
Sent: Monday, February 27, 2006 9:41 AM
To: Diaz, Richard
Subject: Forward Error Correction on IP output

-----Original Message-----
From: Baltzer, Maria
Sent: 27. februar 2006 15:39
To: Baltzer, Maria
Subject: Forward Error Correction on IP output

Hi Rick

I understand your curiosity as to why I chose FEC but actually it comes down to the fact that I wanted to do a project involving hardware programming, especially VHDL. To find a suitable project I applied to several companies and one of the projects I was offered was this one. And since it involves both programming (hopefully) and is a project of great importance for SA in the development of the D9054 encoder, I found is very interesting. Also I had some basic knowledge of Forward Error Correction prior to the project.

Now back to business. Of course a FEC solution would result in some excess data being sent over the network which, in most cases will never be needed, but don't you think that some customers would want the option. After all FEC is at some level a buzzword at the moment what with the Pro-MPEG COP3 and all. When you say that customers see it as a downside since it does not provide direct cash income, are you referring to any specific customers or is it in general. I have the impression that since it is becoming a buzzword, more and more customers will want the FEC option in new encoders.

When I mean customers I am refereeing to Telcos or service providers. The lost bandwidth to FEC can not be sold by Telco's. IE; If I have a 1 gigabit/s pipe but I have to set aside 200 mbit/s for FEC then I will have "lost" the opportunity to make any revenue on that bandwidth.

I see the problem with the choice of FEC algorithm which is why I want to find out if any of the potential customers have any thought on which algorithm to use. Maybe some have read the COP3 or has other preferences from elsewhere. It is hard to tell. And furthermore it is hard to predict whether anyone is actually going to need this FEC. It depends on so many things; the error correction already in the network switches as you mention yourself, where the switches and encoders are physically placed and so on…

From what I have seen, the Telco's and service providers are still investigating the options.

You say that the COP3 solution is not the best solution but do you have any suggestions to other algorithms? I'd like to hear what you think. The proprietary solution from Path1 goes a step further than cop3. I got a verbal overview from Path1 when the rep was here.

I know that Tandberg has already implemented some FEC solution – or so they claim – but which algorithm do they use? They have a summery of the COP3 solution on their webpage and one would think that this is the algorithm they have implemented, but I haven't had the chance to confirm it. Do you know? I have a TB encoder here I will check it out.

As you know, Tut Systems and Path1 also utilize the COP3 recommendations, would this be a reason to implement it in SA products also?

At this point the settop group has to decide what to implement on the settop software. They know if they choose COP3 they will have better success integrating with other encoders vendors that use COP3 or third party devices.

I hope you will continue to share your experiences and knowledge on the subject

I have a question for you. What do you think is more efficient to implement, FEC on one SPTS then combine all SPTS on a switch or FEC on a MPTS?

Best regards

Maria Baltzer Pedersen
Project student
Scientific-Atlanta Denmark A/S
Tel.: +45 39 17 08 54
maria.baltzer@sciatl.com

2.1.4 Richard Diaz, 140306

Hi Rick

First, I'm sorry I haven't replied sooner, but I have been quite busy.

As for the question you asked me in the mail I received from you, I do not have one simple answer.

>> I have a question for you. What do you think is more efficient to implement, FEC on one SPTS then combine all SPTS on a switch or FEC on a MPTS? <<

As I see it, to include FEC in the SA encoders the solution has to be implemented on the SPTS and combined into one stream later otherwise, it would need a switch or some other device to apply the FEC. Also, couldn't you be in a situation where an application consists of various different encoders, which all provide a SPTS later to be combined into one stream? In this case, the FEC should apply to the SPTS from the SA encoder, and not to the others.

Please correct me if I'm wrong. I am still trying to figure out what it is everybody wants ☐

Best regards

Maria Baltzer Pedersen

-----Original Message-----
From: Diaz, Richard
Sent: 28. februar 2006 00:01
To: Baltzer, Maria
Subject: RE: Forward Error Correction on IP output

Hi Maria

See Below

From: Baltzer, Maria
Sent: Monday, February 27, 2006 9:41 AM
To: Diaz, Richard
Subject: Forward Error Correction on IP output

-----Original Message-----
From: Baltzer, Maria
Sent: 27. februar 2006 15:39
To: Baltzer, Maria
Subject: Forward Error Correction on IP output

Hi Rick

I understand your curiosity as to why I chose FEC but actually it comes down to the fact that I wanted to do a project involving hardware programming, especially VHDL. To find a suitable project I applied to several companies and one of the projects I was offered was this one. And since it involves both programming (hopefully) and is a project of great importance for SA in the development of the D9054 encoder, I found is very interesting. Also I had some basic knowledge of Forward Error Correction prior to the project.

Now back to business☐. Of course a FEC solution would result in some excess data being sent over the network which, in most cases will never be needed, but don't you think that some customers would want the option. After all FEC is at some level a buzzword at the moment what with the Pro-MPEG COP3 and all. When you say that customers see it as a downside since it does not provide direct cash income, are you referring to any specific customers or is it in general. I have the impression that since it is becoming a buzzword, more and more customers will want the FEC option in new encoders.

When I mean customers I am refereeing to Telcos or service providers. The lost bandwidth to FEC can not be sold by Telco's. IE; If I have a 1 gigabit/s pipe but I have to set aside 200 mbit/s for FEC then I will have "lost" the opportunity to make any revenue on that bandwidth.

I see the problem with the choice of FEC algorithm which is why I want to find out if any of the potential customers have any thought on which algorithm to use. Maybe some have read the COP3 or has other preferences from elsewhere. It is hard to tell. And furthermore it is hard to predict whether anyone is actually going to need this FEC. It depends on so many things; the error correction already in the network switches as you mention yourself, where the switches and encoders are physically placed and so on…

From what I have seen, the Telco's and service providers are still investigating the options.

You say that the COP3 solution is not the best solution but do you have any suggestions to other algorithms? I'd like to hear what you think. The proprietary solution from Path1 goes a step further than cop3. I got a verbal overview from Path1 when the rep was here.

I know that Tandberg has already implemented some FEC solution – or so they claim – but which algorithm do they use? They have a summery of the COP3 solution on their webpage and one would think that this is the algorithm they have implemented, but I haven't had the chance to confirm it. Do you know? I have a TB encoder here I will check it out.

As you know, Tut Systems and Path1 also utilize the COP3 recommendations, would this be a reason to implement it in SA products also?

At this point the settop group has to decide what to implement on the settop software. They know if they choose COP3 they will have better success integrating with other encoders vendors that use COP3 or third party devices.

I hope you will continue to share your experiences and knowledge on the subject

I have a question for you. What do you think is more efficient to implement, FEC on one SPTS then combine all SPTS on a switch or FEC on a MPTS?

Best regards

Maria Baltzer Pedersen
Project student
Scientific-Atlanta Denmark A/S
Tel.: +45 39 17 08 54
maria.baltzer@sciatl.com

-----Original Message-----
From: Diaz, Richard
Sent: 24. februar 2006 17:02

To: Baltzer, Maria
Subject: RE: Forward Error Correction on IP output

Hi Maria

I am curious as to why you picked FEC for your project?

I am working on system level designs for customers wishing to implement IPTV networks. One of the things I am finding is that there is very little understanding by customers on FEC.

The biggest issue I see with all FEC solutions is that they consume too much bandwidth. Bandwidth savings is the holy grail of IPTV networks. The way customers see it FEC consumes bandwidth but provides no cash income. If the switched network was clean enough then you wouldn't need FEC. Some switch vendors like Cisco and Alacatel do offer some error correction in the hardware.

The other issue with FEC is compatibility. Settop box vendors need to use the same FEC algorithm that the source is sending. That is not an issue when both the encoder and the settop use the same standard. So thats why we are looking at Pro-MPEG COP3 open solution to provide a common solution. However that being said the COP3 solution is not the best solution, it has made some compromises. In my opinion we, (SA), should create a solution that could be used when both our encoders and settops are used together but also offer COP3 for those installations that require integration with other settops.

Take a look at Tut Systems and Path1. Both of those companies offer FEC solutions in hardware.

Good Luck with your project

Rick

From: Baltzer, Maria
Sent: Friday, February 24, 2006 8:41 AM
To: Diaz, Richard
Cc: Pedersen, Lars; Egede, Niels; Nielsen, Ole Stender
Subject: Forward Error Correction on IP output
Hi Rick

The reason for this mail is that I have been told that maybe you could help me.

I am currently a student at the Technological University of Denmark where I am finishing my degree in engineering, within the field of IT and electronics. This last period of the education includes ten weeks of practical experience in a company, working and preparing for my exam project. Following these ten weeks is the actual project which also spans over ten weeks. I am writing the project in close cooperation with, surprisingly, Scientific Atlanta Denmark which explains the e-mail address.

The subject for my project and the research I am currently doing concerns Forward Error Correction, and more specifically the implementation of a FEC algorithm for the IP output in the D9054 encoder. This includes market analysis to determine what the clients are expecting, literary studies into the area of FEC and of course research into the suggestions made by e.g. the Pro-MPEG group as to which algorithm to implement.

This brings me to you and what I hope you can help me with. I was told by Lars Pedersen that you were currently working with applying some sort of FEC to the IP output of an encoder and I was hoping to maybe I could benefit from your experience, and also if you have any recommendations or thoughts as to which, if any, algorithms to concentrate the research.

I hope that you can help me in my quest or maybe guide me in the direction of someone who can.

Best regards

Maria Baltzer Pedersen
Project student
Scientific-Atlanta Denmark A/S
Tel.: +45 39 17 08 54
maria.baltzer@sciatl.com

2.1.5 William Van Nieuwenhove, 050506

Maria,

My experience with the usage of FEC is limited. The main application seems to be for transport of video directly to the home over uncontrolled or public networks. On private networks from headend to hub, the operator has a lot of control over the network and FEC is usually not required. If video is distributed to end-users that are connected via a modem or IP STB to the network than sometimes bandwidth is not that well controlled and the operator wants some extra security. That is why FEC is recommended in the DVB-IPI spec for delivery to end-users.

Hope this helps,
William
-----Original Message-----
From: Baltzer, Maria
Sent: donderdag 4 mei 2006 12:35
To: Van Nieuwenhove, William
Subject: Forward Error Correction on IP output

Hi William
The reason for me contacting you is that I heard you speak at Scientific Atlanta Denmark some weeks ago.

I am currently a student at the Technological University of Denmark where I am finishing my degree in engineering, within the field of IT and electronics. This last period of the education includes ten weeks of practical experience in a company, working and preparing for my exam project. Following these ten weeks is the actual project which also spans over ten weeks. I am writing the project in close cooperation with, surprisingly, Scientific Atlanta Denmark which explains the e-mail address.

The subject for my project and the research I am currently doing concerns Forward Error Correction, and more specifically the implementation of a FEC algorithm for the IP output in the D9054 encoder. This includes market analysis to determine what the clients are expecting, literary studies into the area of FEC and of course research into the suggestions made by e.g. the Pro-MPEG group as to which algorithm to implement. Here is where you come into the picture. You mentioned, only briefly, FEC and I would like to know if you have any further thoughts on the subject.

The research and results of this analysis is very important for S-A since some FEC solution is to be implemented in the encoder and the more I find out, the easier it is for S-A to implement. Therefore, this is not just some imaginary project but a project of great relevance. The problem I am facing right now is the decision of which solution to implement (XOR, Reed Solomon, interleaving…) and I have yet to meet someone who could describe an application where FEC is actually needed – maybe you have some ideas?

I hope you can help me or maybe direct me to someone who can.
Med venlig hilsen / Best regards

Maria Baltzer Pedersen
Project student
Scientific Atlanta, A Cisco Company
Tobaksvejen 23A
DK-2860 Søborg
mailto: maria.baltzer@sciatl.com
Tel.: +45 39 17 08 54

2.1.6 Simon Spraggs, 180406

No problem

From: Baltzer, Maria [mailto:maria.baltzer@sciatl.com]
Sent: 18 April 2006 09:23
To: Simon Spraggs (sspraggs)
Subject: RE: Forward Error Correction on IP output
Simon,

hehe, no problem – I get pretty carried away when someone mentions FEC…
I'll call you later today? Would that be ok?

Maria

-----Original Message-----
From: Simon Spraggs (sspraggs) [mailto:sspraggs@cisco.com]
Sent: 18. april 2006 10:14
To: Baltzer, Maria
Subject: RE: Forward Error Correction on IP output

Maria,
Willy has oversold my knowledge of FEC. I'm involved in the network design for SP environments and as a consequence see potential applications for FEC. Personally I'm not involved in the codec design at all, nor the precise FEC solutions. However I do know alot of people in Cisco and can probably point you in the right direction. When you get a chance I suggest you phone me on 447802262019 or I can phone you to discuss what your project involves and see how I can help.
Simon

From: Baltzer, Maria [mailto:maria.baltzer@sciatl.com]
Sent: 18 April 2006 08:53
To: Simon Spraggs (sspraggs)
Subject: RE: Forward Error Correction on IP output
Hi Simon,

I am very grateful that you will share your knowledge and look forward to discuss the topic.

First, I'd like to know what your experiences with FEC are, and of course I have some questions for you – this is only the top of the iceberg or so you say. I really hope you can help me in my search for the perfect or nearly perfect FEC solution for Scientific Atlanta.

So far I have come to the conclusion that a FEC solution in an encoder is only necessary for contributors and not for distributors since the stream from the encoders often is stripped of the IP headers and split up in TS packets at a server, before being repacked, multiplexed with other streams, and distributed. And often the network from the encoder to the server is a secure local network without any real chance of loosing packets. Therefore the FEC solution is mostly relevant to contributors since these streams are sent directly from the encoder over the internet, and this is where a FEC implementation in the encoder could be useful.

Do you agree with me or do you have some further insight to this? Anything will be useful.

Furthermore, I have to determine which algorithm or solution to implement, and this has proved to be more complicated than expected. Of course we have the Pro-MPEG recommendation of the two-dimensional XOR (CoP3), which would be the obvious choice but there is also the Reed-Solomon or maybe a simple interleaving solution. I have tried to find out what other encoders do, but also this is not as easy as expected. I have found out that Tut Systems use the CoP3 or so they claim (I haven't checked it). Tandberg has a description of CoP3 on their site but does not explicitly say that that is the solution implemented in their encoder. I have tried to analyse the output from one of their encoders which should have FEC, and it seems to be a proprietary solution and I can't find out what they do.

Therefore, to have greater interoperability with other producers of both encoders and decoders I think the CoP3 solution would be the way to go, but what is your opinion?

Well, I think this is enough for now. We have just celebrated Easter and if you have too, I hope you had a nice one.

Med venlig hilsen / Best regards

Maria Baltzer Pedersen
Project student
Scientific Atlanta, A Cisco Company
Tobaksvejen 23A
DK-2860 Søborg
mailto: maria.baltzer@sciatl.com
Tel.: +45 39 17 08 54

-----Original Message-----
From: Willy Verplancke (wverplan) [mailto:wverplan@cisco.com]
Sent: 7. april 2006 17:14
To: Baltzer, Maria
Cc: Simon Spraggs (sspraggs)
Subject: RE: Forward Error Correction on IP output

Maria,

Simon Spraggs has volunteered to share his knowledge on the topic. Feel free to engage with him, he has done some research on the topic and definately can put he topic in perspective.

