UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

FORWARD ERROR CORRECTION IN OPTICAL ETHERNET COMMUNICATIONS

A Degree's Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Jordi Oliveras Boada

In partial fulfilment

of the requirements for the degree in

Telecommunications systems ENGINEERING

Advisor: Josep Prat Gomà

Barcelona, June 2014

Abstract

A way of incrementing the amount of information sent through an optical fibre is ud-WDM (ultra dense – Wavelength Division Multiplexing). The problem is that the sensitivity of the receiver requires certain SNR (Signal Noise Ratio) that are only achieved in low distances, so to increase them a codification called FEC (Forward Error Correction) can be used. This should reduce the BER (Bit Error Rate) at the receiver letting the signal to be transmitted to longer distances. Another problem that has to be faced is that due to the phase noise of the lasers, there are BER floors that cannot be reduced incrementing the SNR so that makes the use of the FEC even more interesting. Moreover, Ethernet also does another codification called 8b/10b and this one has to be combined with the FEC. This project tests different models of encoding the signals to try to reach the one that is the most useful.

Resum

Una manera d’incrementar la informació que es pot enviar per fibra òptica és la ud-WDM (ultra dens – multiplexat per divisió de longitud d’ona). El problema és que degut a la limitada sensibilitat dels receptors es requereix una SNR (Relació Senyal a Soroll) que només es pot obtenir a curtes distancies, i per tal d’incrementar-la es pot usar una codificació anomenada FEC (Correcció d’Errors a Posterior). Això hauria de permetre reduir la BER (quantitat de bits erronis) al receptor permetent la possibilitat de transmetre el senyal a més llargues distàncies. Un altre problema que ens hem d’enfrontar és a que degut al soroll de fase dels làsers hi ha BER mínims que no es poden reduir incrementant la SNR, el qual fa l’ús del FEC encara més interessant. A més a més, Ethernet fa una altre codificació anomenada 8b/10b i ha de ser combinada amb la FEC. Aquest projecte prova diferents models per tal de trobar el que sigui més òptim.

Resumen

Un modo de incrementar la información que se puede enviar a través de una fibra óptica es usando ud-WDM (ultra denso – multiplexado por división de longitud de onda). El problema es que debido a limitaciones en la sensibilidad del receptor se requieren SNR (Relación de Señal a Ruido) solo conseguibles a cortas distancias, y con tal de incrementarla se puede usar una codificación llamada FEC (Corrección de Errores a Posteriori). Esto debería permitir reducir la BER (cantidad de bits erróneos) al receptor permitiendo así la posibilidad de transmitir la señal a distancias más largas. Otro problema que resolver es que debido a ruido de fase de los láseres aparecen limites de BER que no pueden ser reducidos aunque se incremente la SNR, lo cual hace aún mas interesante la FEC. Además, Ethernet hace otra codificación llamada 8b/10b que tiene que ser combinada con la FEC. Este proyecto prueba distintos modelos con el objetivo de conocer el sistema más óptimo.

Acknowledgements

I would like to express my gratitude towards a couple of people that had helped a lot in this project. First of all, to my project tutor Josep Prat who guided on how to develop properly the project and oriented it. Also I would like to thank all the team that has been helping me every time I needed, but specially to Victor Polo and Javi Martinez.

Revision history and approval record

Revision	Date	Purpose
0	09/07/2014	Document creation
1	10/07/2014	Document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
[Student name] Jordi Oliveras Boada	jordi0603@gmail.com
[Project Supervisor 1] Josep Prat Gomà	jprat@tsc.upc.edu
[Project Supervisor 2]	

Written by:		Reviewed and approved by:	
Date	09/07/2014	Date	10/07/2014
Name	Jordi Oliveras Boada	Name	Josep Prat Gomà
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables:

1. Introduction

A passive optical network (PON) is an access telecommunications network that uses optical devices mentioned below. The main parts are the OLT (optical line terminal) that is what provides the service (located on the central office) and the ONUs (optical network terminal) located as near to the user as possible. The network itself is formed of optical fibres, optical splitters and other non-active devices such as multiplexers and filters.

The main advantages of the PONs are the lack of need of powering the installation, the optical devices do not need to be powered, and also that there is no necessity to maintain it, once it is built it is not necessary to touch it again.

Wavelength Division Multiplexing, shortened as WDM, is a technique to multiplex different signals and send them through the same optical fibre, using for each signal a different wavelength. This projects aims to use ud-WDM (ultra-dense) through the current optical networks installed. Ultra-dense means that it has very narrow spacing between frequencies.

1.1. Motivation

Every modern computer has an Ethernet chip at 1Gbit/s, maybe optical or maybe electric. In an optical system of communication, the power budget allowed is 28 dB; to get that in quite long distances it is necessary to use FEC (Forward Error Correction) in order to enhance the bit error rate (BER).

The FEC consist on adding redundancy to the Ethernet signal to be able to correct errors at the receiver. This allows going from a BER of 10^{-3} to a one of 10^{-10}. This codification is already implemented for 10G Ethernet; however, it is not for the 1G, which is the one that can be found in most personal computers.

To sum up, if there is the need of creating a low-cost 1Gbit/s system network (because is the usual LAN (Local Area Network) rate to access), this redundancy must be implemented. Otherwise the cost of the installation in order to maintain the 28 dB of power budget for a certain BER would increase considerably.

1.2. Requirements

The requirements of the system can be divided into 2 types: the requirements imposed by the standards and the technical requirements of the transmitter and receiver.

Project requirements:

- The distance at which something can be transmitted must increase considerably when using the FEC codification, which will enhance the BER.
- The size of the frame cannot increase too much when including the FEC (bandwidth efficiency).

Standard	Specifications
IEEE Section III. Clause 36.2.4.	• 8B/10B transmission codes specified in the standard have high transition density, run-length limited and have DC balance. • The running disparity value has to be used and is re-

	calculated for every code-group transmitted. • The running disparity is also contained on the receiver, and if it matches with the code-group received is considered valid and decoded, otherwise it is considered invalid.
IEEE Section V. Clause 65.2.3.	• Optional. • BER objective 10-12 at PCS. • BER objective 10-4 at FEC sublayers. • FEC code used is a linear cyclic bloc code. • RS(255,239,8) over a GF(28).
ITU-T G.975	• $BER_{input}=BER_c+BER_{output}$ (BER_c are the number of bits corrected). • The system works properly for $10^{-3}<BER_{input}<10^{-15}$. • Blocks of N = K + R where N is the number of symbols, K the symbols with information and R the redundancy. • Codification used: RS(255,239) with 8 bits words.
SMPTE St 2022-1	• RTP shall be required. • The error correcting function is XOR.

Table 1.1. Table with a summary

On the table above, the most relevant specifications of the standards of IEEE and ITU can be observed. For further information, check ANNEX 1. Standard summary table, which is also referenced to all the standards used.