Br,

Willy

From: Baltzer, Maria [mailto:maria.baltzer@sciatl.com]
Sent: Friday, April 07, 2006 1:53 PM
To: Willy Verplancke (wverplan)
Subject: Forward Error Correction on IP output
Hi Willy

First, thank you for helping me, or at least trying to help me – it is very difficult to get in contact with the right people.

Here is a little about me, and what I am doing:

I am currently a student at the Technological University of Denmark where I am finishing my degree in engineering, within the field of IT and electronics. This last period of the education includes ten weeks of practical experience in a company, working and preparing for my exam project. Following these ten weeks is the actual project which also spans over ten weeks. I am

writing the project in close cooperation with, surprisingly, Scientific Atlanta Denmark which explains the e-mail address.

The subject for my project and the research I am currently doing concerns Forward Error Correction, and more specifically the implementation of a FEC algorithm for the IP output in the D9054 encoder. This includes market analysis to determine what the clients are expecting, literary studies into the area of FEC and of course research into the suggestions made by e.g. the Pro-MPEG group as to which algorithm to implement.

The research and results of this analysis is very important for S-A since some FEC solution is to be implemented in the encoder and the more I find out, the easier it is for S-A to implement. Therefore, this is not just some imaginary project but a project of great relevance.

What I hope you can help me with is to direct me to the right people within Cisco who might be able to share some knowledge or experiences on the subject. I am interested in everything from whitepapers and what ever kind of knowledge to actual practical experience with Forward Error Correction.

Just so you know, I am on Easter holiday next week and will be back again on the 18th

Have a nice weekend,
Med venlig hilsen / Best regards

Maria Baltzer Pedersen
Project student
Scientific Atlanta, A Cisco Company
Tobaksvejen 23A
DK-2860 Søborg
mailto: maria.baltzer@sciatl.com
Tel.: +45 39 17 08 54

3 Appendix

3.1 Image Sampling Formats

television system. Source formats are referenced in SMPTE 260M and ANSI/SMPTE 274M.

3.2 interim specifications: Values given in brackets are interim and subject to revision following further investigation by the SMPTE Committee on Television Signal Technology (see 8.1.2, 8.1.9, 8.2.1, and 9.1).

4 Source format data

4.1 Source data shall be 10-bit words representing an E_Y', E_{C_B}', E_{C_R}' signal, where E_Y' is one formatted parallel data stream and E_{C_B}', E_{C_R}' is a second formatted parallel data stream. This limits the serial data rate to 1.5 Gb/s although the source format parallel data may allow higher data rates for RGB or Y, C_B, C_R keytype operation.

4.2 Data for each television line are divided into four areas: SAV (start of active video) timing reference, digital active line, EAV (end of active video) timing reference, and digital line blanking as shown in figure 1. The number of words and defined data in each area are specified by the source format document.

4.3 Since not all bit-parallel digital television data formats have the same timing reference data, a modification may be required prior to multiplexing and serialization in order to meet the requirements of clause 5. Where additional words are required for EAV/SAV, data words from the adjacent digital blanking area shall be used. Modifications are typically made using a coprocessor in the parallel domain.

4.4 Parameters for referenced source formats are shown in table 1.

4.5 The total data rate is either 1.485 Gb/s or 1.485/1.001 Gb/s. In table 1, the former is indicated by a data rate division of 1 and the latter by a divsor of M, which equals 1.001.

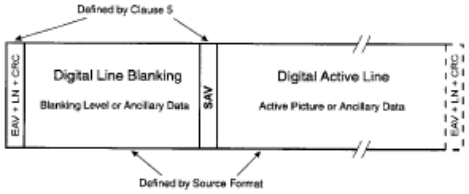


Figure 1 – Television horizontal line data

Table 1 -- Source format parameters

Reference SMPTE standard	260M		295M	274M								296M	
Format	A	B	C	D	E	F	G	H	I	J	K	L	M
Lines per frame	1125	1125	1250	1125	1125	1125	1125	1125	1125	1125	1125	750	750
Words per active line (each channel Y C_B/C_R)	1920	1920	1920	1920	1920	1920	1920	1920	1920	1920	1920	1280	1280
Total active lines	1035	1035	1080	1080	1080	1080	1080	1080	1080	1080	1080	720	720
Words per total line (each channel Y C_B/C_R)	2200	2200	2376	2200	2200	2640	2200	2200	2640	2750	2750	1650	1650
Frame rate (Hz)	30	30/M	25	30	30/M	25	30	30/M	25	24	24/M	60	60/M
Fields per frame	2	2	2	2	2	2	1	1	1	1	1	1	1
Data rate divisor (see 4.5)	1	M	1	1	M	1	1	M	1	1	M	1	M

SMPTE STANDARD

SMPTE 296M-2001
Revision of
ANSI/SMPTE 296M-1997

for Television — 1280 × 720 Progressive Image Sample Structure — Analog and Digital Representation and Analog Interface

Contents

1 Scope

1.1 This standard defines a family of progressive image sample systems for the representation of stationary or moving two-dimensional images sampled temporally at a constant frame rate and having an image format of 1280 pixels by 720 lines and an aspect ratio of 16:9 as given in table 1. All systems in the table have the common characteristic that all the samples gathered within a single temporal unit, a frame, shall be spatially contiguous and provide a complete description of that frame (4.2) This standard specifies:

– R′G′B′ color encoding;
– R′G′B′ analog and digital representation;
– $Y'P'_BP'_R$ color encoding, analog representation, and analog interface; and
– $Y'C'_BC'_R$ color encoding and digital representation.

Table 1 – Image sampling systems

	System nomenclature	Luma or R′G′B′ samples per active line (S/AL)	Active lines per frame (AL/F)	Frame rate, Hz	Luma or R′G′B′ sampling frequency (fs), MHz	Luma sample periods per total line (S/TL)	Total lines per frame
1	1280 × 720/60	1280	720	60	74.25	1650	750
2	1280 × 720/59.94	1280	720	60/1.001	74.25/1.001	1650	750
3	1280 × 720/50	1280	720	50	74.25	1980	750
4	1280 × 720/30	1280	720	30	74.25	3300	750
5	1280 × 720/29.97	1280	720	30/1.001	74.25/1.001	3300	750
6	1280 × 720/25	1280	720	25	74.25	3960	750
7	1280 × 720/24	1280	720	24	74.25	4125	750
8	1280 × 720/23.98	1280	720	24/1.001	74.25/1.001	4125	750
NOTE – For systems 4 through 8, analog video interface is not preferred. See clause 12.							

Approved
January 4, 2001

4 Appendix

4.1 FEC calculations

Stream	Mbyte/sec	packets/sec	packets/hour
15	1,875	1250	4500000

This table was calculated based on the delay in the decoder.

delay in msec	msec/packet	packets	+packets	Corrected packets %:	extra bandwidth in %	Bandwidth total in Mbit	Corrected packets per hour
30	0,8	37,5	38	2,67	2,60	15,39	120000
29	0,8	36,25	36	2,76	2,68	15,40	124137,931
28	0,8	35	35	2,86	2,78	15,42	128571,4286
27	0,8	33,75	34	2,96	2,88	15,43	133333,3333
26	0,8	32,5	33	3,08	2,99	15,45	138461,5385
25	0,8	31,25	31	3,20	3,10	15,47	144000
24	0,8	30	30	3,33	3,23	15,48	150000
23	0,8	28,75	29	3,48	3,36	15,50	156521,7391
22	0,8	27,5	28	3,64	3,51	15,53	163636,3636
21	0,8	26,25	26	3,81	3,67	15,55	171428,5714
20	0,8	25	25	4,00	3,85	15,58	180000
19	0,8	23,75	24	4,21	4,04	15,61	189473,6842
18	0,8	22,5	23	4,44	4,26	15,64	200000
17	0,8	21,25	21	4,71	4,49	15,67	211764,7059
16	0,8	20	20	5,00	4,76	15,71	225000
15	0,8	18,75	19	5,33	5,06	15,76	240000
14	0,8	17,5	18	5,71	5,41	15,81	257142,8571
13	0,8	16,25	16	6,15	5,80	15,87	276923,0769
12	0,8	15	15	6,67	6,25	15,94	300000
11	0,8	13,75	14	7,27	6,78	16,02	327272,7273
10	0,8	12,5	13	8,00	7,41	16,11	360000
9	0,8	11,25	11	8,89	8,16	16,22	400000
8	0,8	10	10	10,00	9,09	16,36	450000
7	0,8	8,75	9	11,43	10,26	16,54	514285,7143
6	0,8	7,5	8	13,33	11,76	16,76	600000
5	0,8	6,25	6	16,00	13,79	17,07	720000
4	0,8	5	5	20,00	16,67	17,50	900000
3	0,8	3,75	4	26,67	21,05	18,16	1200000
2	0,8	2,5	3	40,00	28,57	19,29	1800000
1	0,8	1,25	1	80,00	44,44	21,67	3600000
0	0,8	0	0				

This table was calculated based on the number of packets included the FEC.

packets	msec/packet	dealy	Corrected packets	extra bandwidth in %	Bandwidth total in Mbit	Corrected packets per h
1	0,8	0,8	100,00	50,00	22,50	4500000
2	0,8	1,6	50,00	33,33	20,00	2250000
3	0,8	2,4	33,33	25,00	18,75	1500000
4	0,8	3,2	25,00	20,00	18,00	1125000
5	0,8	4	20,00	16,67	17,50	900000
6	0,8	4,8	16,67	14,29	17,14	750000
7	0,8	5,6	14,29	12,50	16,88	642857
8	0,8	6,4	12,50	11,11	16,67	562500
9	0,8	7,2	11,11	10,00	16,50	500000
10	0,8	8	10,00	9,09	16,36	450000
11	0,8	8,8	9,09	8,33	16,25	409091
12	0,8	9,6	8,33	7,69	16,15	375000
13	0,8	10,4	7,69	7,14	16,07	346154
14	0,8	11,2	7,14	6,67	16,00	321429
15	0,8	12	6,67	6,25	15,94	300000
16	0,8	12,8	6,25	5,88	15,88	281250
17	0,8	13,6	5,88	5,56	15,83	264706
18	0,8	14,4	5,56	5,26	15,79	250000
19	0,8	15,2	5,26	5,00	15,75	236842
20	0,8	16	5,00	4,76	15,71	225000
21	0,8	16,8	4,76	4,55	15,68	214286
22	0,8	17,6	4,55	4,35	15,65	204545
23	0,8	18,4	4,35	4,17	15,63	195652
24	0,8	19,2	4,17	4,00	15,60	187500
25	0,8	20	4,00	3,85	15,58	180000
26	0,8	20,8	3,85	3,70	15,56	173077
27	0,8	21,6	3,70	3,57	15,54	166667
28	0,8	22,4	3,57	3,45	15,52	160714
29	0,8	23,2	3,45	3,33	15,50	155172
30	0,8	24	3,33	3,23	15,48	150000
31	0,8	24,8	3,23	3,13	15,47	145161
32	0,8	25,6	3,13	3,03	15,45	140625
33	0,8	26,4	3,03	2,94	15,44	136364
34	0,8	27,2	2,94	2,86	15,43	132353
35	0,8	28	2,86	2,78	15,42	128571
36	0,8	28,8	2,78	2,70	15,41	125000
37	0,8	29,6	2,70	2,63	15,39	121622
38	0,8	30,4	2,63	2,56	15,38	118421
39	0,8	31,2	2,56	2,50	15,38	115385
40	0,8	32	2,50	2,44	15,37	112500
41	0,8	32,8	2,44	2,38	15,36	109756
42	0,8	33,6	2,38	2,33	15,35	107143
43	0,8	34,4	2,33	2,27	15,34	104651
44	0,8	35,2	2,27	2,22	15,33	102273
45	0,8	36	2,22	2,17	15,33	100000
46	0,8	36,8	2,17	2,13	15,32	97826
47	0,8	37,6	2,13	2,08	15,31	95745
48	0,8	38,4	2,08	2,04	15,31	93750
49	0,8	39,2	2,04	2,00	15,30	91837
50	0,8	40	2,00	1,96	15,29	90000

5 Appendix

5.1 Source Code

5.1.1 Top Entity of the FEC system

```
-- ********************************************************************************************
-- * Scientific Atlanta Denmark A/S
-- ********************************************************************************************
-- * Project      : HD Encoder D9054
-- *
-- * Used in      : FEC
-- *
-- * Description  : Top entity of the FEC system
-- *
-- * Made by Maria Baltzer, Technical University of Denmark for Scientific Atlanta Denmark
-- ********************************************************************************************
Library IEEE;
Use     IEEE.STD_Logic_1164.all;
Use     IEEE.STD_Logic_arith.all;
Use     IEEE.std_logic_unsigned.all;

entity FECTop is
port (
        RESET             : in  std_logic;
        CLK               : in  std_logic;
        CLK_EXT           : in  std_logic; --PCI clock
        TSMODE            : in  std_logic;
        DSRC_MEM_SERVICED : in  std_logic;
        DSRC_MEM_SELECT   : in  std_logic;
        DATAREADY         : in std_logic;
        BUFFERSELECT      : in std_logic_vector(3 downto 0);
        SEQUENCE_NUMBER   : in std_logic_vector(15 downto 0);
        DATAVALID         : in std_logic;
        GRANT             : in std_logic;
        BUFFEROVERFLOW    : out std_logic;
        MEMREADY          : out std_logic;
        MEMSELECT         : out std_logic;
        PCI_DPRAM_ADDR    : in  std_logic_vector(9 downto 0);
        PCI_DPRAM_DIN     : in  std_logic_vector(31 downto 0);
        PCI_DPRAM_DOUT    : out  std_logic_vector(31 downto 0);
        PCI_DPRAM_EN      : in  std_logic;
        PCI_DPRAM_WE      : in  std_logic;
        PCI_HEADER_ADDR   : in  std_logic_vector(8 downto 0);
        PCI_HEADER_DIN    : in  std_logic_vector(31 downto 0);
        PCI_HEADER_DOUT   : out  std_logic_vector(31 downto 0);
        PCI_HEADER_EN     : in  std_logic;
        PCI_HEADER_WE     : in  std_logic;
        FIFODATAIN        : in  std_logic_vector(31 downto 0);
        FIFOFULL          : out std_logic
);
end FECTop;


architecture Behavioral of FECTop is

-- Components

component FECProcessing
port
(
    reset            : in  std_logic;
    clk              : in  std_logic;
    FifoOutputData   : in  std_logic_vector(31 downto 0);
```

```
    FifoEmpty        : in  std_logic;
    FifoOutputEnable : out std_logic;
    HeaderRamRE      : out std_logic;
    HeaderRamAddress : out std_logic_vector(8 downto 0);
    HeaderRamData    : in  std_logic_vector(31 downto 0);
    DPRAMSelect      : out std_logic;
    DPRAMAddress     : out std_logic_vector(9 downto 0);
    DPRAMData        : out std_logic_vector(31 downto 0);
    DPRAMWE          : out std_logic;
    FECRamWeA        : out std_logic;
    FECRamReA        : out std_logic;
    FECRamAddressA   : out std_logic_vector(8 downto 0);
    FECRamDataA      : out std_logic_vector(31 downto 0);
    FECRamWeB        : out std_logic;
    FECRamReB        : out std_logic;
    FECRamAddressB   : out std_logic_vector(8 downto 0);
    FECRamDataB      : in  std_logic_vector(31 downto 0);
    SequenceNumber1bit: in  std_logic;
    TSMode           : in  std_logic;
    LastWord         : in  std_logic;
    FECPacketReady   : out std_logic;
    CounterReset     : in  std_logic

);

end component;

component FECLastWordCounter
port
(
   reset             : in std_logic;
   clk               : in std_logic;
   TsMode            : in std_logic; -- 0 = 188, 1 = 204 TS packets
   DataReadyFromFifo : in std_logic;
   FifoOutputEnable  : in std_logic;
   CounterReset      : out std_logic;
   LastWord          : out std_logic
);
end component;


component FECPacketSupervising
port
(
   reset          : in std_logic;
   clk            : in  std_logic;
   FECPacketReady : in  std_logic;
   DPRAMSelect    : in  std_logic;
   BufferOverflow : out std_logic;
   MemReady       : out std_logic; -- næste pakke er klar i ram'en
   MemSelect      : out std_logic;
   DSRCMemServiced: in  std_logic;
   DSRCMemSelect  : in  std_logic --flag der fortæller at bufferen er serviceret.
);
end component;

component FECInFifoInputEnable
port
(
   reset                   : in std_logic;
   clk                     : in  std_logic;
   DataReady               : in  std_logic;
   BufferSelect            : in  std_logic_vector(3 downto 0);
   SequenceNumber16bit     : in  std_logic_vector(15 downto 0);
   DataValid               : in  std_logic;
   Grant                   : in  std_logic;
   SequenceNumberOutbit    : out std_logic;
   FifoInputEnable         : out std_logic
);
end component;

component fec_output
    port (
    addra: IN std_logic_VECTOR(9 downto 0);
    addrb: IN std_logic_VECTOR(9 downto 0);
    clka: IN std_logic;
    clkb: IN std_logic;
    dina: IN std_logic_VECTOR(31 downto 0);
    dinb: IN std_logic_VECTOR(31 downto 0);
```

```
    doutb: OUT std_logic_VECTOR(31 downto 0);
    enb: IN std_logic;
    wea: IN std_logic;
    web: IN std_logic);
end component;


component fec_input
    port (
    clk: IN std_logic;
    din: IN std_logic_VECTOR(33 downto 0);
    rd_en: IN std_logic;
    rst: IN std_logic;
    wr_en: IN std_logic;
    dout: OUT std_logic_VECTOR(33 downto 0);
    empty: OUT std_logic;
    full: OUT std_logic);
end component;


component fec_header
    port (
    addra: IN std_logic_VECTOR(8 downto 0);
    addrb: IN std_logic_VECTOR(8 downto 0);
    clka: IN std_logic;
    clkb: IN std_logic;
    dinb: IN std_logic_VECTOR(31 downto 0);
    douta: OUT std_logic_VECTOR(31 downto 0);
    doutb: OUT std_logic_VECTOR(31 downto 0);
    ena: IN std_logic;
    enb: IN std_logic;
    web: IN std_logic);
end component;


component fec_storage
    port (
    addra: IN std_logic_VECTOR(8 downto 0);
    addrb: IN std_logic_VECTOR(8 downto 0);
    clka: IN std_logic;
    clkb: IN std_logic;
    dina: IN std_logic_VECTOR(31 downto 0);
    doutb: OUT std_logic_VECTOR(31 downto 0);
    enb: IN std_logic;
    wea: IN std_logic);
end component;


-- Signal definition
signal SequenceNumberOutbit : std_logic;
signal FifoInputEnable : std_logic;
signal FifoOutputEnable : std_logic;
signal FifoOutputData : std_logic_vector(33 downto 0);
signal FifoData : std_logic_vector(31 downto 0);
signal FifoEmpty : std_logic;
signal CounterReset : std_logic;
signal LastWord : std_logic;
signal FECPacketReady : std_logic;
signal DPRAMSelect : std_logic;
signal DPRAMWE : std_logic;
signal DPRAMAddress : std_logic_vector(9 downto 0);
signal DPRAMData : std_logic_vector(31 downto 0);
signal HeaderRamRE : std_logic;
signal HeaderRamAddress : std_logic_vector(8 downto 0);
signal HeaderRamData : std_logic_vector(31 downto 0);
signal FECRamWeA : std_logic;
signal FECRamAddressA : std_logic_vector(8 downto 0);
signal FECRamDataA : std_logic_vector(31 downto 0);
signal FECRamReB : std_logic;
signal FECRamAddressB : std_logic_vector(8 downto 0);
signal FECRamDataB : std_logic_vector(31 downto 0);
signal DataReadyFifo : std_logic;
signal FifoDataSequenceNumberDataReady : std_logic_vector(33 downto 0);
signal SequenceNumber1bitTmp : std_logic;

Begin

FifoData <= FifoOutputData(31 downto 0);
DataReadyFifo <= FifoOutputData(32);
```

```
SequenceNumber1bitTmp <= FifoOutputData(33);
FifoDataSequenceNumberDataReady <= SequenceNumberOutbit & DATAREADY & FIFODATAIN;

FEC1 : FECProcessing port map(
    reset                   => RESET,
    clk                     => CLK,
    FifoOutputData          => FifoData,
    FifoEmpty               => FifoEmpty,
    FifoOutputEnable        => FifoOutputEnable,
    HeaderRamRE             => HeaderRamRE,
    HeaderRamAddress        => HeaderRamAddress,
    HeaderRamData           => HeaderRamData,
    DPRAMSelect             => DPRAMSelect,
    DPRAMAddress            => DPRAMAddress,
    DPRAMData               => DPRAMData,
    DPRAMWE                 => DPRAMWE,
    FECRamWeA               => FECRamWeA,
    FECRamAddressA          => FECRamAddressA,
    FECRamDataA             => FECRamDataA,
    FECRamReB               => FECRamReB,
    FECRamAddressB          => FECRamAddressB,
    FECRamDataB             => FECRamDataB,
    SequenceNumber1bit      => SequenceNumber1bitTmp,
    TSMode                  => TSMODE,
    LastWord                => LastWord,
    FECPacketReady          => FECPacketReady,
    CounterReset            => CounterReset
);


FEC2 : FECLastWordCounter port map
(
    reset               => RESET,
    clk                 => CLK,
    TsMode              => TSMODE,
    DataReadyFromFifo   => DataReadyFifo,
    FifoOutputEnable    => FifoOutputEnable,
    CounterReset        => CounterReset,
    LastWord            => LastWord
);

FEC3 : FECPacketSupervising port map
(
    reset          => RESET,
    clk            => CLK,
    FECPacketReady => FECPacketReady,
    DPRAMSelect    => DPRAMSelect,
    BufferOverflow => BUFFEROVERFLOW,
    MemReady       => MEMREADY,
    MemSelect      => MEMSELECT,
    DSRCMemServiced=> DSRC_MEM_SERVICED,
    DSRCMemSelect  => DSRC_MEM_SELECT
);

FEC4 : FECInFifoInputEnable port map
(
    reset                 => RESET,
    clk                   => CLK,
    DataReady             => DATAREADY,
    BufferSelect          => BUFFERSELECT,
    SequenceNumber16bit   => SEQUENCE_NUMBER,
    DataValid             => DATAVALID,
    Grant                 => GRANT,
    SequenceNumberOutbit  => SequenceNumberOutbit,
    FifoInputEnable       => FifoInputEnable
);


FEC5 : fec_output port map
(
    addra     => DPRAMAddress,
    addrb     => PCI_DPRAM_ADDR,
    clka      => CLK,
    clkb      => CLK_EXT,
    dina      => DPRAMData,
    dinb      => PCI_DPRAM_DIN,
    doutb     => PCI_DPRAM_DOUT,
    enb       => PCI_DPRAM_EN,
    wea       => DPRAMWE,
```

```
    web       => PCI_DPRAM_WE
);

FEC6 : fec_input port map
(
    clk       => CLK,
    din       => FifoDataSequenceNumberDataReady,
    rd_en     => FifoOutputEnable,
    rst       => RESET, -- er det en reset?
    wr_en     => FifoInputEnable,
    dout      => FifoOutputData,
    empty     => FifoEmpty,
    full      => FIFOFULL -- Port and Port Map does not match
);


FEC7 : fec_header port map
(
    addra     => HeaderRamAddress,
    addrb     => PCI_HEADER_ADDR,
    clka      => CLK,
    clkb      => CLK_EXT,
    dinb      => PCI_HEADER_DIN,
    douta     => HeaderRamData,
    doutb     => PCI_HEADER_DOUT,
    ena       => HeaderRamRE,
    enb       => PCI_HEADER_EN,
    web       => PCI_HEADER_WE
);

FEC8 : fec_storage port map
(
    addra     => FECRamAddressA,
    addrb     => FECRamAddressB,
    clka      => CLK,
    clkb      => CLK,
    dina      => FECRamDataA,
    doutb     => FECRamDataB,
    enb       => FECRamReB,
    wea       => FECRamWeA
);

end Behavioral;
```

5.1.2 Processing entity

```
-- ********************************************************************************************
-- * Scientific Atlanta Denmark A/S
-- ********************************************************************************************
-- * Project      : HD Encoder D9054
-- *
-- * Used in      : FEC
-- *
-- * Description  : Processing unit for FEC
-- *
-- * Made by Maria Baltzer, Technical University of Denmark for Scientific Atlanta Denmark
-- ********************************************************************************************
--                         -----           -----
--                        |    |          |    |
--                        \/   |          \/   |
--     --------      --------       --------       --------
--    |       |     |       | ---> |       |      |       |
--    | Init  |---> | idle  |      | Fetch | ---> | Write |
--  |       |     |       | <--- |       |      |       |
--  --------      --------       --------       --------
--                    /\                           |
--                    |                           |
--                    -----------------------------|
--

Library IEEE;
Use     IEEE.STD_Logic_1164.all;
Use     IEEE.std_logic_arith.all;
use     IEEE.std_logic_unsigned.all;

Entity FECProcessing is

port
(
   reset            : in  std_logic;
   clk              : in  std_logic;
   FifoOutputData   : in  std_logic_vector(31 downto 0);
   FifoEmpty      : in  std_logic;
   FifoOutputEnable     : out std_logic;
   HeaderRamRE      : out std_logic;
   HeaderRamAddress : out std_logic_vector(8 downto 0);
   HeaderRamData    : in  std_logic_vector(31 downto 0);
   DPRAMSelect      : out std_logic;
   DPRAMAddress     : out std_logic_vector(9 downto 0);
   DPRAMData        : out std_logic_vector(31 downto 0);
   DPRAMWE          : out std_logic;
   FECRamWeA        : out std_logic;
   FECRamReA        : out std_logic;
   FECRamAddressA   : out std_logic_vector(8 downto 0);
   FECRamDataA      : out std_logic_vector(31 downto 0);
   FECRamWeB        : out std_logic;
   FECRamReB        : out std_logic;
   FECRamAddressB   : out std_logic_vector(8 downto 0);
   FECRamDataB      : in  std_logic_vector(31 downto 0);
   SequenceNumber1Bit: in  std_logic;
   TSMode           : in  std_logic;
   LastWord         : in  std_logic;
   FECPacketReady   : out std_logic;
   CounterReset     : in  std_logic
);
end FECProcessing;

architecture behavioral of FECProcessing is

constant PayloadSize : integer := 7;
constant FECSize : integer := 10;
--IP = 5 words, UDP = 2 words, RTP = 3 words and FEC = 4 words...
constant HeaderSize : integer := 14;

type state_t is (init, idle, fetch1, fetch2, fetch3, fetch4, fetch5, InitHeader,
                 ReadHeader, WriteHeader, write1, write2, write3);
signal state : state_t;
signal fifoData : std_logic_vector(31 downto 0);
```

```
signal fecData :  std_logic_vector(31 downto 0);
signal mode, PreviousMode : integer range 1 to 51;
signal OutputRamSelect : std_logic;
signal MemCount : integer range 0 to 512;
signal LastWordCounter : integer range 0 to 7; --counts how many last word signals have
been recieved
signal PayloadCounter : integer range 1 to 10;
signal WordCount : integer range 0 to 52;
signal HeaderCounter : integer range 0 to 20; -- skal måske være mindre...
signal SequenceNumber, tmpSequenceNumber : std_logic_vector(15 downto 0);
signal PacketReady : std_logic;
signal WaitForIPPacket : std_logic;
signal LastWordEdge : std_logic;

begin

  mode <= 51 when TsMode = '1' else 47;

DPRAMSelect <= OutputRamSelect;

ProcessData : process(reset, LastWord, FifoEmpty, clk)
variable initCount : integer range 0 to 3;
begin
  if reset = '1' then
    state <= init;
    fifoData <= (others => '0');
    fecData <= (others => '0');
    PreviousMode <= 47;
    OutputRamSelect <= '0';
    MemCount <= 0;
    LastWordCounter <= 0;
    PayloadCounter <= 1;
    WordCount <= 0;
    HeaderCounter<= 0;
    SequenceNumber <= (others => '0');
    PacketReady <= '0';
    FifoOutputEnable <= '0';
    HeaderRamRE  <= '0'; --aktiv høj eller lav?
    HeaderRamAddress  <= (others => '0');
    DPRAMAddress <= (others => '0');
    DPRAMData <= (others => '0');
    DPRAMWE <= '0';
    FECRamReA <= '0';
    FECRamWeA <= '0';
    FECRamAddressA <= (others => '0');
    FECRamDataA <= (others => '0');
    FECRamWeB <= '0';
    FECRamReB <= '0';
    FECRamAddressB <= (others => '0');
    LastWordEdge <= '0';
  elsif clk'EVENT and clk = '1' then
    if CounterReset = '1' or mode /= PreviousMode then
      state <= init;
    end if;
    PreviousMode <= mode;
    case state is
      -- gå eventuelt til denne state hvis der sker et mode-skift...
      when init =>
        WordCount <= 0;
        LastWordCounter <= 0;
        PayloadCounter <= 1;
        MemCount <= 0;
        OutputRamSelect <= '0';
        state <= idle;
        FifoOutputEnable <= '0';
        PacketReady <= '0';
      when idle =>
        LastWordEdge <= LastWord;
        if LastWordEdge = '0' and LastWord = '1' and LastWordCounter /= PayloadSize then
          LastWordCounter <= LastWordCounter + 1;
        elsif LastWordEdge = '0' and LastWord = '1' and LastWordCounter = PayloadSize and
              PayloadCounter /= FECSize then
          LastWordCounter <= 0;
          PayloadCounter <= PayloadCounter + 1;
        elsif LastWord = '1' and LastWordCounter = PayloadSize and
              PayloadCounter = FECSize then
          LastWordCounter <= 0;
          PayloadCounter <= 1;
        end if;
```

```
    if FifoEmpty = '1' then

      if LastWord = '1' and LastWordCounter = PayloadSize
           and PayloadCounter = FECSize then
        state <= initHeader;
      else
        state <= idle;
      end if;
    elsif FifoEmpty = '0' then
      if LastWord = '1' and LastWordCounter = PayloadSize
           and PayloadCounter = FECSize then
        state <= initHeader;
      else
        state <= fetch1;
      end if;
    end if;
    if WordCount = mode then
      WordCount <= 0;
    end if;

  when fetch1 =>
      state <= fetch2;

  when fetch2 =>
    FECRamReB <= '1'; -- aktiv høj eller lav
    FECRamAddressB <= conv_std_logic_vector(WordCount, 9);
    FifoOutputEnable <= '1';
    state <= fetch3;

  when fetch3 =>
    FifoOutputEnable <= '0';
    FECRamReB <= '0'; -- aktiv høj eller lav
    state <= fetch4;

  when fetch4 =>
    FECRamWeA <= '1'; -- aktiv høj eller lav
    FECRamAddressA <= conv_std_logic_vector(WordCount, 9);
    FECRamDataA <= FECRamDataB xor FifoOutputData;
    if WordCount >= 5 and WordCount <= 20 then
      SequenceNumber(WordCount-5) <= SequenceNumber1Bit;
    end if;
    WordCount <= WordCount + 1;
    state <= fetch5;

  when fetch5 =>
    FECRamWeA <= '0';
    state <= idle;

  when InitHeader =>
    HeaderRamRE <= '1';
    HeaderRamAddress <= conv_std_logic_vector(HeaderCounter, 9);
    OutputRamSelect <= not OutputRamSelect;
    state <= ReadHeader;

  when ReadHeader =>
    if HeaderCounter = HeaderSize then
      HeaderRamRE <= '0';
      DPRAMWE <= '0';
      state <= write1;
    else
      HeaderCounter <= HeaderCounter + 1;
      state <= WriteHeader;
      HeaderRamAddress <= conv_std_logic_vector(HeaderCounter, 9);
    end if;
  when WriteHeader =>
    DPRAMAddress <= OutputRamSelect & conv_std_logic_vector(HeaderCounter,9);
    DPRAMData <= HeaderRamData;
    DPRAMWE <= '1';
    state <= ReadHeader;
  when write1 =>
    FECRamReB <= '1'; -- aktiv høj eller lav
    FECRamAddressB <= conv_std_logic_vector((PayloadCounter * mode) + WordCount, 9);
    state <= write2;
  when write2 =>
    if WordCount = mode then
      PayloadCounter <= PayloadCounter + 1;
      WordCount <= 0;
      if PayloadCounter = PayloadSize then
```

```
            FECRamReB <= '0';
            DPRAMWE <= '0';
            PayloadCounter <= 1;
            PacketReady <= '1';
            state <= idle;
          end if;
        else

          FECRamAddressB <= conv_std_logic_vector((PayloadCounter * mode) + WordCount, 9);

          WordCount <= WordCount + 1;
          state <= write3;
        end if;
      when write3 =>
        --write data to output ram
        DPRAMAddress <= OutputRamSelect &
                conv_std_logic_vector(HeaderSize + (PayloadCounter * mode) + WordCount,9);
        DPRAMData <= FECRamDataB;
        DPRAMWE <= '1';
        state <= write2;
    end case;
  if PacketReady = '1'then
    PacketReady <= '0';
  end if;

  end if;

end process ProcessData;

SignalPacketReady : process(clk)
begin
  if reset = '1' then
    WaitForIPPacket <= '0';
    FECPacketReady <= '0';
    tmpSequenceNumber <= (others => '0');
  elsif clk'EVENT and clk = '1' then

    if PacketReady = '1' then
      WaitForIPPacket <= '1';
      tmpSequenceNumber <= SequenceNumber;
    elsif WaitForIPPacket = '1' and WordCount > 18
                and (SequenceNumber - tmpSequenceNumber) > 10 then
      FECPacketReady <= '1';
      WaitForIPPacket <= '0';
    else
      FECPacketReady <= '0';
    end if;
  end if;

end process SignalPacketReady;

end behavioral;
```