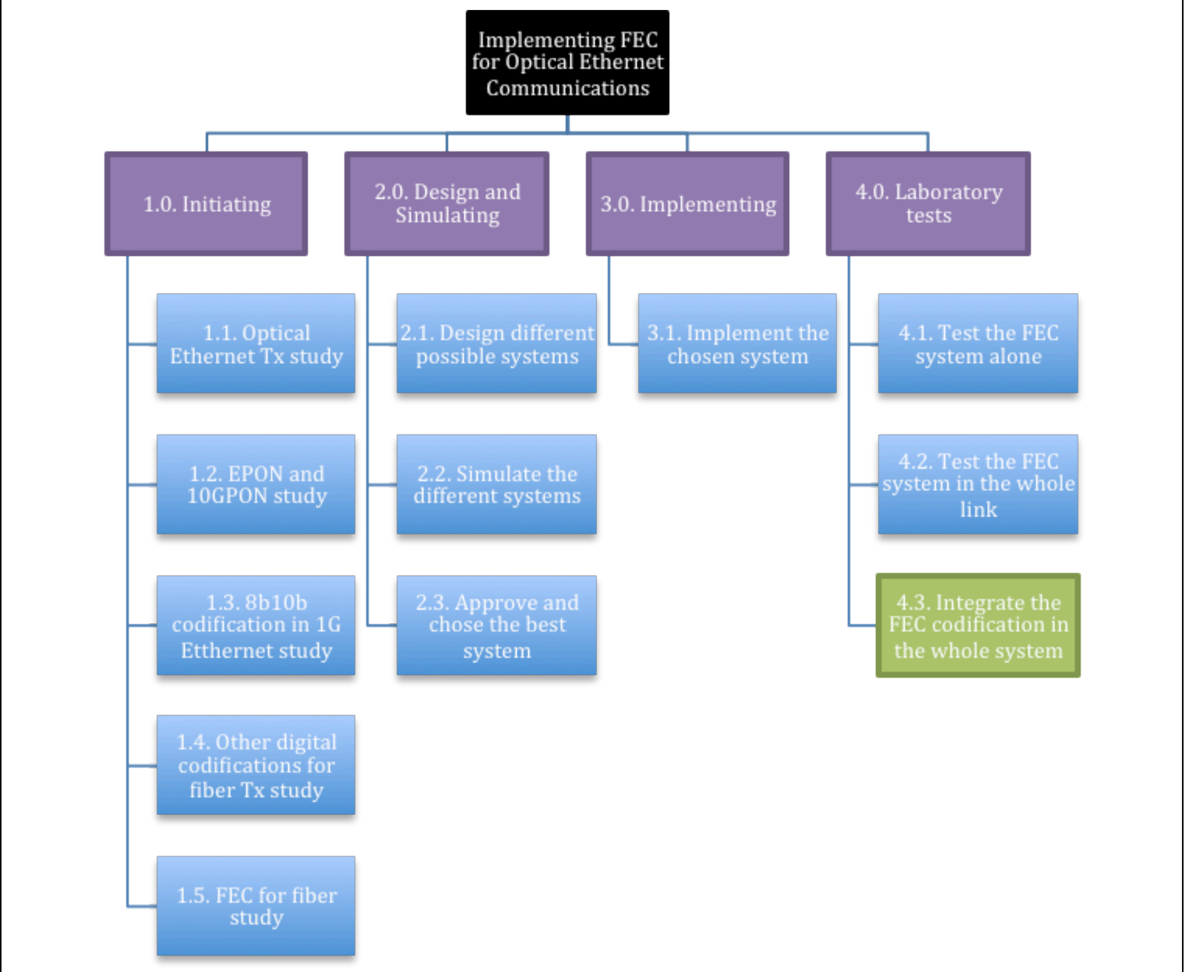
1.3. Method

This project is part of another more general project that is trying to implement ud-WDM (explained above) using the current optical installations. This concrete part of the project is focusing on implementing a forward error correction (FEC) to the message and observe how improves the BER.

To do so, first some mathematical models simulated with matlab, comparing the BER using and not using FEC, concretely a Reed-Solomon codification. After performing those simulations it will also be test using a laboratory prototype of the current optical installation.

1.4. Tasks

The initial work breakdown structure (WBS) was the following:



Implementing FEC for Optical Ethernet Communications
1.0. Initiating
2.0. Design and Simulating
3.0. Implementing
4.0. Laboratory tests
1.1. Optical Ethernet Tx study
1.2. EPON and 10GPON study
1.3. 8b10b codification in 1G Etthernet study
1.4. Other digital codifications for fiber Tx study
1.5. FEC for fiber study
2.1. Design different possible systems
2.2. Simulate the different systems
2.3. Approve and chose the best system
3.1. Implement the chosen system
4.1. Test the FEC system alone
4.2. Test the FEC system in the whole link
4.3. Integrate the FEC codification in the whole system

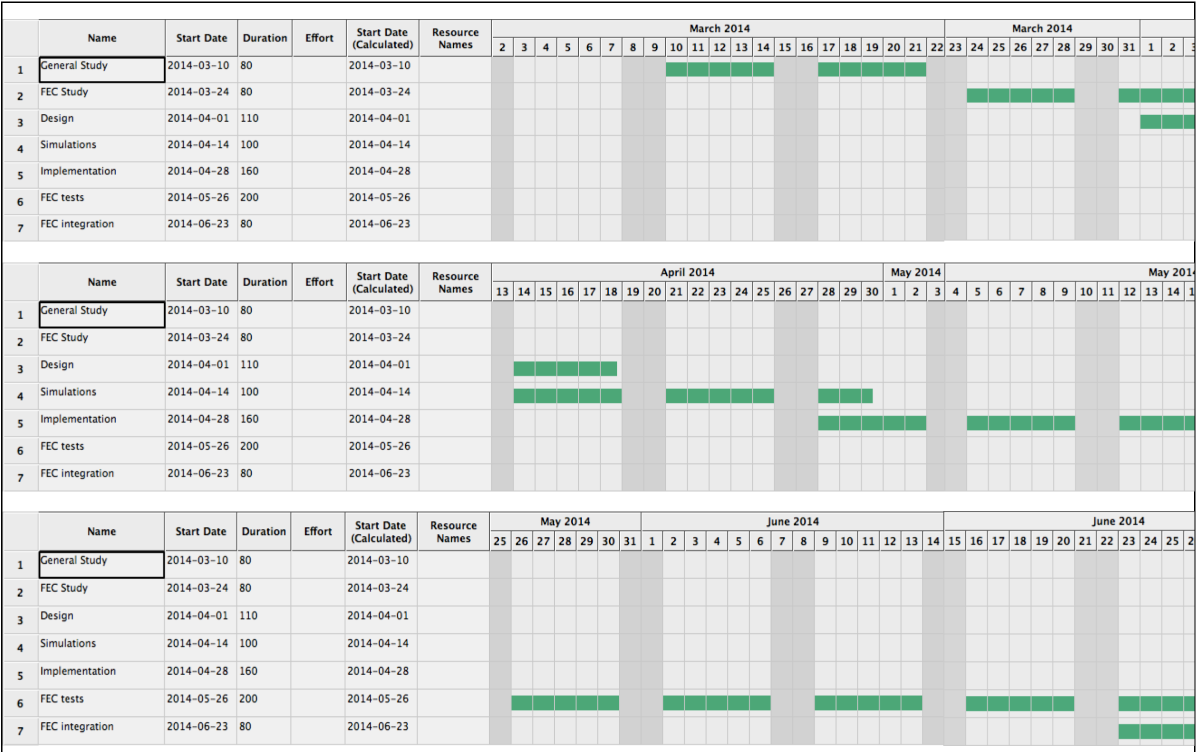
Figure 1.1. Initial WBS.

However this WBS has suffered some modifications. These changes are explained at the point 1.5.Deviations from the initial plan. The work packages can be found on ANNEX 2. Initial Work Packages, and milestones associated with this initial WBS can be found below:

WP#	Task#	Short title	Milestone / deliverable	Date (week)
2	2	CDR	Deliver CDR including the chosen design, the studies performed to support this choice.	30/04/14
3	1	Implementation	Deliver the implementation of the FEC codification.	25/05/14
4	1	FEC tests	Deliver a document with the results of the tests.	29/06/14
4	2	Integration	Demonstration of the whole system.	06/07/14

Table 1.2. List of milestones.