5.1.3 Counter

```
-- ********************************************************************************************
-- * Scientific Atlanta Denmark A/S
-- ********************************************************************************************
-- * Project      : HD Encoder D9054
-- *
-- * Used in      : FEC
-- *
-- * Description  : TS packet counter for the FEC system
-- *
-- * Made by Maria Baltzer, Technical University of Denmark for Scientific Atlanta Denmark
-- ********************************************************************************************
Library IEEE;
Use     IEEE.STD_Logic_1164.all;
Use     IEEE.std_logic_arith.all;
use     IEEE.std_logic_unsigned.all;

Entity FECLastWordCounter is

port
(
   reset          : in std_logic;
   clk            : in std_logic;
   TsMode         : in std_logic; -- 0 = 188, 1 = 204 TS packets
   DataReadyFromFifo    : in std_logic;
   FifoOutputEnable   : in std_logic;
   CounterReset   : out std_logic;
   LastWord       : out std_logic
);
end FECLastWordCounter;

architecture behavioral of FECLastWordCounter is

signal mode, LastWordCount : integer range 1 to 51;
signal DataReadyFromFifoOld : std_logic;

begin

  mode <= 51 when TsMode = '1' else 47;

count : process(clk, TsMode, DataReadyFromFifo, FifoOutputEnable, reset)
begin
  if reset = '1' then
    CounterReset <= '0';
    LastWord <= '0';
    LastWordCount <= 1;
    DataReadyFromFifoOld <= '0';
  elsif clk'EVENT and clk = '1' then
    if FifoOutputEnable = '1' then
      if DataReadyFromFifoOld = '0' and DataReadyFromFifo = '1'
               and LastWordCount /= mode then
        LastWord <= '0';
        LastWordCount <= 1;
        CounterReset <= '1';
      elsif LastWordCount = mode then
        LastWord <= '1';
        LastWordCount <= 1;
        CounterReset <= '0';
      else
        LastWord <= '0';
        LastWordCount <= LastWordCount + 1;
        CounterReset <= '0';
      end if;
      if DataReadyFromFifo = '0' then
        DataReadyFromFifoOld <= '0';
      else
        DataReadyFromFifoOld <= '1';
      end if;

    else
      CounterReset <= '0';
      if DataReadyFromFifo = '0' then
        DataReadyFromFifoOld <= '0';
```

```
      else
        DataReadyFromFifoOld <= '1';
      end if;

    end if;
  end if;

end process count;

end behavioral;
```

5.1.4 Supervising entity

```
-- ********************************************************************************************
-- * Scientific Atlanta Denmark A/S
-- ********************************************************************************************
-- * Project      : HD Encoder D9054
-- *
-- * Used in      : FEC
-- *
-- * Description  : Supervising of the output memory in the FEC system
-- *
-- * Made by Maria Baltzer, Technical University of Denmark for Scientific Atlanta Denmark
-- ********************************************************************************************
Library IEEE;
Use     IEEE.STD_Logic_1164.all;
Use     IEEE.std_logic_arith.all;
use     IEEE.std_logic_unsigned.all;

Entity FECPacketSupervising is

port
(
   reset          : in std_logic;
   clk            : in  std_logic;
   FECPacketReady : in  std_logic;
   DPRAMSelect    : in  std_logic;
   BufferOverflow : out std_logic;
   MemReady       : out std_logic; -- næste pakke er klar i ram'en
   MemSelect      : out std_logic;
   DSRCMemServiced: in  std_logic; --flag der fortæller at bufferen er serviceret.
   DSRCMemSelect  : in  std_logic
);
end FECPacketSupervising;

architecture behavioral of FECPacketSupervising is

signal buffer1, buffer2 : std_logic;

begin

MemSelect <= DPRAMSelect;
MemReady <= FECPacketReady;

Supervising : process(clk, reset)
begin
  if reset = '1' then
    BufferOverflow <= '0';
    buffer1 <= '0';
    buffer2 <= '0';
  elsif clk'EVENT and clk = '1' then
    if FECPacketReady = '1' then
      if DPRAMSelect = '0' and buffer1 = '1' then
        BufferOverflow <= '1';
      elsif DPRAMSelect = '1' and buffer2 = '1' then
        BufferOverflow <= '1';
      else
        BufferOverflow <= '0';
      end if;

    elsif DSRCMemServiced = '1' then
      if DSRCMemSelect = '0' then
        buffer1 <= '0';
      elsif DSRCMemSelect = '1' then
        buffer2 <= '0';
      end if;

    end if;
  end if;
end process Supervising;


end behavioral;
```

5.1.5 FIFO enable signalling

```
-- ********************************************************************************************
-- * Scientific Atlanta Denmark A/S
-- ********************************************************************************************
-- * Project      : HD Encoder D9054
-- *
-- * Used in      : FEC
-- *
-- * Description  : FIFO enable signalling for the input fifo of the FEC system
-- *
-- * Made by Maria Baltzer, Technical University of Denmark for Scientific Atlanta Denmark
-- ********************************************************************************************
Library IEEE;
Use     IEEE.STD_Logic_1164.all;
Use     IEEE.std_logic_arith.all;
use     IEEE.std_logic_unsigned.all;

Entity FECInFifoInputEnable is

port
(
   reset                : in std_logic;
   clk                  : in  std_logic;
   DataReady            : in  std_logic;
   BufferSelect         : in  std_logic_vector(3 downto 0);
   SequenceNumber16bit  : in  std_logic_vector(15 downto 0);
   DataValid            : in  std_logic;
   Grant                : in  std_logic;
   SequenceNumberOutbit : out std_logic;
   FifoInputEnable      : out std_logic

);
end FECInFifoInputEnable;

architecture behavioral of FECInFifoInputEnable is

constant channel : std_logic_vector(3 downto 0) := "0111"; -- channel 7
type state_t is (one, two, three);
signal state : state_t;
signal WaitForDataValid : std_logic;
signal GrantFlag : std_logic;
signal SequenceNumber : std_logic_vector(15 downto 0);
signal k : integer range 0 to 15;
signal i : integer range 0 to 60;

begin

FifoInputEnable <= DataValid when Grant = '1' and WaitForDataValid = '1' else '0';

InFifoInputEnableInput : process(clk, reset)
begin
  if reset = '1' then
    SequenceNumberOutbit <= '0';
    GrantFlag <= '0';
    state <= one;
    WaitForDataValid <= '0';
    SequenceNumber <= (others => '0');
    k <= 0;
    i <= 0;
  elsif clk'EVENT and clk = '1' then
    case state is
      when one =>
        if DataReady = '1' then
          state <= two;
        else
          state <= one;
        end if;
     when two =>
        if BufferSelect = channel then
          WaitForDataValid <= '1';
          state <= three;
        else
          state <= one;
```

```
        end if;
      when three =>
        if Grant = '1' then
          GrantFlag <= '1';
        end if;
        if GrantFlag = '1' and Grant = '0' then
          GrantFlag <= '0';
          WaitForDataValid <= '0';
          state <= one;
        else
          state <= three;
        end if;

    end case;
    if Grant = '1' then
      if i = 59 then
        i <= 1;
      elsif i <= 3 then
        SequenceNumber <= SequenceNumber16bit;
        SequenceNumberOutbit <= '0';
        k <= 0;
        i <= i + 1;
      elsif i > 3 and i < 20 then
        if k = 15 then
          k <= 0;
        else
          k <= k + 1;
        end if;
        SequenceNumberOutbit <= SequenceNumber(k);
        i <= i + 1;
      else
        SequenceNumberOutbit <= '0';
        i <= i + 1;

      end if;
    elsif Grant = '0' then
        i <= 0;


    end if;

  end if;

end process InFifoInputEnableInput;

end behavioral;
```

5.1.6 Test Bench

```
-- ********************************************************************************************
-- * Scientific Atlanta Denmark A/S
-- ********************************************************************************************
-- * Project      : HD Encoder D9054
-- *
-- * Used in      : FEC
-- *
-- * Description  : Test bench for FEC system
-- *
-- * Made by Maria Baltzer, Technical University of Denmark for Scientific Atlanta Denmark
-- ********************************************************************************************

Library IEEE;
Use     IEEE.STD_Logic_1164.all;
Use     IEEE.std_logic_arith.all;
use     IEEE.std_logic_unsigned.all;

entity TB_FECTOP is
end TB_FECTOP;

architecture BEH of TB_FECTOP is

   component FECTOP
      port(RESET               : in std_logic ;
           CLK                 : in std_logic ;
           CLK_EXT             : in std_logic ;
           TSMODE              : in std_logic ;
           DSRC_MEM_SERVICED   : in std_logic ;
           DSRC_MEM_SELECT     : in std_logic ;
           DATAREADY           : in std_logic ;
           BUFFERSELECT        : in std_logic_vector ( 3 downto 0 );
           SEQUENCE_NUMBER     : in std_logic_vector ( 15 downto 0 );
           DATAVALID           : in std_logic ;
           GRANT               : in std_logic ;
           BUFFEROVERFLOW      : out std_logic ;
           MEMREADY            : out std_logic ;
           MEMSELECT           : out std_logic ;
           PCI_DPRAM_ADDR      : in std_logic_vector ( 9 downto 0 );
           PCI_DPRAM_DIN       : in std_logic_vector ( 31 downto 0 );
           PCI_DPRAM_DOUT      : out std_logic_vector ( 31 downto 0 );
           PCI_DPRAM_EN        : in std_logic ;
           PCI_DPRAM_WE        : in std_logic ;
           PCI_HEADER_ADDR     : in std_logic_vector ( 8 downto 0 );
           PCI_HEADER_DIN      : in std_logic_vector ( 31 downto 0 );
           PCI_HEADER_DOUT     : out std_logic_vector ( 31 downto 0 );
           PCI_HEADER_EN       : in std_logic ;
           PCI_HEADER_WE       : in std_logic ;
           FIFODATAIN          : in std_logic_vector ( 31 downto 0 );
           FIFOFULL            : out std_logic );

   end component;


   constant PERIOD : time := 13.47 ns; --74.25 MHz Clock

   signal RESET               : std_logic ;
   signal CLK                 : std_logic  := '0';
   signal CLK_EXT             : std_logic  := '0';
   signal TSMODE              : std_logic ;
   signal DSRC_MEM_SERVICED   : std_logic ;
   signal DSRC_MEM_SELECT     : std_logic ;
   signal DATAREADY           : std_logic ;
   signal BUFFERSELECT        : std_logic_vector ( 3 downto 0 );
   signal SEQUENCE_NUMBER     : std_logic_vector ( 15 downto 0 );
   signal DATAVALID           : std_logic ;
   signal GRANT               : std_logic ;
   signal BUFFEROVERFLOW      : std_logic ;
   signal MEMREADY            : std_logic ;
   signal MEMSELECT           : std_logic ;
   signal PCI_DPRAM_ADDR      : std_logic_vector ( 9 downto 0 );
   signal PCI_DPRAM_DIN       : std_logic_vector ( 31 downto 0 );
   signal PCI_DPRAM_DOUT      : std_logic_vector ( 31 downto 0 );
```

```
   signal PCI_DPRAM_EN        : std_logic ;
   signal PCI_DPRAM_WE        : std_logic ;
   signal PCI_HEADER_ADDR     : std_logic_vector ( 8 downto 0 );
   signal PCI_HEADER_DIN      : std_logic_vector ( 31 downto 0 );
   signal PCI_HEADER_DOUT     : std_logic_vector ( 31 downto 0 );
   signal PCI_HEADER_EN       : std_logic ;
   signal PCI_HEADER_WE       : std_logic ;
   signal FIFODATAIN          : std_logic_vector ( 31 downto 0 );
   signal FIFOFULL            : std_logic ;


   signal memselectTmp : std_logic;

begin

   DUT : FECTOP
      port map(RESET               => RESET,
               CLK                 => CLK,
               CLK_EXT             => CLK_EXT,
               TSMODE              => TSMODE,
               DSRC_MEM_SERVICED   => DSRC_MEM_SERVICED,
               DSRC_MEM_SELECT     => DSRC_MEM_SELECT,
               DATAREADY           => DATAREADY,
               BUFFERSELECT        => BUFFERSELECT,
               SEQUENCE_NUMBER     => SEQUENCE_NUMBER,
               DATAVALID           => DATAVALID,
               GRANT               => GRANT,
               BUFFEROVERFLOW      => BUFFEROVERFLOW,
               MEMREADY            => MEMREADY,
               MEMSELECT           => MEMSELECT,
               PCI_DPRAM_ADDR      => PCI_DPRAM_ADDR,
               PCI_DPRAM_DIN       => PCI_DPRAM_DIN,
               PCI_DPRAM_DOUT      => PCI_DPRAM_DOUT,
               PCI_DPRAM_EN        => PCI_DPRAM_EN,
               PCI_DPRAM_WE        => PCI_DPRAM_WE,
               PCI_HEADER_ADDR     => PCI_HEADER_ADDR,
               PCI_HEADER_DIN      => PCI_HEADER_DIN,
               PCI_HEADER_DOUT     => PCI_HEADER_DOUT,
               PCI_HEADER_EN       => PCI_HEADER_EN,
               PCI_HEADER_WE       => PCI_HEADER_WE,
               FIFODATAIN          => FIFODATAIN,
               FIFOFULL            => FIFOFULL);

   CLK <= not CLK after PERIOD/2;
   CLK_EXT <= not CLK_EXT after PERIOD/2;

FifoSide : process
begin
  wait until RESET = '0';

  wait for 3*PERIOD;
  for h in 34952 to 100000 loop
    for i in 0 to 6 loop
      DATAREADY           <= '0';
      GRANT               <= '0';
      DATAVALID           <= '0';
      wait until CLK = '1';
      DATAREADY           <= '1';
      FIFODATAIN          <= conv_std_logic_vector(2, 32);
      BUFFERSELECT        <= "0111";
      wait for 3*PERIOD;
      GRANT               <= '1';
      DATAVALID           <= '1';
      SEQUENCE_NUMBER     <= conv_std_logic_vector(h, 16);

      for j in 3 to 48 loop
        wait until CLK = '1';
        FIFODATAIN          <= conv_std_logic_vector(j, 32);
        if j = 4 then
          DATAREADY <= '0';
        end if;
      end loop;
      wait for PERIOD;
      DATAVALID           <= '0';
      GRANT               <= '0';
      wait for 930*PERIOD;
    end loop;
```

```
  end loop;

end process FifoSide;

DPRAMSide : process
begin

  wait until MEMREADY = '1';
  memselectTmp <= MEMSELECT;

  wait for 5000 ns;
  wait until CLK = '1';
  DSRC_MEM_SERVICED   <= '1';
  DSRC_MEM_SELECT     <= memselectTmp;
  wait until CLK = '1';
  DSRC_MEM_SERVICED   <= '0';

end process DPRAMSide;

DPRAM : process
begin

  wait until MEMREADY = '1';
  for i in 0 to 343 loop
    wait until CLK = '1';
    PCI_DPRAM_ADDR      <= MEMSELECT & conv_std_logic_vector(i, 9);
    PCI_DPRAM_DIN       <= (others => '0');
    PCI_DPRAM_EN        <= '1';
    PCI_DPRAM_WE        <= '0';
  end loop;
end process DPRAM;


STIMULI : process
begin
  --Reset
  RESET               <= '1', '0' after 2*PERIOD;

  -- TSMode = 0 - 188 bytes
  TSMODE              <= '0';
  wait;
end process STIMULI;

end BEH;

configuration CFG_TB_FECTOP of TB_FECTOP is
   for BEH
   end for;
end CFG_TB_FECTOP;
```

6 Appendix

6.1 Dual port block memory datasheet from Core Generator

XILINX® LogiCORE™

Dual-Port Block Memory v5.0

DS235 (v0.1) November 1, 2002 **Product Specification**

Features

- Drop-in module for Virtex™, Virtex-E, Virtex-II, Virtex-II Pro™, Spartan™-II, Spartan-IIE, and Spartan-3 FPGAs
- Supports all three Virtex-II write mode options: Read-After-Write, Read-Before-Write, and No-Read-On-Write (Available only for Virtex-II and Spartan-3 implementation)
- Supports ROM and RAM functions
- Supports data widths from 1 to 256 bits
- Supports memory depths from 2 to 1M words depending on architecture selected
- Supports ROM functions, enabling simultaneous read operations from the same location
- Supports RAM functions, enabling simultaneous write operations to separate locations and simultaneous read operations from the same location
- The ports are completely independent of each other
- Supports asymmetric A and B port configurations
- Supports cores designed for area optimization or using a single SelectRAM+ or SelectRAM-II primitive
- Supports different pin polarities for control signals: clock, enable, write enable and output initialization pins
- Incorporates Xilinx Smart-IP™ technology for utmost parameterization and optimum implementation
- Available in the Xilinx CORE Generator™ System v5.1i and later

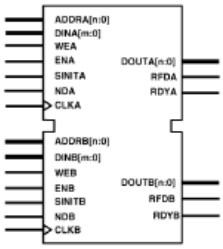


Figure 1: **Core Schematic Symbol**

Functional Description

The Dual-Port Block Memory module for Spartan-II and Virtex is composed of single or multiple 4Kb blocks called Select-RAM+™. The Dual-Port Block Memory module for Virtex-II and Spartan-3, on the other hand, is composed of single or multiple Virtex-II 18Kb blocks (SelectRAM-II™) enabling deeper and/or wider memory implementations. Both the SelectRAM+ and SelectRAM-II memories are True Dual-Port™ RAM, offering fast, discrete, and large blocks of memory in the Spartan-II and Virtex device families. Since Spartan-II and Virtex both use the 4Kb SeletectRAM+ blocks, any particular reference to a Virtex implementation also applies to a Spartan-II, Virtex-E, Virtex-II Pro, or Spartan-IIE implementation.

A memory module has two independent ports that enable shared access to a single memory space and which are generated based on user-defined width and depth. Both ports are functionally identical, with each port providing read and write access to the memory. Simultaneous reads from the same memory location may occur, but all other simultaneous, reading-from, and writing-to the same memory location will result in correct data being written into the memory, but invalid data being read.

The memory's Port A and Port B are configured to support user-defined data input and address widths. When both ports are disabled (ENA and ENB inactive) the memory contents and output ports remain unaltered. When either

port is enabled (ENA or ENB asserted) all memory operations occur on the active edge of the clock input.

During a write operation (WEA or WEB asserted), the data presented at the port's data input is stored in memory at the location selected by the port's address input. During this operation, the data output port behaves differently for the Spartan-II/Virtex and Virtex-II architectures.

The data output port of the Virtex-II/Spartan-3 implementation is dependent on a chosen "write mode" option. Three "write mode" options are supported by the Virtex-II and Spartan-3 architectures. Each of these options determines the behavior of the corresponding data output port when a write operation occurs.

The Spartan-II/Virtex implementation, on the other hand, supports a single "write mode" option: Read-After-Write. This "write mode" causes the data being written to the addressed memory location to be transferred to the data output port when a write operation occurs.

During a read operation, the memory contents at the location selected by the address will appear at the module's output. When Synchronous Initialization (SINITA or SINITB) is active, the module's registered outputs are synchronously reset to zero for Spartan-II/Virtex and to a user-defined value for Virtex-II. The Synchronous Initialization command has no effect on the contents of the memory or write operations.

The enable, write enable, and synchronous initialization can also be specified as active high or active low.

For additional information on the Spartan-II, Virtex, Virtex-II, and Spartan-3 BlockRAM, refer to the respective *Spartan-II, Virtex, Virtex-II, or Spartan-3 Databook*, available at the website:
http://www.xilinx.com/partinfo/databook.htm.

Pinout

Port names for the core module are shown in Figure 1 and defined in Table 1. The inclusion of some ports on the module is optional; exclusion of these ports will alter the function of the module. The optional ports are marked in Table 1 and described in more detail below.

Clock - CLK[AIB]

Each port is fully synchronous with independent clock pins. All port input pins have setup time referenced to the active edge of their corresponding CLK pin. The data bus has a clock-to-out time referenced to the CLK pin.

By default, the block memory operates synchronously to the rising edge of the clock. Users, however, have the option to perform all memory operations on the rising or the falling edge of the clock. Performing the memory operation on the falling edge of the clock will not use any extra resources.

Table 1: **Core Signal Pinout**

Name	Direction	Description
DIN[AIB]<n:0> (Optional)	Input	**Data Input**: data to be written into memory via Port [AIB].
ADDR[AIB]<m:0>	Input	**Address**: the memory location to which data will be written or read via Port [AIB].
WE[AIB] (Optional)	Input	**Write Enable**: control signal used to allow transfer of input data into memory via Port [AIB].
EN[AIB] (Optional)	Input	**Enable**: control signal used to enable memory accesses via read and write operations from Port [AIB].
SINIT[AIB] (Optional)	Input	**Synchronous Initialization**: control signal used to force the module's outputs to a predefined state.
CLK[AIB]	Input	**Clock**: clock input, all memory access is synchronous with the clock input.
ND[AIB] (Optional)	Input	**New Data Port A**: indicates that there is a new and valid address on Port ADDR[AIB] (Active High).
DOUT[AIB]<n:0> (Optional)	Output	**Data Output**: synchronous output of memory.
RFD[AIB] (Optional)	Output	**Ready for Data**: indicates that the memory is ready to accept new data (Active High).
RDY[AIB] (Optional)	Output	**Output Ready**: indicates valid data on port DOUT [AIB] (Active High).

Enable - EN[AIB]

The enable pin affects the read, write, and SINIT functionality of the port. Ports with an inactive enable pin keep the output pins in the previous state and do not write data to the memory locations.

By default, the enable pin is active high. Users, however, have the option to configure the enable pin active high or active low. Configuring the enable pin active low will not use extra resources.

Write Enable - WE[AIB]

Activating the write enable pin allows the port to write to the memory locations. When active, the contents of the DIN bus is written to memory at the address pointed to by the ADDR bus. The output latches are loaded or not loaded according to the write configuration (write first, read first, no change). When inactive, a read operation occurs, and the contents of the memory locations referenced by the address bus reflect on the DOUT bus, regardless of the write mode selected.

By default, the write enable pin is active high. Users, however, have the option to configure the write enable pin active high or active low. Configuring the write enable pin active low will not use extra resources.

Synchronous Initialization - SINIT[AIB]

The SINIT pin forces the data output ports to a SINIT value. For the Spartan-II/Virtex implementation, the SINIT value is zero, and for the Virtex-II implementation the SINIT value is user-defined. The data output ports are each synchronously asserted to their respective SINIT value. This operation does not affect memory locations and does not disturb write operations on the other port. If the core is configured with an enable pin, the SINIT function is active only when the enable port is active.

By default the SINIT pin is active high. Users, however, have the option to configure the SINIT pin active high or active low. Configuring the write enable pin active low will not use extra resources.

Address Bus - ADDR[AIB]<m:0>

The address bus selects the memory location that will be accessed during a read or write operation.

Data-In Bus - DIN[AIB]<n:0>

The DIN buses provide the new data value to be written into the memory. Data input and output signals are always busses; that is, in a 1-bit width configuration, the data input signal is DIN[0] and the data output signal is DOUT[0].

Data-Out Bus - DOUT[AIB]<n:0>

The DOUT buses reflect the contents of memory locations referenced by the address bus during a read operation.

During a write operation of a Spartan-II/Virtex memory (write first configuration), the DOUT buses reflect the DIN buses.

During a write operation of a Virtex-II/Spartan-3 memory (write first or read first configuration), the DOUT buses reflect either the DIN buses or the stored value before write. During a write operation of a Virtex-II/Spartan-3 memory in no change mode, DOUT buses are not affected.

New Data - ND[AIB]

ND indicates that there is a new and valid address on ADDR[AIB] port. It affects only the RDY port.

Ready for Data - RFD[AIB]

RFD indicates that the memory is ready to accept new data. RFD[AIB] is always true, except when EN[AIB] is inactive.

Output Ready (valid) - RDY[AIB]

Indicates valid output on port DOUT[AIB] relative to when ND is asserted. RDY[AIB] will lag ND[AIB] by the latency of the block memory.

CORE Generator Parameters

The parameters in the main parameterization window are as follows:

- **Component Name**: Enter a name for the output files generated for this module (up to 256 characters).
- **Memory Size - Port A**:
  - **Width A**: Select the data bit width. The width can be between 1 and 256.
  - **Depth A**: Select the number of words in memory. The value range is 2 to 1,048,576 (1M) depending on the architecture selected. Available depths will vary depending on the width entered for Port A; the absolute maximum number of words is 256K for the Spartan-II/Virtex architecture and 1M for the Virtex-II/Spartan-3 architecture. Be aware that cores should not exceed the number of Block RAM primitives in the targeted device.
- **Memory Size - Port B**:
  - **Width B**: Select the data bit width. Available widths will vary depending on the width entered for port A. For the Spartan-II/Virtex architecture, the available widths could be 1, 2, 4, 8 and 16 times larger than the width entered for port A. The available widths for the Virtex-II architecture could be 1, 2, 4, 8,16 and 32 times larger than the width entered for port A.
  - **Depth B**: Reports the depth of Port B. This value is calculated such that Port A and Port B have the same memory size. For the Spartan-II/Virtex architecture, the depth could be 1, 2, 4, 8 and 16 times smaller than the depth entered for port A.

The depth for the Virtex-II architecture could be 1, 2, 4, 8,16 and 32 times smaller than the depth entered for port A. The minimum depth for Port B is 2.

- **Port A Options:**
  - **Configuration:**
    - **Read and Write:** Configures Port A to have DINA and DOUTA ports allowing read and write access to the memory.
    - **Write Only:** Configures Port A to have a DINA port enabling this port to be used only for write access. Note that only one port can be configured to be Write Only.
    - **Read Only:** Configures Port A to have a DOUTA port enabling this port to be used for read only access.
  - **Write Mode:** Select one for Virtex-II architecture. The default is Read-After-Write. The Spartan-II/Virtex architecture supports only Read-After-Write.
    - **Read-after-Write:** (Virtex-II/Spartan-3, Virtex/Spartan-II)
    - (1) No Inputs or Outputs Registered: The input data is transferred onto the DOUTA port on the active clock edge immediately following the assertion of the WEA input.
    - (2) With Inputs Registered Only: The input data is transferred onto the DOUTA port on the second active clock edge immediately following the assertion of the WEA input.
    - (3) With Outputs Registered Only: The input data is transferred onto the DOUTA port on the second active clock edge immediately following the assertion of the WEA input.
    - (4)With Inputs and Outputs Registered: The input data is transferred onto the DOUTA port on the third active clock edge immediately following the assertion of the WEA input.
    - **Read-before-Write:** (Virtex-II/Spartan-3 Only)
    - 1) No Inputs or Outputs Registered: The current data in the addressed memory location is transferred onto the DOUTA port on the active clock edge immediately following the assertion of the WEA input.
    - (2) With Inputs Registered Only: The current data in the addressed memory location is transferred onto the DOUTA port on the second active clock edge immediately following the assertion of the WEA input.
    - (3) With Outputs Registered Only: The current data in the addressed memory location is transferred onto the DOUTA port on the second active clock edge immediately following the assertion of the WEA input.
    - (4)With Inputs and Outputs Registered: The current data in the addressed memory location is transferred onto the DOUTA port on the third active clock edge immediately following the assertion of the WEA input.
    - **No-Read-on-Write:** (Virtex-II/Spartan-3 Only)
    - A read operation is not performed when WEA is asserted. The DOUTA port will contain the contents of the last read memory location.
- **Port B Options:**
  - **Configuration:**
    - **Read and Write:** Configures Port B to have DINB and DOUTB ports allowing read and write access to the memory.
    - **Write Only:** Configures Port B to have a DINB port enabling this port to be used only for write access. Note that only one port can be configured to be Write Only.
    - **Read Only:** Configures Port B to have a DOUTB port enabling this port to be used only for read access.
  - **Write Mode:** Select one for Virtex-II/Spartan-3 architecture. The default is Read-After-Write. The Spartan-II/Virtex architecture supports only Read-After-Write.
    - **Read-after-Write:**
    - (1) No Inputs or Outputs Registered: The input data is transferred onto the DOUTB port on the active clock edge immediately following the assertion of the WEB input.
    - (2) With Inputs Registered Only: The input data is transferred onto the DOUTB port on the second active clock edge immediately following the assertion of the WEB input.
    - (3) With Outputs Registered: The input data is transferred onto the DOUTB port on the second active clock edge immediately following the assertion of the WEB input.
    - (4)With Inputs and Outputs Registered: The input data is transferred onto the DOUTB port on the third active clock edge immediately following the assertion of the WEB input.
    - **Read-before-Write:**
    - 1) No Inputs or Outputs Registered: The current data in the addressed memory location is transferred onto the DOUTB port on the active clock edge immediately following the assertion of the WEB input.
    - (2) With Inputs Registered: The current data in the addressed memory location is transferred onto the DOUTB port on the second active clock edge immediately following the assertion of the WEB input.
    - (3) With Outputs Registered: The current data in the addressed memory location is transferred onto the DOUTB port on the second active clock edge immediately following the assertion of the WEB input.
    - (4)With Inputs and Outputs Registered: The current data in the addressed memory location is transferred onto the DOUTB port on the third

active clock edge immediately following the assertion of the WEB input.
- **No-Read-on-Write**:
- A read operation is not performed when WEB is asserted. The DOUTB port will contain the contents of the last read memory location.