The corresponding Gantt diagram of this project is the following:



	Name	Start Date	Duration	Effort	Start Date (Calculated)	Resource Names
1	General Study	2014-03-10	80		2014-03-10	
2	FEC Study	2014-03-24	80		2014-03-24	
3	Design	2014-04-01	110		2014-04-01	
4	Simulations	2014-04-14	100		2014-04-14	
5	Implementation	2014-04-28	160		2014-04-28	
6	FEC tests	2014-05-26	200		2014-05-26	
7	FEC integration	2014-06-23	80		2014-06-23	

Figure 1.2. Initial Gantt diagram of the project.

1.5. Deviations from the initial plan

The design of the FEC codifications RS required for the standards was already developed so instead of focusing on creating this codification from scratch, the project has moved to implement it to the current PONs.

More precise, some simulations have been performed, emulating the Ethernet 8b/10b encoding and decoding, the FEC codification and decodification and the noise of the channel. Moreover changing a little bit the position of each part of the system has also been added to the objective of this project (e.g. doing first 8b/10b or FEC).

2. State of the art of the technology used or applied in this thesis:

2.1. System

As commented on the introduction, this project aims to use the current communication systems already deployed. This scheme below shows the part of the system that would be contained in each house.

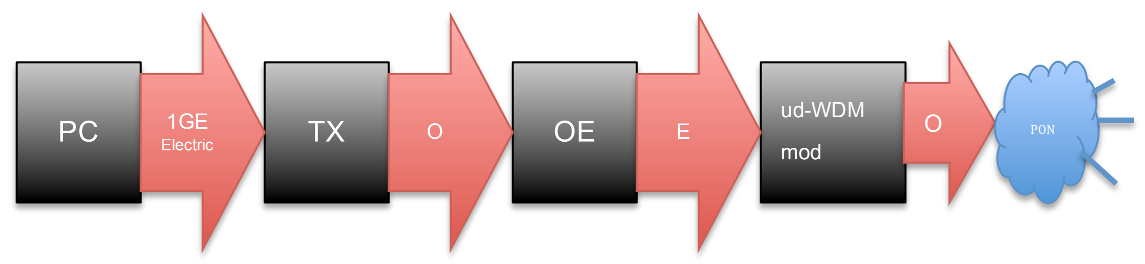


Figure 2.1. Scheme of the part of the system contained at home.

The first bloc is the one where the user can operate. It can be a computer, a laptop or whatever element which can transmit in 1 Gbit/s Ethernet, either in optical or in electrical domain. The next block is the transmitter, this block is in charge first of all to convert the signal to optical and modulate it. For this system it doesn't matter if it transmits on second or the third window of the optical spectrum because it will have to be converted into electrical again for using the ud-WDM laser and modulator. These are the third and fourth blocks.

Once this signal goes out of this first part of the system, it enters to the PON. Here the signal can find optical multiplexers and de-multiplexers, splitters, optical switches… Then this signal arrives to the co (central office), the receiver, which its bloc diagram can be seen below.

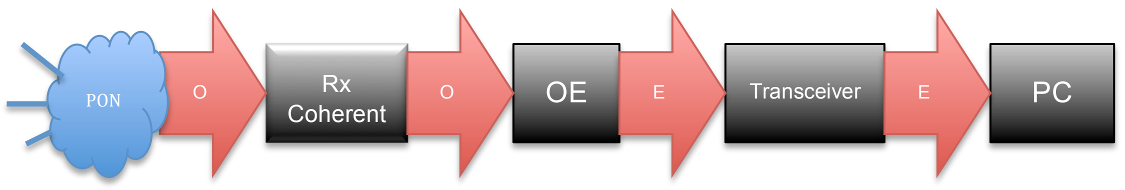


Figure 2.2. Scheme of the part of the system located at the co.

As it can be seen the first part of the receiver is the coherent detector used to demodulate properly the signal. Then this signal is converted into the electrical domain in order to go into the transceiver. Once the information is known, it can be electrically transmitted to the PC used as a receiver. This system has to follow the standards specified on the sub-section of the introduction of this project 1.2. Requirements.

2.2. Laboratory equipment

Here below a list can be found with all the equipment available to do tests in the lab, which is a really good orientation of the technology used nowadays:

Name	Type	Characteristics	Amount
Trend Unipro GbE	Tx/Rx/Analyzer		2
Planet Networking & Communication	Gigabit Ethernet Converter	100Base-T to 1000Base-XS/LX	5+2
	Gigabit Ethernet Media Converter	Fiber/T Transeiver 10/100/1000 Base-T & 1000 Base-SX	3
Infineon Technologies	OLT1G	V23818-S5-V2 REV	2
Luminent OIC	SFP	1250nm	1
Sumito Electric	SFP		4
JPSV	Electrical converter	10G	1
	SFP	1310nm	1
	SFP	52M	1
Luminent OIC	SFP	1551.225nm	1
Alcatel (Sumito Electric)	SFP		1
Optoway	SFP	1310nm	2
APAC Opto	SFP	1550nm 1.25Gbps	1
SMC Networks	SFP	10Gbase	1
Finisar	SFP	1547.7nm	1
Finisar	SFP	1552.52nm	1
Zenko	SFP		1
	Cable	SC green - LC blue	5
	Cable	LC blue - LC blue	12
	Cable	SC blue - LC blue	7
	Cable	SC blue - LC green	1
	Adapter	2x2 LC	2
	Adapter	SC-SC	1

Table 2.1. List of equipment of the lab.

Tx/Rx/Analyser: It is a machine capable of either transmitting, receiving or analysing some parameters of the communication link such as the BER.

Gigabit Ethernet Media Converter: It is a simple device that tries to connect two different types of media, for example an optical fibre with an RJ45.

OLT: Optical Line Termination/Terminal is a device that is used at an endpoint of a PON. It has to receive the electrical control signals to control the optical signals of the PON and also to coordinate the multiplexing between it and the other end of the PON, either if it is an OLT or an ONU (Optical Network Unit).

SFP: Small Form-factor Pluggable is a transceiver used in communications to connect a network device with an optical fibre. There can be different types according to the rate or to the wavelength it transmits.

Connector LC: These types of connectors are small (can be connected to another LC connector using an LC adapter).

Connector SC: These types of connectors are much bigger than LCs (can be connected to another SC using an SC adapter).

Green connector: These types of connectors are when the optical fibre is cut with a certain angle to avoid return losses of the light at the connecting point.

Blue connector: These types of connectors have the cut perpendicular to the optical fibre.

2.3. Codifications

According to the standards, there is a compulsory codification before sending Ethernet data through an optical fibre: 8b10b. The aim of this encoding is to convert a symbol of 8 bits into a 10 bits. The motivation of doing that is to ensure the achievement of a DC-balance and also to simplify the clock recovery at the receiver.

It can be found that the 8 bit symbol to be 8 0s or 8 1s, so there can be problems with the receiver synchronization and is also adding a high continuous component to the signal. This codification ensures that this will not happen. This encoding is done separating the 5 lower bits and the 3 top ones and also takes into account which is the less significant bit of each pack and the disparity of the signal. See ANNEX 3. 8b/10b encoding tables to see the encoding tables.

This codding tables work the following way, the initial byte is formed by the 8 bits HGFEDCBA where A is the lowest bit and H the highest. As said before, the codification separates the EDCBA bits and converts them into abcdei using the first table from the HGF bits, which are converted to fghj using the second table.

There are some of this words that contain more 1s that 0s and vice versa. To control so, it uses the current parity RD. This can be 1 or -1, and it changes every time a conflictive word is encoded, e.g. RD=1 and codifies a 00000 the codified word will be 011000 and RD=-1.