When the next button is clicked, the second parameterization window is displayed. The parameters are as follows:

- **Port A Design Options**:
  - Optional Pins:
    - **Enable Pin**: Check the box to include the enable ENA port on the module; uncheck the box to remove it.
    - **Handshaking Pins**: Check the box to include the following ports; uncheck the box to remove them.
    - **ND** [New Data]: Signals a new and valid memory address whenever active. This port has no effect on the memory read and write operations. ND is valid only when RFD is active.
    - **RFD** [Ready For Data]: Indicates that the memory can accept new addresses. Always active when the memory is enabled.
    - **RDY** [Output is Ready]: Indicates to the user that the data on the output is valid. RDY will lag ND by the latency of the module.
    - **Register Options**:
    - Register Inputs: Check this box to register ports DIN, ADDR, and WE prior to accessing block memory. See Figure 4.
    - Output Register Options:
    - **Additional Output Pipe Stages**: Select "1" to enable an additional register on the output of the memory; select "0" to disable an additional register on the output of the memory. See Figure 4.
    - **SINIT Pin**: Check box to add the synchronous port SINIT to the memory.
  - **Pin Polarity**: The option enables the user to configure the polarities of the Port A control signals if the signals exist.
    - **Active Clock Edge**: Select whether the memory operation occurs on the rising edge or falling edge of the clock.
    - **Enable Pin:** Select whether the enable pin is active high or active low.
    - **Write Enable Pin:** Select whether the write enable pin in active high or active low.
    - **Initialization Pin:** Select whether the initialization pin is active high or active low.

When the next button is clicked, the third parameterization window is displayed. The parameters are as follows:

- **Port B Design Options**:
  - Optional Pins:
    - **Enable Pin:** Check the box to include the enable ENB port on the module; uncheck the box to remove it.
    - **Handshaking Pins:** Check the box to include the following ports; uncheck the box to remove them.
    - **ND** [New Data]: Signals a new and valid memory address whenever active. This port has no effect on the memory read and write operations. ND is valid only when RFD is active.
    - **RFD** [Ready For Data]: Indicates that the memory can accept new addresses. Always active when the memory is enabled.
    - **RDY** [Output is Ready]: Indicates to the user that the data on the output is valid. RDY will lag ND by the latency of the module.
  - **Register Options**:
    - **Register Inputs**: Check this box to register ports DIN, ADDR, and WE prior to accessing block memory. See Figure 4.
  - **Output Register Options**:
    - **Additional Output Pipe Stages**: Select "1" to enable an additional register on the output of the memory; select "0" to disable an additional register on the output of the memory. See Figure 4.
    - **SINIT Pin**: Check box to add the synchronous port SINIT to the memory.
  - **Pin Polarity**: The option enables the user to configure the polarities of the Port B control signals if the signals exist.
    - **Active Clock Edge**: Select whether the memory operation occurs on the rising edge or falling edge of the clock.
    - **Enable Pin**: Select whether the enable pin is active high or active low.
    - **Write Enable Pin**: Select whether the write enable pin in active high or active low.
    - **Initialization Pin**: Select whether the initialization pin is active high or active low.
- **Primitive selection**: Choose whether the core is optimized for area or created using a single 4kb SelectRAM+ or 16kb SelectRAM-II block or primitive.
  - **Select primitive**: Choose which block or primitive is used to create the core.
    - The primitives for the Virtex/Virtex-E/Spartan-II architectures are: 4kx1, 2kx2, 1kx4, 512x8 and 256x16.
    - The primitives for the Virtex-II architecture are: 16kx1, 8kx2, 4kx4, 2kx9, 1kx18, and 512x36.

When the next button is clicked, the fourth parameterization window is displayed. The parameters are as follows:

- **Initial Contents**: Enter the parameter fields related to the data stored in the memory directly after device configuration. Note that these initial data must conform to the chosen Port A parameter fields.
  - **Global Init Value**: Enter the value to be stored in any memory location not specified by another means. When no values are entered, this field defaults to 0. Value must be in Hex. This value must be smaller than Port A's largest word.

- **Load Init File**: Select this if the initial contents of the memory are to be read from a *coe* file.
- **Load File**: Press this button to activate a browser window that lets the user pick a coefficient or *coe* file that contains the initial contents of the memory. This is an ASCII file with a ".coe" extension. For further information regarding the memory's initial contents, refer to the *Specifying Memory Contents* section.

• **Information Panel**: Lists the resulting configuration of the core.

- **Address Width A**: Displays the number of address bits required for Port A for this configuration.
- **Address Width B**: Displays the number of address bits required for Port B for this configuration.
- **Blocks Used**: Displays the number of BlockRAM primitives needed to implement this configuration. It is recommended to verify that the required number of blocks does not exceed those available in the targeted device.
- **Port A Read Pipeline Latency**: Displays the total latency of Port A from the point when a new address is presented to the memory to when it becomes a valid output. The total latency will be increased by one if the inputs are registered or if an additional output register is added. A latency of 1 is defined as follows: the read address is read in on the active edge of the clock and the resulting output is seen following that same active edge of the clock.
- **Port B Read Pipeline Latency**: Displays the total latency of Port B from the point when a new address is presented to the memory to when it becomes a valid output. The total latency will be increased by one if the inputs are registered or if an additional output register is added. A latency of 1 is defined as follows: the read address is read in on the active edge of the clock and the resulting output is seen following that same active edge of the clock.

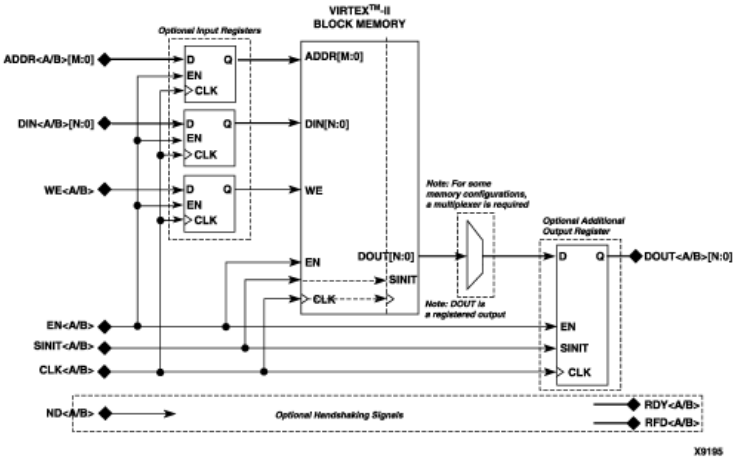


Figure 2: **Dual-Port Memory Block Diagram**

Operating Modes

The Virtex-II/Spartan-3 block SelectRAM-II can maximize the utilization of the True Dual-Port memory at each clock edge by supporting three different write modes. Each port's write mode is independently configurable. The Read-Before-Write mode offers the flexibility of using the data output bus during a write operation on the same port. Output port behavior is determined by the configuration. This choice increases the effective bandwidth of the Block Memory. Note that the Spartan-II/Virtex SelectRAM+ supports only the Read-After-Write mode.

Read Operations

Read operations are synchronous to the rising edge of the clock. The data in the memory location selected by the address appears on the DOUT port after the active edge of the clock.

Write Operations

Write operations are synchronous to the active edge of the clock. The data on the DIN port is written into the memory location selected by the address on the active edge of the clock when WE is active. The user can configure the memory in one of three ways to determine the behavior of the DOUT port during a write cycle. Each port's write mode is independently configurable. Note that the following timing diagrams and descriptions of the write modes assume that the memory has been configured without input registering and additional output registers.

Write First or Read-After-Write Mode

In Write-First mode, data input is loaded simultaneously with a write operation on the DOUT port. As shown in Figure 3, the data input is stored in memory and mirrored on the output.

Read First or Read-Before-Write Mode

In Read-First mode, data previously stored at the write address appears on the DOUT port. Data input is stored in memory and the prior contents of that location is driven on the output, during the same clock cycle (shown in Figure 4).

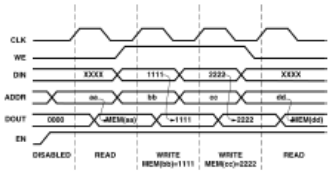


Figure 3: **Write First Mode Waveform**

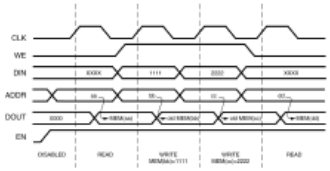


Figure 4: **Read First Mode Waveform**

No Change or No Read-on-Write Mode

In No-Read-on-Write mode, the DOUT port remains unchanged during a write operation. As shown in Figure 5, data output is still the last read data and is unaffected by a write operation on the same port. Mode configuration is static. One of these three modes is set individually for each port by an attribute. The default mode is write first.

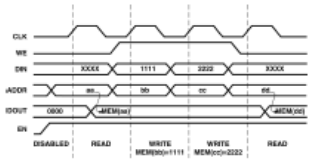


Figure 5: **No-Read-on-Write Mode Waveform**

Conflict Resolution

The Spartan-II/Virtex and Virtex-II/Spartan-3 block memory is True Dual-Port RAM that allows both ports to simultaneously access the same memory location. When one port writes to a given memory location, the other port must not address that memory location (for a write/read) within the clock-to-clock setup window. Note that conflicts do not cause any physical damage to BlockRAM cells. For more information on conflict resolution, refer to the *Spartan-II, Virtex, Virtex-II/Spartan-3 Databook* available at the website: *http://www.xilinx.com/partinfo/databook.htm.*

Specifying Memory Contents

The initial memory contents can be assigned by specifying the desired information in a separate text file called a *coe* file. To select and load a *coe* file, press the "Load Init Values..." button on the parameterization window and choose the desired file from the dialog box. An example of a *coe* file for a 3 by 16 RAM is shown in Figure 6.

```
memory_initialization_radix=16;
memory_initialization_vector=123, 456, aaaa;
```

Figure 6: **Example COE file for Dual-Port Block RAM version 3**

When specifying the initial contents for a memory in a *coe* file, the keywords MEMORY_INITIALIZATION_RADIX and MEMORY_INITIALIZATION_VECTOR can be used. The MEMORY_INITIALIZATION_VECTOR takes the form of a sequence of comma-separated values, one value per memory location, terminated by a semicolon. Any amount of white space, including new lines, can be included in the vector to enhance readability. The format of an individual value in the vector will depend on the MEMORY_INITIALIZATION_RADIX value, which can be 2, 10, or 16 (the default value is 10). The vector must be consistent with the MEMORY_INITIALIZATION_RADIX value and must fall within the range of 0 to $2^{DATA\_WIDTH}-1$. Values must not be negative.

If the initial contents for a memory is specified by a coe file, the initial values will be embedded in the EDIF netlist that is needed for implementation. To support HDL simulations, MIF files containing the initialization values are generated. These files must be copied to the active simulation directory for a successful simulation of a core.

Parameter Values in the XCO File

Names of the XCO parameters and their parameter values are identical to the names and values shown in the GUI, except that underscore characters (_) are used instead of spaces. The text in an XCO file is case insensitive.

Table 2 and Table 3 show the XCO file parameters and values, and summarizes the GUI defaults. The following is an example of the CSET parameters in an XCO file:

```
CSET component_name = abc123
CSET width_a = 16
CSET width_b = 16
CSET depth_a = 256
CSET depth_b = 256
CSET configuration_port_a = read_and_write
CSET configuration_port_b = read_only
CSET write_mode_port_a = read_before_writ
CSET write_mode_port_b = read_only
CSET global_init_value = 456a
CSET load_init_file = true
CSET coefficient_file = example.coe
CSET port_a_enable_pin = false
CSET port_b_enable_pin = true
CSET port_a_handshaking_pins = true
CSET port_b_handshaking_pins = false
CSET port_a_register_inputs = true
CSET port_b_register_inputs = false
CSET port_a_additional_output_pipe_stages = 0
CSET port_b_additional_output_pipe_stages = 1
CSET port_a_init_pin = false
CSET port_b_init_pin = false
CSET port_a_init_value = 1234
CSET port_b_init_value = abcd
CSET primitive_selection = optimize_for_area
CSET select_primitive = 4kx1
CSET port_a_write_enable_polarity = active_high
CSET port_a_enable_pin_polarity = active_high
CSET port_a_initialization_pin_polarity = active_high
CSET port_a_active_clock_edge =
    rising_edge_triggered
CSET port_b_write_enable_polarity = active_high
CSET port_b_enable_pin_polarity = active_high
CSET port_b_initialization_pin_polarity = active_high
CSET port_b_active_clock_edge =
    rising_edge_triggered
```

Core Resource Utilization

The number of Block RAM primitives required is dependent on the values of the data depth and width fields selected in the CORE Generator parameterization window.

For Virtex and Spartan-II implementations, this value must be at least (width x depth)/4096; while for Virtex-II and Spartan-3 implementations this value must be at least (depth x width)/18432. Note that for many configurations, the number of Block RAMs will exceed this estimated value.

For some memory depths, extra logic is required to decode the address and multiplex the outputs from various primitives. Spartan-II, Virtex, Virtex-II, or Virtex-E CLB slices are used to provide this functionality. The number of slices required depends on the way that the depth is constructed from the primitives, the data width, and the implementation of any decoding or multiplexing.

For an accurate measure of the usage of primitives, slices, and CLBs for a particular point solution, check the **Display Core Viewer after Generation** checkbox in the CORE Generator. For more information about the number of block RAMS in each device, please refer to Table 4, Table 5, Table 6, and Table 7.

Table 2: **Parameter File Information for Spartan-II/Virtex**

Parameter Name	XCO Filename Values	Default GUI Setting
component_name	ASCII text starting with a letter and based upon the following character set: a..z, 0..9 and _.	blank
width_[A|B]	Integer in the range of 1 to 256	16
depth_[A|B]	Integer in the range of 2 to 256K	16
configuration_port_ [A|B]	One of the following keywords: read_and_write, read_only, write_only	read_and_write
write_mode_port_ [A|B]	There is only one options for Spartan-II/Virtex architecture: read_after_write	read_after_write
global_init_value	A hex value in the range of 0 to $2^{width\_a|b} - 1$	0
load_init_file	One of the following keywords: true, false	false
coefficient_file	The name of the coe file in ASCII text starting with a letter and based upon the following character set: a..z, 0..9 and _.	blank
port_[A|B]_enable_pin	One of the following keywords: true, false	false
port_[A|B]_handshaking_pins	One of the following keywords: true, false	false
port_[A|B]_register_inputs	One of the following keywords: true, false	false
port_[A|B]_additional _output_pipe_stages	Integer in the range of 0 to 1	0
port_[A|B]_init_pin	One of the following keywords: true, false	false
port_[A|B]_init_value	0	0
primitive_selection	One of two values: optimize_for_area, select_primitive	optimize_for_area
select_primitive	4kx1, 2kx2, 1kx4, 512x8, 256x16	4kx1
port_a_enable_pin_polarity	One of two values: active_high, active_low	active_high
port_a_initialization_pin_polarity	One of two values: active_high, active_low	active_high
port_a_write_enable_pin_polarity	One of two values: active_high, active_low	active_high
port_a_active_clock_edge	One of two values: rising_edge_triggered, falling_edge_triggered	rising_edge_triggered
port_b_enable_pin_polarity	One of two values: active_high, active_low	active_high
port_b_initialization_pin_polarity	One of two values: active_high, active_low	active_high
port_b_write_enable_pin_polarity	One of two values: active_high, active_low	active_high
port_b_active_clock_edge	One of two values: rising_edge_triggered, falling_edge_triggered	rising_edge_triggered

Table 3: **Parameter File Information for Virtex-II/Spartan-3**

Parameter Name	XCO Filename Values	Default GUI Setting
component_name	ASCII text starting with a letter and based upon the following character set: a..z, 0..9 and _.	blank
width_[A|B]	Integer in the range of 1 to 256	16
depth_[A|B]	Integer in the range of 2 to 1M (256K for Spartan-3)	16
configuration_port_ [A|B]	One of the following keywords: read_and_write, read_only, write_only	read_and_write
write_mode_port_ [A|B]	One of the following keywords: read_before_write, read_after_write, no_read_on_write	read_after_write
global_init_value	A hex value in the range of 0 to $2^{width\_a|b} - 1$	0
load_init_file	One of the following keywords: true, false	false
coefficient_file	The name of the coe file in ASCII text starting with a letter and based upon the following character set: a..z, 0..9 and _.	blank
port_[A|B]_enable_pin	One of the following keywords: true, false	false
port_[A|B]_handshaking_pins	One of the following keywords: true, false	false
port_[A|B]_register_inputs	One of the following keywords: true, false	false
port_[A|B]_additional _output_pipe_stages	Integer in the range of 0 to 1	0
port_[A|B]_init_pin	One of the following keywords: true, false	false
port_[A|B]_init_value	A hex value in the range of 0 to $2^{width\_a|b} - 1$	0
primitive_selection	One of two values: optimize_for_area, select_primitive	optimize_for_area
select_primitive	16kx1, 8kx2, 4kx4, 2kx9, 1kx18, 512x36	16kx1
port_a_enable_pin_polarity	One of two values: active_high, active_low	active_high
port_a_initialization_pin_polarity	One of two values: active_high, active_low	active_high
port_a_write_enable_pin_polarity	One of two values: active_high, active_low	active_high
port_a_active_clock_edge	One of two values: rising_edge_triggered, falling_edge_triggered	rising_edge_triggered
port_b_enable_pin_polarity	One of two values: active_high, active_low	active_high
port_b_initialization_pin_polarity	One of two values: active_high, active_low	active_high
port_b_write_enable_pin_polarity	One of two values: active_high, active_low	active_high
port_b_active_clock_edge	One of two values: rising_edge_triggered, falling_edge_triggered	rising_edge_triggered

Table 4: **Spartan-II Device Block RAM Counts**

Devices	# Blocks	Total Block (bits)
xc2s15	4	16384
xc2s30	6	24,576
xc2s50	8	32,768
xc2s100	10	40,960
xc2s150	12	49,152

Table 5: **Virtex Device Block RAM Counts**

Devices	# Blocks	Total Block (bits)
xcv50	8	32,768
xcv100	10	40,910
xcv150	12	49,152
xcv200	14	57,344
xcv300	16	65,536
xcv400	20	81,920
xcv600	24	98,304
xcv800	28	114,688
xcv1000	32	131,072

Table 6: **Virtex-E Device Block RAM Counts**

Devices	# Blocks	Total Block (bits)
xcv50E	16	65,536
xcv100E	20	81,920
xcv200E	28	114,688
xcv300R	32	131,072
xcv400E	40	163,840
xcv600E	72	294,912
xcv1000E	96	393,216
xcv1600E	144	589,824
xcv2000E	160	655,360
xcv2600E	184	753,664
xcv3200E	208	851,968
xcv405E	140	573,440
xcv812E	280	1,146,880

Table 7: **Virtex-II Device Block RAM Counts**

Devices	# Blocks	Total Block (Kb)
xc2v250	24	432
xc2v500	32	576
xc2v1000	40	720
xc2v1500	48	864
xc2v2000	56	1,008
xc2v3000	96	1,728
xc2v4000	120	2,160
xc2v6000	144	2,592
xc2v8000	168	3,026

Ordering Information

This core may be downloaded from the Xilinx IP Center for use with the Xilinx CORE Generator System v5.1i and later. The Xilinx CORE Generator System tool is bundled with all Alliance and Foundation Series Software packages, at no additional charge.

To order Xilinx software, please visit the Xilinx Silicon Xpresso Cafe or contact your local Xilinx sales representative.

Information on additional Xilinx LogiCORE modules is available on the Xilinx IP Center.

6.2 FIFO datasheet from Core Generator

XILINX® LogiCORE™

FIFO Generator v2.1

DS317 April 28, 2005 — Product Specification

Introduction

The Xilinx LogiCORE FIFO Generator is a fully verified first-in first-out (FIFO) memory queue for applications requiring in-order storage and retrieval. The core provides an optimized solution for all FIFO configurations and delivers maximum performance (up to 350 MHz) while utilizing minimum resources. Delivered through the CORE Generator, the structure can be customized by the user, including the width, depth, status flags, memory type, and the write/read port aspect ratios. In this v2.1 release, the core adds support for first-word fall-through, a feature critical to latency-sensitive applications.

This document describes the features of the core, specifies the interfaces, and defines the input and output signals. For detailed information about designing with the core, refer to the *FIFO Generator User Guide*.

LogiCORE™ Facts	
Core Specifics	
Supported Device Families	Virtex-4, Spartan-3E, Spartan-3, Virtex-II Pro, Virtex-II, Virtex-E, Virtex, Spartan-IIE, Spartan-II
Performance and Resources Used	See Tables 8 and 10
Provided with Core	
Documentation	Product Specification User Guide Release Notes
Design File Formats	VHDL, Verilog
Instantiation Template	VHDL, Verilog
Design Tool Requirements	
Implementation	Xilinx ISE v7.1i SP1 or higher
Simulation	Modelsim, NC-SIM
Support	
Provided by Xilinx, Inc.	

Features

- Drop-in module for Virtex™-4, Spartan™-3E, Spartan-3, Virtex-II Pro, Virtex-II, Virtex-E, Virtex, Spartan-IIE, Spartan-II FPGAs
- FIFO depths up to 4,194,304 words
- FIFO data widths from 1 to 256 bits
- Non-symmetric aspect ratios (read-to-write port ratios ranging from 1:8 to 8:1)
- Independent or common clock domains
- Selectable memory type (block RAM, distributed RAM, shift register, or Virtex-4 built-in FIFO)
- First-word fall-through (FWFT)
- Full and empty status flags, and almost full and almost empty flags for indicating one-word-left
- Programmable full and empty status flags, set by user defined constant(s) or dedicated input port(s)
- Configurable handshaking signals
- Fully configurable using the Xilinx CORE Generator™ system v7.1i SP1 or higher

Applications

In digital designs, FIFOs are ubiquitous constructs required for data manipulation tasks such as clock domain crossing, low latency memory buffering, and bus width conversion. Figure 1 highlights just one of many configurations that the FIFO Generator supports. In this example, the design has two independent clock domains and the width of the write data bus is four times wider than the read data bus. Using the FIFO Generator, the user is able to rapidly generate solutions such as this one, that is customized for their specific requirements and provides a solution fully optimized for Xilinx FPGAs.

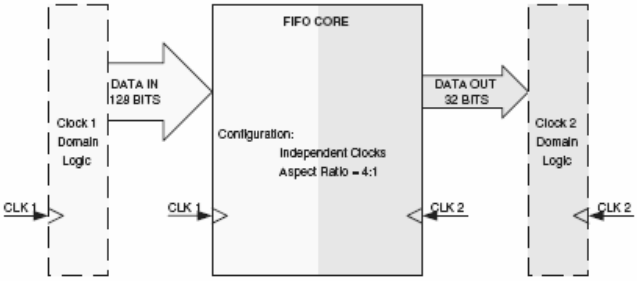


Figure 1: **FIFO Generator Application Example**

Feature Overview

Clock Implementation Operation

The FIFO Generator enables FIFOs to be configured with either independent or common clock domains for write and read operations. The independent clock configuration of the FIFO Generator enables the user to implement unique clock domains on the write and read ports. The FIFO Generator handles the synchronization between clock domains, placing no requirements on phase and frequency. When data buffering in a single clock domain is required, the FIFO Generator can be used to generate a core optimized for that clock.

Built-in FIFO Support in Virtex-4

The FIFO Generator supports the Virtex-4 built-in FIFO modules, enabling large FIFOs to be created by cascading the built-in FIFOs in both width and depth. The core expands the capabilities of the built-in FIFOs by utilizing the FPGA fabric to create optional status flags not implemented in the built-in FIFO macro.

First-Word Fall-Through (FWFT) - *new in v2.1 release*

The first-word fall-through (FWFT) feature provides the ability to look-ahead to the next word available from the FIFO without issuing a read operation. When data is available in the FIFO, the first word falls through the FIFO and appears automatically on the output bus (DOUT). FWFT is useful in applications that require low latency access to data and to applications that require throttling based on the contents of the data that are read. FWFT support is included in independent clock FIFOs created with either block RAM or distributed RAM.

Memory Types

The FIFO Generator implements FIFOs built from block RAM, distributed RAM, shift registers, or the Virtex-4 built-in FIFOs. The core combines memory primitives in an optimal configuration based on the selected width and depth of the FIFO. The following table provides best-use recommendations for specific design requirements.

Table 1: **Memory Configuration Benefits**

	Independent Clocks	Common Clock	Small Buffering	Medium-Large Buffering	High Performance	Minimal Resources
Built-in FIFO	✔	✔		✔	✔	✔
Block RAM	✔	✔		✔	✔	✔
Shift Register		✔	✔		✔	
Distributed RAM	✔		✔		✔	

Non-Symmetric Aspect Ratio

The core supports generating FIFOs whose write and read ports have different widths, enabling automatic width conversion of the data width. Non-symmetric aspect ratios ranging from 1:8 to 8:1 are supported for the write and read port widths. This feature is available for FIFOs implemented with block RAM that are configured to have independent write and read clocks.

FIFO Generator Configurations

Table 2 provides a summary of the supported memory and clock configurations.

Table 2: **FIFO Configurations**

Clock Domain	Memory Type	Supported Configuration	Non-symmetric Aspect Ratios	First-Word Fall-Through
Common	Block RAM	✔		
Common	Distributed RAM	✔		
Common	Shift Register	✔		
Common	Built-in FIFO [1]	✔		
Independent	Block RAM	✔	✔	✔
Independent	Distributed RAM	✔		✔
Independent	Built-in FIFO[1]	✔	[2]	[3]

1. The built-in FIFO primitive is only available in the Virtex-4 architecture.
2. For non-symmetric aspect ratios, use the block RAM implementation (feature not supported in Virtex-4 built-in FIFO primitive).
3. FWFT for the built-in FIFO will be supported by the FIFO Generator in a future release. Contact your Xilinx representative for more details.

Independent Clocks: Block RAM and Distributed RAM

This implementation category allows the user to select block RAM or distributed RAM and supports independent clock domains for write and read data accesses. Operations in the read domain are synchronous to the read clock and operations in the write domain are synchronous to the write clock.

The feature set supported for this type of FIFO includes non-symmetric aspect ratios (different write and read port widths), status flags (full, almost full, empty, and almost empty), as well as programmable full and empty flags generated with user-defined thresholds. Optional read data count and write data count indicators provide the number of words in the FIFO relative to their respective clock domains. In addition, optional handshaking and error flags are available (write acknowledge, overflow, valid, and underflow).

Independent Clocks: Virtex-4 Built-in FIFO

This implementation category allows the user to select the built-in FIFO available in the Virtex-4 architecture. Operations in the read domain are synchronous to the read clock and operations in the write domain are synchronous to the write clock.

The feature set supported for this configuration includes status flags (full and empty) and programmable full and empty flags generated with user-defined thresholds. In addition, optional handshaking and error flags are available (write acknowledge, overflow, valid, and underflow).

Common Clock: Block RAM, Distributed RAM, Shift Register

This implementation category allows the user to select block RAM, distributed RAM, or shift register and supports a common clock for write and read data accesses.

The feature set supported for this configuration includes status flags (full, almost full, empty, and almost empty) and programmable empty and full flags generated with user-defined thresholds. In addition, optional handshaking and error flags are supported (write acknowledge, overflow, valid, and underflow), and an optional data count provides the number of words in the FIFO.

Common Clock: Virtex-4 Built-in FIFO

This implementation category allows the user to select the built-in FIFO available in the Virtex-4 architecture and supports a common clock for write and read data accesses.

The feature set supported for this configuration includes status flags (full and empty) and optional programmable full and empty flags with user-defined thresholds. In addition, optional handshaking and error flags are available (write acknowledge, overflow, valid, and underflow).

FIFO Generator Features

Table 3 summarizes the FIFO Generator features supported for each clock configuration and memory type. For more detailed information, refer to the *FIFO Generator User Guide*.

Table 3: **FIFO Configurations Summary**

FIFO Feature	Independent Clocks			Common Clock		
	Block RAM	Distributed RAM	Built-in FIFO	Block RAM	Distributed RAM, Shift Register	Built-in FIFO
Non-symmetric Aspect Ratios	✔			[1]		
Symmetric Aspect Ratios	✔	✔	✔	✔	✔	✔
Almost Full	✔	✔		✔	✔	
Almost Empty	✔	✔		✔	✔	
Handshaking	✔	✔	✔	✔	✔	✔
Data Count	✔	✔		✔	✔	
Programmable Empty/Full Thresholds	✔	✔	✔	✔	✔	✔
First-Word Fall-Through	✔[2]	✔[2]				
DOUT Reset Value	✔[3]	✔		✔[3]	✔	

1. For applications with a single clock that require non-symmetric ports, use the independent clock configuration and connect the write and read clocks to the same source. A dedicated solution for common clocks will be available in a future release. Contact your Xilinx representative for more details.
2. To enable First-Word Fall-Through, select Registered Outputs with Read Latency 0 on page 2 of the GUI.
3. All architectures except for Virtex, Virtex-E, Spartan-II, and Spartan-IIE.