Doing this process at the inverse can also be used to detect errors. The errors can be caused because it arrives a wrong bit and the decoder does not have the translation for that word, because even the word has translation it does not correspond with the current RD or because there was a non-detected error before and the RD was wrongly computed.

When 8b/10b decoder detects an invalid word, it sets it to an arbitrary number and sends also a signal warning the error.

FEC codifications, also known as channel codding, is a process used to control errors in a data transmitted over channels that may introduce errors. It works by adding redundancy on the transmitted signal in order to be able to recover it at the receiver.

There are two types of FEC, fixed-size blocs and convolutional codes. In this project it is used a specific type of the first category, named Reed-Solomon. This codification comes from a polynomial taking into account some parameters and a cyclic polynomial generator. Reed-Solomon codes receive this name in honour of its developers, Irving S. Reed and Gustave Solomon.

Reed-Solomon codes are a family of codes which can be defined with the parameters q, n and k: q is the length of the alphabet, n the block length (n<q) and k the message length (k<n). The rate R of the code is R=k/n. Cauchy- RS and Vandermonde-RS are specialized forms of Reed-Solomon codes commonly used to recover unreliable data.

According to the standards of IEEE and ITU-T, the codification that has to be done to send data through Ethernet is RS(255,239,8) where q=255, k=239 and n=8. Nevertheless there are other standards such as SMPTE St 2022-1 that instead of using this correcting function it uses XOR.

The number of errors that the RS codification is capable to correct is the following:

$t = \frac{D_{min}-1}{2}$ Where $D_{min} = N - K + 1$

Taking that into account and also the codification recommended by the standard, the proposed codification RS(255,239) would be able to correct 8 bytes per each packet of 239 bytes of payload + 16 bytes of redundancy.

To do the codification, first of all the generator polynomial has to be computed following the next equation:

$$g(x) = (x + \alpha)(x + \alpha^2) \dots (x + \alpha^{2t})$$

Once this generator polynomial is known, P(x), which are the protection words, can be found. To do so, it is need to know the V(x) (source information). Combining this together with the generator polynomial, the codewords to send can be found following the next equation:

$$Codewords = x^{2t} \cdot V(x) + \{V(x) \cdot x^{2t}\} mod\ g(x)$$

3. Methodology / project development:

3.1. Theoretical Model

FEC codifications (Forward Error Correction), as explained before, is a method that consists on adding redundancy to a message that has to be send in order to correct some errors that may occur when sending the information through non-reliable channels.

First of all, it is necessary to have some knowledge about FEC and specifically, RS (Reed-Solomon), which is the FEC codification recommended by the IEEE and ITU-T standards, is needed. According to previous points of this project, this codifications are done using a generator polynomial that can be get from the GF (Galois Field) taking into consideration the numbers of bits per word. Once known that, the extra bytes can be computed.

According to the standards, the codification recommended to implement FEC in messages that are sent in optical fibre using ud-WDM is RS(255,239). This means that each frame would contain 239 bytes of payload (the useful information desired to be sent) and 16 bytes of redundancy. This would lead that this encoding would have an overhead of 6.69%.

Taking into consideration some theoretical models, the aim of this codification that generates this overhead (so we are loosing useful bandwidth) is to enhance the BER (Bit Error Rate) at the receiver. According to these theoretical models, the improvements in the BER would be similar than the following:

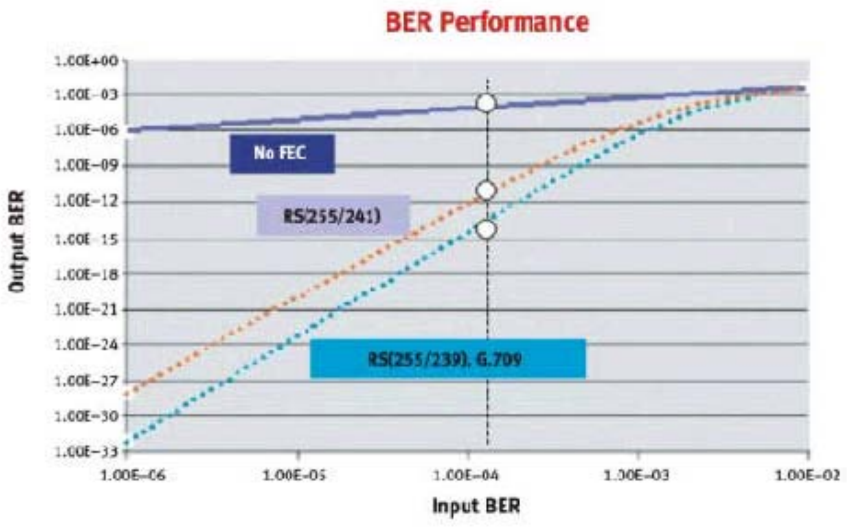


Figure 18: Effect of using FEC with various algorithms.

Figure 3.1. Theoretical win of BER using a system with FEC.

As can be seen, the BER should improve considerably when using FEC. Nowadays another FEC system called super-FEC is being studied, which consists on applying two times the FEC codification. However this codification is not being implemented yet.

3.2. Mathematical Model

To verify up to what to expect of a FEC codification system using the software proposed for this project, Matlab, the following model is realized as a first approximation of what can be obtained. The used model is the following:

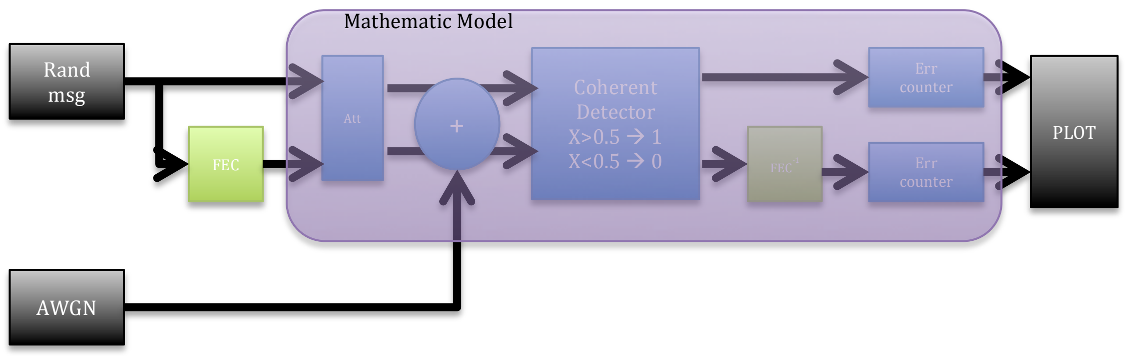


Figure 3.2. Mathematical model scheme.

This model consists on creating a large random message, then this message is copied and one of the copies is codified with FEC whereas the other not. Once done that, it is added some Gaussian noise to both messages. After that, the FEC code is decodified and it is computed the number of errors in both cases.

The detector, as is supposed that 1s and 0s would have the same probability to be transmitted, has the threshold at 0.5. The power of the Gaussian noise and the signal is controlled by an SNR (Signal to Noise Ratio) input. Modifying this input and repeating this function several times, a model that would be similar to the mathematical model should be obtained (Maybe a little worse taking into account that this receiver of this system is quite simple).

3.3. Real Model

To simulate exactly what would really happen, the mathematic model is not enough. It is necessary to implement another function to simulate the codifications done in the Ethernet system, 8b/10b. To do so, a couple of different models have been simulated. These models call different function according to what are simulating. The main functions used are FECcod, FECdec, enc8b10b, dec8b10b, bit2byte and byte2bit. All these functions, as well as the Matlab models of each of the systems, can be found on the ANNEXES.

3.3.1. Only FEC

The first system created calls the functions FECenc and FECdec to simulate a system that would only have the first codification. This system would not be a real simulation of what would happen on a real case but it serves as a first approximation of which would be the enhance of the BER on the receiver.

This system first of all imports a picture from your computer. It copies it and one of them is divided into 255 bytes packets, whereas the other in 239 bytes. This second one is encoded using the FEC codification RS(255,239). Then a noisy channel similar to the mathematical model is simulated. At the receiver is decoded the FEC pictures and are plotted both of them to compare which is the best. Also some information about the BER, the erroneous bytes and the erroneous pixels of each picture is shown.

The scheme of this system would be really similar with the mathematical model with the exception of the signal transmitted that is not random and also the way it plots the results is a little bit different too.

3.3.2. 8b/10b study

This second model is done with the aim of simulating the effect of 1 error in the 10 bits words that are sent through Ethernet. This error, when is decoded to the 8 bits word, can be worsening.

To simulate that possible effect, this model uses the functions enc8b10b and the dec10b8b. The way it works can be seen in the following scheme:

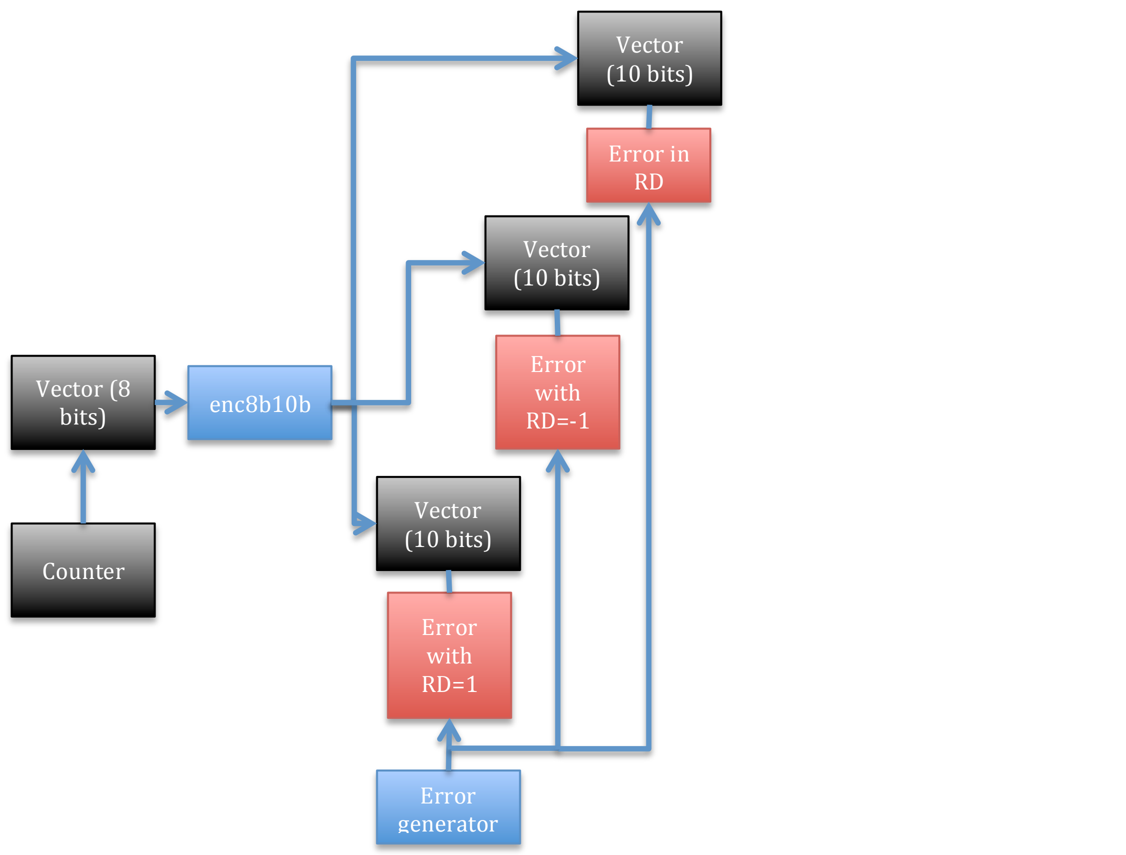


Figure 3.3. Scheme of the first part of the model of 8b/10b to test errors.

As it can be observed, there is a counter to generate all possible combinations of bits of the 10 bits words. These words are encoded and an error is artificially generated. The type of errors are classified according if it is a bit error with RD=1, RD=-1 or if the error is on the RD. This RD is the variable used to compute the disparity of the words, for further information seek for the chapter 2 of this project where it is explained more deeply the 8b/10b encoding and decoding.

Once done that, the 10b/8b decoding is performed in order to see the effects in any possible case. It is calculated the number of erroneous bits per every case and also the mean of them is computed in the three cases.

Moreover, as commented on the chapter 2 of this project, this encoding and decoding system can also detect errors, not correct but detect. Each time the error is detected a signal notifies the receiver of the error. This means that depending of the Internet protocol that the receiver is using, it could ask for a retransmission or it could ignore the error signal. In any case, these detected errors are also computed on this model as a percentage of the errors received.

3.3.3. FEC and 8b/10b study

The third and fourth models aim to simulate the nearest approximation to what will really happen. These systems both combine the RS FEC codification with the 8b/10b codification done by the Ethernet.

Both systems are quite similar, with the only difference of the order in which are combined these 2 functions.

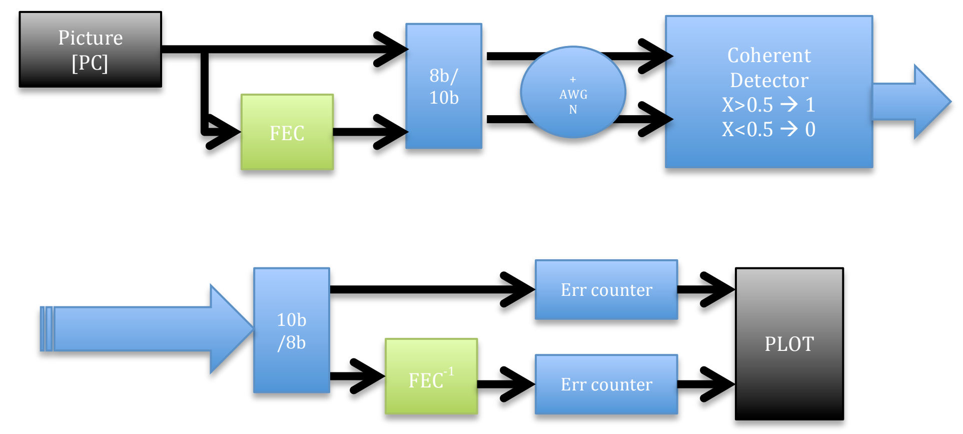


Figure 3.4. Third model scheme: FEC + 8b/10b.