FIFO Interfaces

The following two sections provide definitions for the FIFO interface signals. Figure 2 illustrates these signals (both the standard and optional ports) for a FIFO core that supports independent write and read clocks.

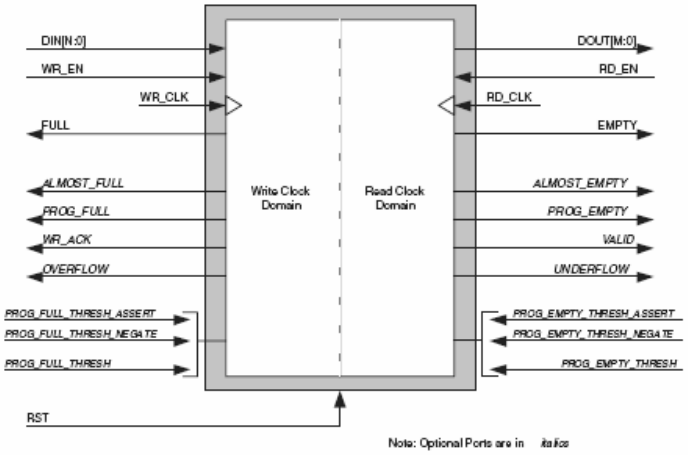


Figure 2: **FIFO with Independent Clocks: Interface Signals**

Interface Signals: FIFOs With Independent Clocks

The signal (RST) causes a reset of the entire core logic (both write and read clock domains) and is defined in Table 4. It is an asynchronous input which is synchronized internally in the core before being used. The initial hardware reset should be generated by the user. See the *FIFO Generator User Guide* for specific information on reset requirements and behavior.

Table 4: **Reset Signal for FIFOs with Independent Clocks**

Name	Direction	Description
RST	Input	Reset: This signal is an asynchronous reset that initializes all internal pointers and output registers.

Table 5 defines the signals for the write interface for FIFOs with independent clocks. The write interface signals are divided into required and optional signals and all signals are synchronous to the write clock (WR_CLK).

Table 5: **Write Interface Signals for FIFOs with Independent Clocks**

Name	Direction	Description
		Required
WR_CLK	Input	Write Clock: All signals on the write domain are synchronous to this clock.
DIN[N:0]	Input	Data Input: The input data bus used when writing the FIFO.
WR_EN	Input	Write Enable: If the FIFO is not full, asserting this signal causes data (on DIN) to be written to the FIFO.
FULL	Output	Full Flag: When asserted, this signal indicates that the FIFO is full. Write requests are ignored when the FIFO is full, initiating a write when the FIFO is full is non-destructive to the contents of the FIFO.
		Optional
ALMOST_FULL	Output	Almost Full: When asserted, this signal indicates that only one more write can be performed before the FIFO is full.
PROG_FULL	Output	Programmable Full: This signal is asserted when the number of words in the FIFO is greater than or equal to the assert threshold. It is deasserted when the number of words in the FIFO is less than the negate threshold.
WR_DATA_COUNT [D:0]	Output	Write Data Count: This bus indicates the number of words stored in the FIFO. The count is guaranteed to never under-report the number of words in the FIFO, to ensure the user never overflows the FIFO.
WR_ACK	Output	Write Acknowledge: This signal indicates that a write request (WR_EN) during the prior clock cycle succeeded.
OVERFLOW	Output	Overflow: This signal indicates that a write request (WR_EN) during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is non-destructive to the contents of the FIFO.
PROG_FULL_THRESH	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. The user can either choose to set the assert and negate threshold to the same value (using PROG_FULL_THRESH), or the user can control these values independently (using PROG_FULL_THRESH_ASSERT and PROG_FULL_THRESH_NEGATE).
PROG_FULL_THRESH_ASSERT	Input	Programmable Full Threshold Assert: This signal is used to set the upper threshold value for the programmable full flag, which defines when the signal is asserted. The threshold can be dynamically set in-circuit during reset.
PROG_FULL_THRESH_NEGATE	Input	Programmable Full Threshold Negate: This signal is used to set the lower threshold value for the programmable full flag, which defines when the signal is de-asserted. The threshold can be dynamically set in-circuit during reset.

Table 6 defines the signals on the read interface of a FIFO with independent clocks. The read interface signals are divided into required signals and optional signals, and all signals are synchronous to the read clock (RD_CLK).

Table 6: **Read Interface Signals for FIFOs with Independent Clocks**

Name	Direction	Description
		Required
RD_CLK	Input	Read Clock: All signals on the read domain are synchronous to this clock.
DOUT[M:0]	Output	Data Output: The output data bus is driven when reading the FIFO.
RD_EN	Input	Read Enable: If the FIFO is not empty, asserting this signal causes data to be read from the FIFO (output on DOUT).
EMPTY	Output	Empty Flag: When asserted, this signal indicates that the FIFO is empty. Read requests are ignored when the FIFO is empty, initiating a read while empty is non-destructive to the FIFO.
		Optional
ALMOST_EMPTY	Output	Almost Empty Flag: When asserted, this signal indicates that the FIFO is almost empty and one word remains in the FIFO.
PROG_EMPTY	Output	Programmable Empty: This signal is asserted when the number of words in the FIFO is less than or equal to the programmable threshold. It is de-asserted when the number of words in the FIFO exceeds the programmable threshold.
RD_DATA_COUNT [C:0]	Output	Read Data Count: This bus indicates the number of words available for reading in the FIFO. The count is guaranteed to never over-report the number of words available for reading, to ensure that the user does not underflow the FIFO.
VALID	Output	Valid: This signal indicates that valid data is available on the output bus (DOUT).
UNDERFLOW	Output	Underflow: Indicates that the read request (RD_EN) during the previous clock cycle was rejected because the FIFO is empty. Underflowing the FIFO is non-destructive to the FIFO.
PROG_EMPTY_THRESH	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset. The user can either choose to set the assert and negate threshold to the same value (using PROG_EMPTY_THRESH), or the user can control these values independently (using PROG_EMPTY_THRESH_ASSERT and PROG_EMPTY_THRESH_NEGATE).
PROG_EMPTY_THRESH_ASSERT	Input	Programmable Empty Threshold Assert: This signal is used to set the lower threshold value for the programmable empty flag, which defines when the signal is asserted. The threshold can be dynamically set in-circuit during reset.
PROG_EMPTY_THRESH_NEGATE	Input	Programmable Empty Threshold Negate: This signal is used to set the upper threshold value for the programmable empty flag, which defines when the signal is de-asserted. The threshold can be dynamically set in-circuit during reset.

Interface Signals: FIFOs with Common Clock

Table 7 defines the interface signals of a FIFO with a common write and read clock. The table is divided into standard and optional interface signals, and all signals (except reset) are synchronous to the common clock (CLK). See the *FIFO Generator User Guide* for specific information on reset requirements and behavior.

Table 7: **Interface Signals for FIFOs with a Common Clock**

Name	Direction	Description
		Required
RST	Input	Reset: This signal is an asynchronous reset that initializes all internal pointers and output registers.
CLK	Input	Clock: All signals on the write and read domains are synchronous to this clock.
DIN[N:0]	Input	Data Input: The input data bus used when writing the FIFO.
WR_EN	Input	Write Enable: If the FIFO is not full, asserting this signal causes data (on DIN) to be written to the FIFO.
FULL	Output	Full Flag: When asserted, this signal indicates that the FIFO is full. Write requests are ignored when the FIFO is full, initiating a write when the FIFO is full is non-destructive to the contents of the FIFO.
DOUT[M:0]	Output	Data Output: The output data bus driven when reading the FIFO.
RD_EN	Input	Read Enable: If the FIFO is not empty, asserting this signal causes data to be read from the FIFO (output on DOUT).
EMPTY	Output	Empty Flag: When asserted, this signal indicates that the FIFO is empty. Read requests are ignored when the FIFO is empty, initiating a read while empty is non-destructive to the FIFO.
		Optional
DATA_COUNT [C:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO.
ALMOST_FULL	Output	Almost Full: When asserted, this signal indicates that only one more write can be performed before the FIFO is full.
PROG_FULL	Output	Programmable Full: This signal is asserted when the number of words in the FIFO is greater than or equal to the assert threshold. It is deasserted when the number of words in the FIFO is less than the negate threshold.
WR_ACK	Output	Write Acknowledge: This signal indicates that a write request (WR_EN) during the prior clock cycle succeeded.
OVERFLOW	Output	Overflow: This signal indicates that a write request (WR_EN) during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is non-destructive to the FIFO.
PROG_FULL_THRESH	Input	Programmable Full Threshold: This signal is used to set the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. The user can either choose to set the assert and negate threshold to the same value (using PROG_FULL_THRESH), or the user can control these values independently (using PROG_FULL_THRESH_ASSERT and PROG_FULL_THRESH_NEGATE).
PROG_FULL_THRESH_ASSERT	Input	Programmable Full Threshold Assert: This signal is used to set the upper threshold value for the programmable full flag, which defines when the signal is asserted. The threshold can be dynamically set in-circuit during reset.

Table 7: **Interface Signals for FIFOs with a Common Clock** *(Continued)*

Name	Direction	Description
PROG_FULL_THRESH_NEGATE	Input	Programmable Full Threshold Negate: This signal is used to set the lower threshold value for the programmable full flag, which defines when the signal is de-asserted. The threshold can be dynamically set in-circuit during reset.
ALMOST_EMPTY	Output	Almost Empty Flag: When asserted, this signal indicates that the FIFO is almost empty and one word remains in the FIFO.
PROG_EMPTY	Output	Programmable Empty: This signal is asserted after the number of words in the FIFO is less than or equal to the programmable threshold. It is de-asserted when the number of words in the FIFO exceeds the programmable threshold.
VALID	Output	Valid: This signal indicates that valid data is available on the output bus (DOUT).
UNDERFLOW	Output	Underflow: Indicates that read request (RD_EN) during the previous clock cycle was rejected because the FIFO is empty. Underflowing the FIFO is non-destructive to the FIFO.
PROG_EMPTY_THRESH	Input	Programmable Empty Threshold: This signal is used to set the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset. The user can either choose to set the assert and negate threshold to the same value (using PROG_EMPTY_THRESH), or the user can control these values independently (using PROG_EMPTY_THRESH_ASSERT and PROG_EMPTY_THRESH_NEGATE).
PROG_EMPTY_THRESH_ASSERT	Input	Programmable Empty Threshold Assert: This signal is used to set the lower threshold value for the programmable empty flag, which defines when the signal is asserted. The threshold can be dynamically set in-circuit during reset.
PROG_EMPTY_THRESH_NEGATE	Input	Programmable Empty Threshold Negate: This signal is used to set the upper threshold value for the programmable empty flag, which defines when the signal is de-asserted. The threshold can be dynamically set in-circuit during reset.

Resource Utilization and Performance

Performance and resource utilization for a FIFO varies depending on the configuration and features selected when customizing the core. The tables below provide example FIFO configurations and the maximum performance and resources required.

Table 8 provides results for a FIFO configured without optional features. The benchmarks were performed using a Virtex-II 2v3000 -5 and Virtex-4 4vlx15 -11.

Table 8: **Benchmarks: FIFO Configured Without Optional Features**

FIFO Type	Depth x Width	Performance		Resources				
		Virtex-II	Virtex-4	LUTs	FFs	Block RAMs	Shift Registers	Distributed RAMs
Synchronous FIFO (block RAM)	512 x 16	233 MHz	275 MHz	64	43	1	0	0
Synchronous FIFO (block RAM)	4096 x 16	200 MHz	275 MHz	80	55	4	0	0
Synchronous FIFO (distributed RAM)	64 x 16	250 MHz	325 MHz	49	83	0	0	128
Synchronous FIFO (distributed RAM)	512 x 16	175 MHz	225 MHz	135	438	0	0	1024
Independent Clocks FIFO (block RAM)	512 x 16	225 MHz	275 MHz	103	105	1	0	0
Independent Clocks FIFO (block RAM)	4096 x 16	225 MHz	275 MHz	134	138	4	0	0
Independent Clocks FIFO (distributed RAM)	64 x 16	250 MHz	325 MHz	70	123	0	0	128
Independent Clocks FIFO (distributed RAM)	512 x 16	175 MHz	225 MHz	173	239	0	0	1024
Shift Register FIFO	64 x 16	225 MHz	325 MHz	68	43	0	64	0
Shift Register FIFO	512 x 16	150 MHz	200 MHz	312	82	0	512	0

Table 9 provides results for a FIFO configured with multiple threshold inputs. The benchmarks were performed using a Virtex-II 2v3000 -5 and Virtex-4 4vlx15 -11.

Table 9: **FIFO Benchmarks: Configured with Multiple Programmable Thresholds**

FIFO Type	Depth x Width	Performance		Resources				
		Virtex-II	Virtex-4	LUTs	FFs	Block RAMs	Shift Registers	Distributed RAMs
Synchronous FIFO (block RAM)	512 x 16	175 MHz	200 MHz	109	68	1	0	0
Synchronous FIFO (block RAM)	4096 x 16	175 MHz	225 MHz	135	86	4	0	0
Synchronous FIFO (distributed RAM)	64 x 16	225 MHz	250 MHz	80	109	0	0	128
Synchronous FIFO (distributed RAM)	512 x 16	150 MHz	200 MHz	182	420	0	0	1024
Independent Clocks FIFO (block RAM)	512 x 16	200 MHz	250 MHz	163	126	1	0	0
Independent Clocks FIFO (block RAM)	4096 x 16	175 MHz	250 MHz	214	165	4	0	0
Independent Clocks FIFO (distributed RAM)	64 x 16	225 MHz	325 MHz	143	146	0	0	128
Independent Clocks FIFO (distributed RAM)	512 x 16	150 MHz	225 MHz	251	277	0	0	1024
Shift Register FIFO	64 x 16	150 MHz	200 MHz	97	62	0	64	0
Shift Register FIFO	512 x 16	150 MHz	200 MHz	355	106	0	512	0

Table 10 provides results for FIFOs configured to use the Virtex-4 built-in FIFO without optional features. The benchmarks were done against a Virtex-4 4vlx15-11 SF363 target device.

Table 10: **Benchmarks: FIFO Configured with Virtex-4 Built-in FIFO Resources**

FIFO Type	Depth x Width	Performance	Resources		
			LUTs	FFs	Built-in FIFOs
Built-in FIFO (basic)	512 x 36	350 MHz	2	2	1
Built-in FIFO (basic)	4096 x 36	300 MHz	5	2	4
Built-in FIFO (with handshaking)	512 x 36	350 MHz	4	8	1
Built-in FIFO (with handshaking)	4096 x 36	300 MHz	7	8	4

Verification

Xilinx has verified the FIFO Generator core in a proprietary test environment, using an internally developed bus functional model. Tens of thousands of test vectors were generated and verified, including both valid and invalid write and read data accesses.

References

UG175 FIFO Generator User Guide

Support

For technical support, visit http://www.xilinx.com/support. Xilinx provides technical support for this product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of the product if implemented in devices not listed in the documentation, if customized beyond that allowed in the product documentation, or if any changes are made in sections of design marked *DO NOT MODIFY*.

Revision History

The table below shows the revision history of this document.

Date	Version	Revision
4/23/04	1.0	Initial Xilinx release.
5/21/04	1.1	Support for Virtex-4 built-in FIFO implementation.
11/11/04	2.0	Updated for Xilinx software v6.3i.
04/28/05	2.1	The original product specification has been divided into two documents - a data sheet and a user guide. The document has also been updated to indicate core support of first-word fall-through feature and Xilinx software v7.1i.