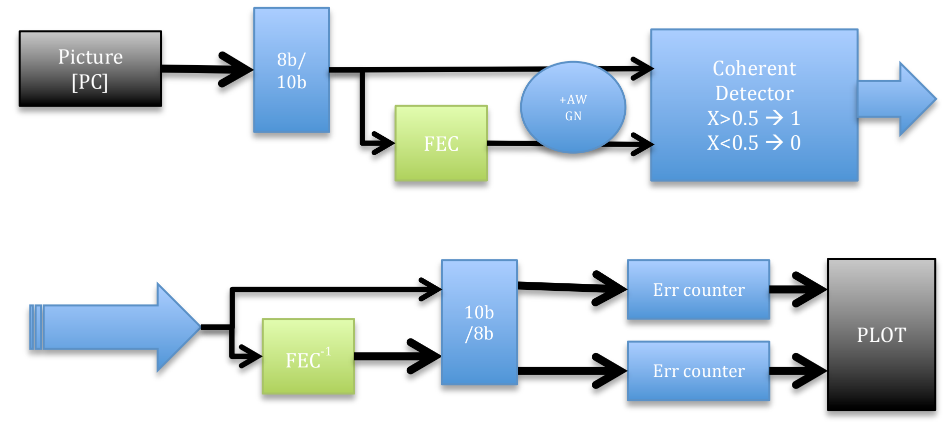


Figure 3.5. Fourth model scheme: 8b/10b + FEC.

The third model is the one recommended for the standards IEEE Section V clause 65.2.3., ITU-T G.975 and ITU-T G.984.2 and G.984.3, but both have interesting properties to take into consideration when choosing the system to implement. The advantage of the first model is that it ensures all the advantages of the 8b/10b codification, for example it ensures that the signal does not have a high continuous component and it helps with the clock synchronizer, whereas on the second case these properties may be lost.

The second model has the advantage that, if there is an error in a certain bit, when this word is decoded, it can lead to more erroneous bits than at the beginning. So if the errors were corrected before this 10b/8b decoding, it would decrease even more the BER. However the price to pay with this second model is that the signal to be transmitted wins an extra overhead, so its bandwidth useful is reduced.

4. Results

4.1. Theoretical model vs. mathematical model

As seen on the chapter before of this project, the mathematical model of how the FEC affects to the BER is not exactly the same compared as the theoretical model. So as in the model schemes can be observed some differences, so does with the results. The graph below shows the results obtained using the mathematical model (all the functions needed to simulate this can be found on the annexes).

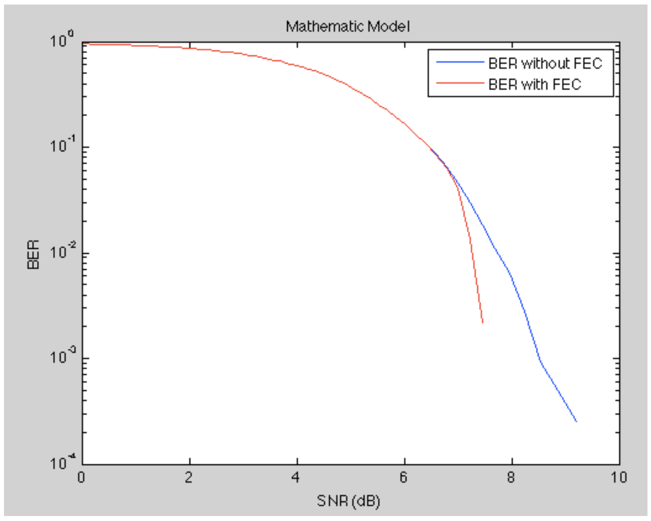


Figure 4.1. Results of the mathematical model.

As can be seen in this figure, it follows the same relation as the mathematical model seen in previous chapters, as it keeps the same BER for low SNRs but the BER using FEC decreases much faster than the one that is not using it.

According to the mathematical model, the results are not as good as on the theoretical one. This is due mainly to two reasons. The first one, the detector on the receiver is assuming that the bits have the same probability to appear, whereas in a better detector could approximate the threshold better. The second is that on a real case the signal would be codded with some more modulations, and this could worsen more one of the cases.

Apart from that, this model is not able to reach lower values of the BER. This is due to limitations on the simulation. As is increased the number of points, with this system the BER decreases really slow so to simulate lower BER values than the ones on this example would be really time-consuming for the computer.

4.2. 8b/10b and FEC results

The first and second models are in charge of simulating separately the effect of the two main codifications the signal is going to have: 8b/10b and FEC.

As it was explained on the previous chapter, to compute the average amount of errors produced by a single error on a word of 10 bits it has been using a couple of counters: one for generating any possible word and the other to place a single error on any possible position. The results obtained with the 8b/10b codification when there is 1 error on the channel behave as can be seen in the following picture:

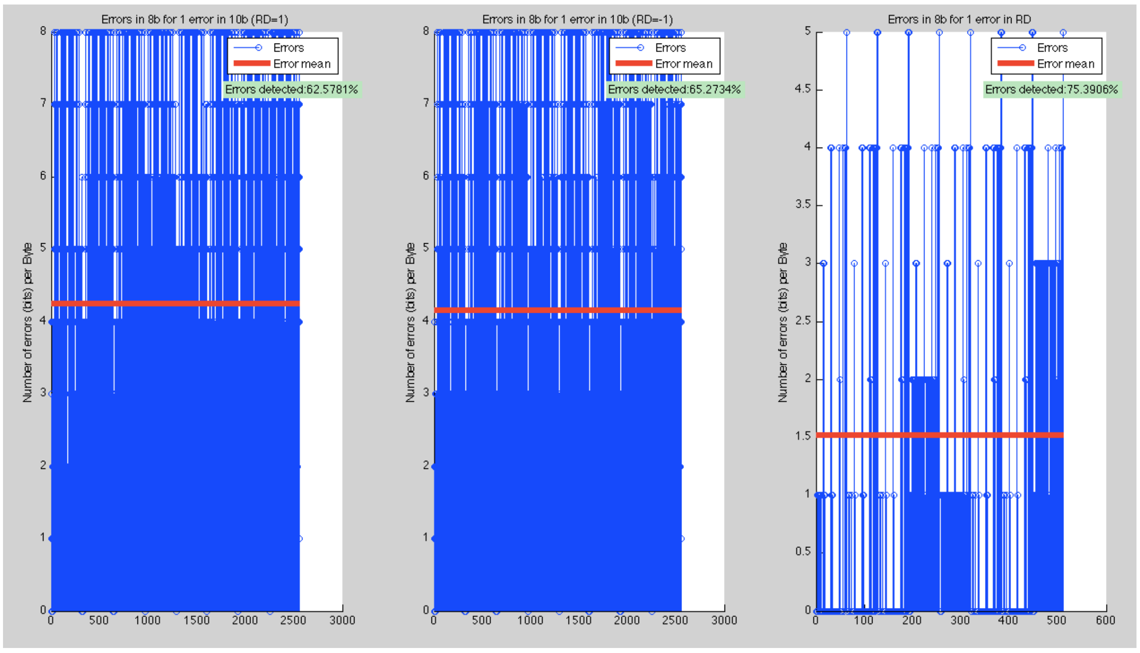


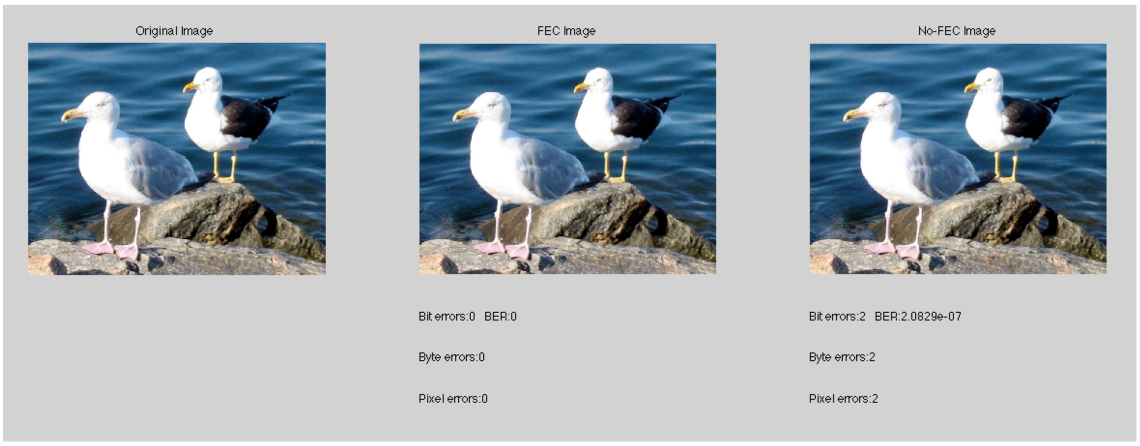
Figure 4.2. Error bits after doing 10b/8b decoding when there is 1 error per 10 bits word in the channel.

From this first simulation, some important information can be extracted. The first important thing before starting to analyse the results is to distinguish the first two graphs from the graph at the right. Whereas the first two are simulating what would happen when there is an error in any bit of the 10b, the third one is an error on RD, what means that this is not actually an error but is detected as so due to a previous error that has lead to a wrong calculation of the RD value.

The first important result is the mean of an error when the word is codded with 10 bits, when is reconverted has an average of 4 bits with error. This would lead to a worsening of the signal at the decoder. Contrasting that, the effect is not so harmful when the error is in the RD.

Another important result that has been obtained is that, although this codification worsens the BER, it is able to detect that an error has occurred with an average of 60%. This means that depending on the application and on the Internet protocol, it could ask for a retransmission if it is necessary. However, if the protocol used is like UDP, this signal is not useful any more.

Moving to the next model (the model that simulates the system only taking into account the FEC codification but not the 8b/10b), the FEC result is quite as it was expected to be according to the theoretical model and the mathematical model. Below here, three pictures can be seen: the left one is the original one, the centre one is the image at the receiver using FEC codification and the right one is the one at the receiver without using any codification. Also the error important parameters of each picture appear below it. These pictures have been taken for several different SNRs.

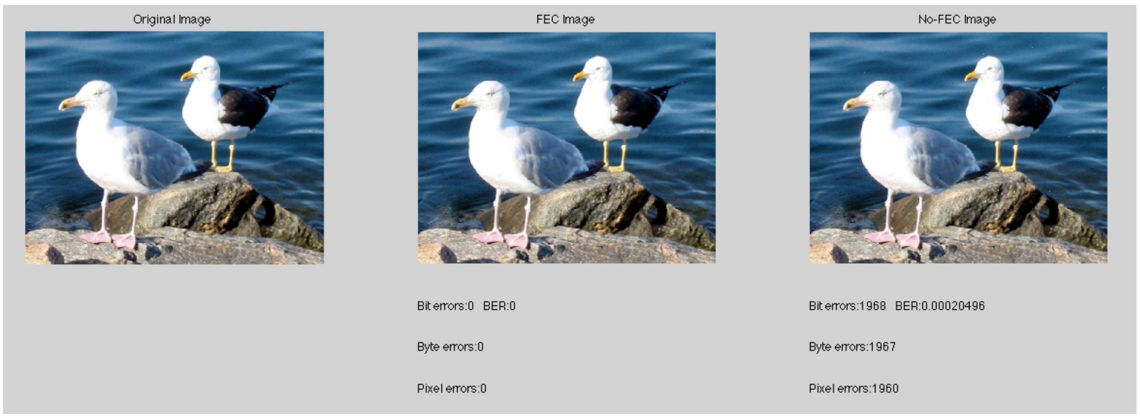


```
Bit errors:0   BER:0                Bit errors:2   BER:2.0829e-07

Byte errors:0                       Byte errors:2

Pixel errors:0                      Pixel errors:2
```

Figure 4.3. System with only FEC working with a SNR of 10dB.

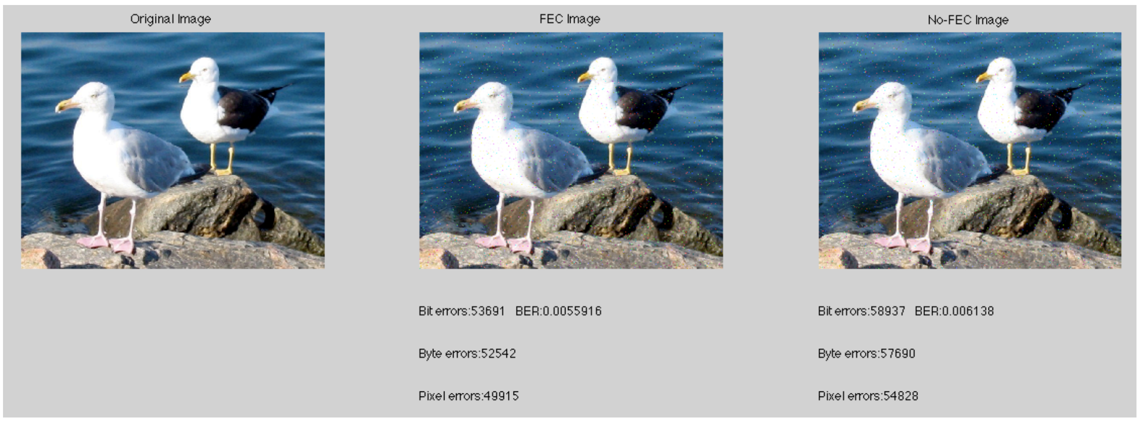


```
Bit errors:0   BER:0                Bit errors:1968   BER:0.00020496

Byte errors:0                       Byte errors:1967

Pixel errors:0                      Pixel errors:1960
```

Figure 4.4. System with only FEC working with a SNR of 8.5dB.



```
Bit errors:53691   BER:0.0055916    Bit errors:58937   BER:0.006138

Byte errors:52542                   Byte errors:57690

Pixel errors:49915                  Pixel errors:54828
```

Figure 4.5. System with only FEC working with a SNR of 7dB.

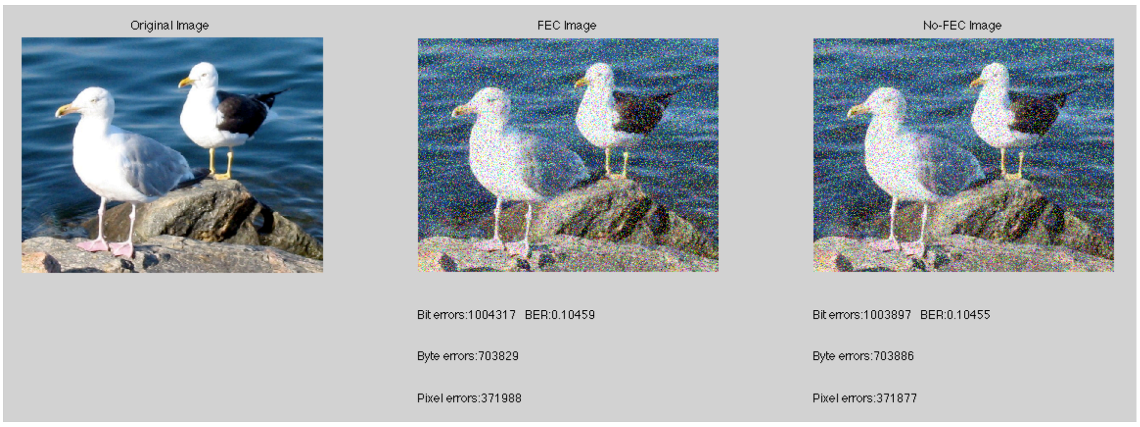


Figure 4.6. System with only FEC working with a SNR of 4dB.

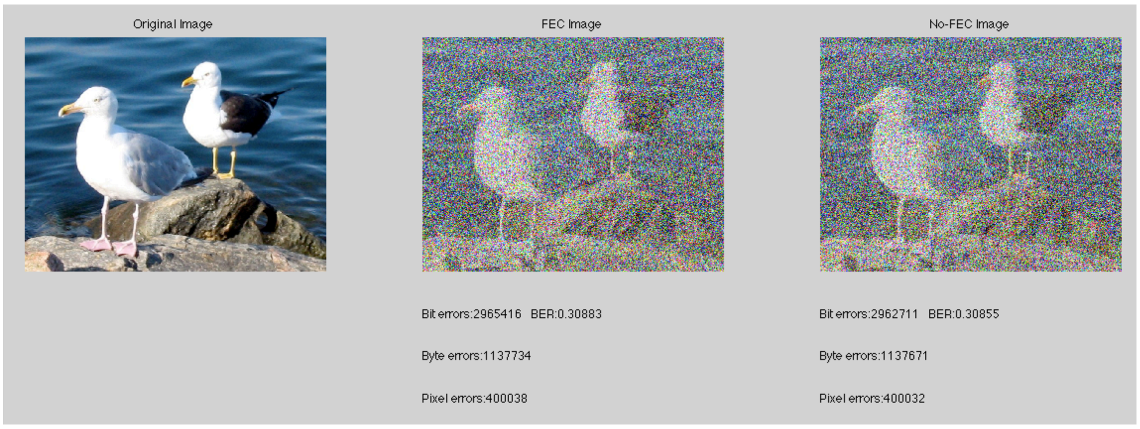


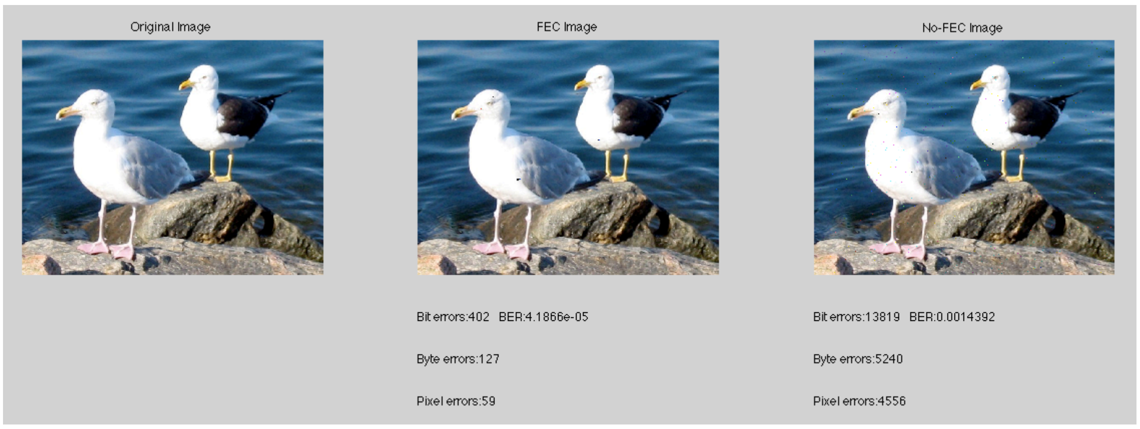
Figure 4.7. System with only FEC working with a SNR of 0dB.

As it can be observed in the previous figures, the results are similar to the mathematical model and the theoretical model. With SNRs higher, the FEC reduces more the BER in comparison with the case that is not using the FEC. Complementary, when the SNR is reduced, the difference between both cases is not so important.

Another thing that can be gathered from these results is that when the BER is the minimum required by the standard, it is even not noticeable by the human eye to see the errors.

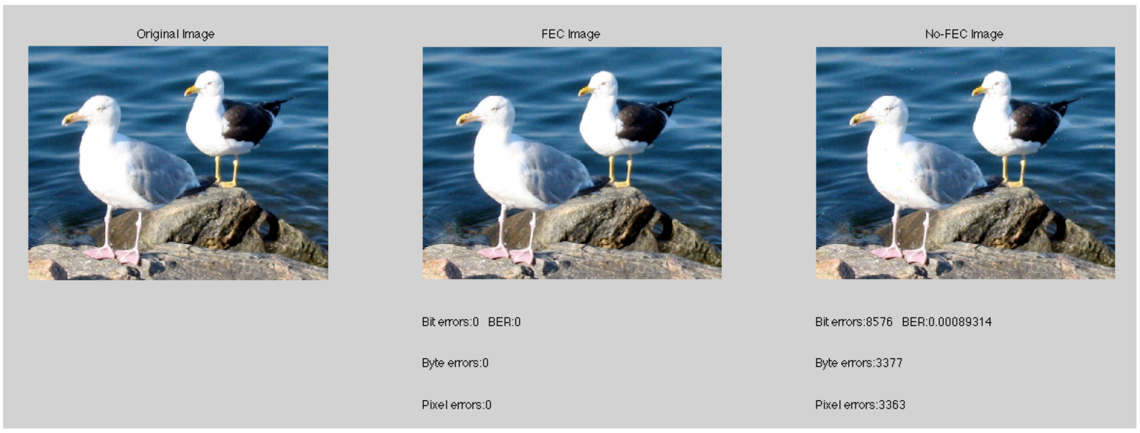
4.3. FEC + 8b/10b vs. 8b/10b + FEC

As it was explained on the chapter 3 of this project, there are two possibilities of connecting these two codifications. The figures below show the resultant picture of both cases for different SNRs. The top picture is with the system configuration of FEC + 8b/10b and the bottom one is the 8b/10b + FEC model.



Bit errors:402 BER:4.1866e-05
Byte errors:127
Pixel errors:59

Bit errors:13819 BER:0.0014392
Byte errors:5240
Pixel errors:4556

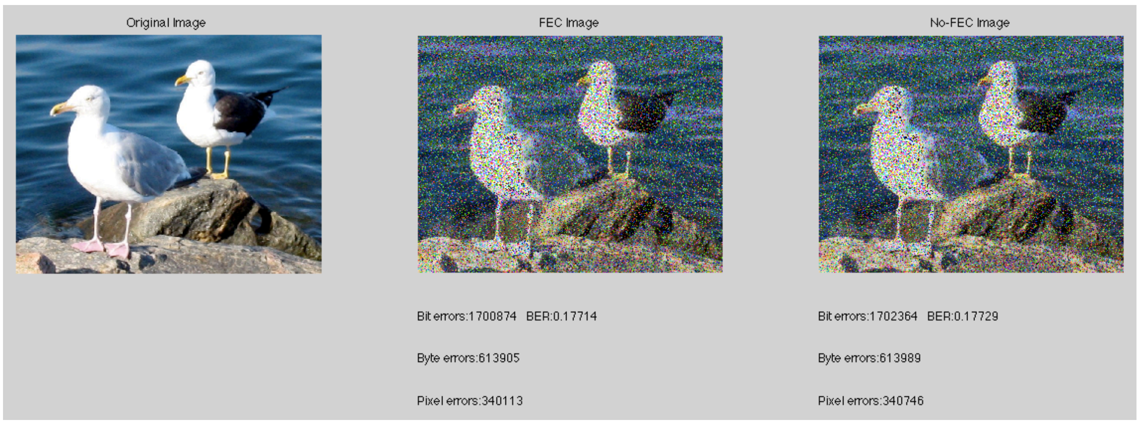


Bit errors:0 BER:0
Byte errors:0
Pixel errors:0

Bit errors:8576 BER:0.00089314
Byte errors:3377
Pixel errors:3363

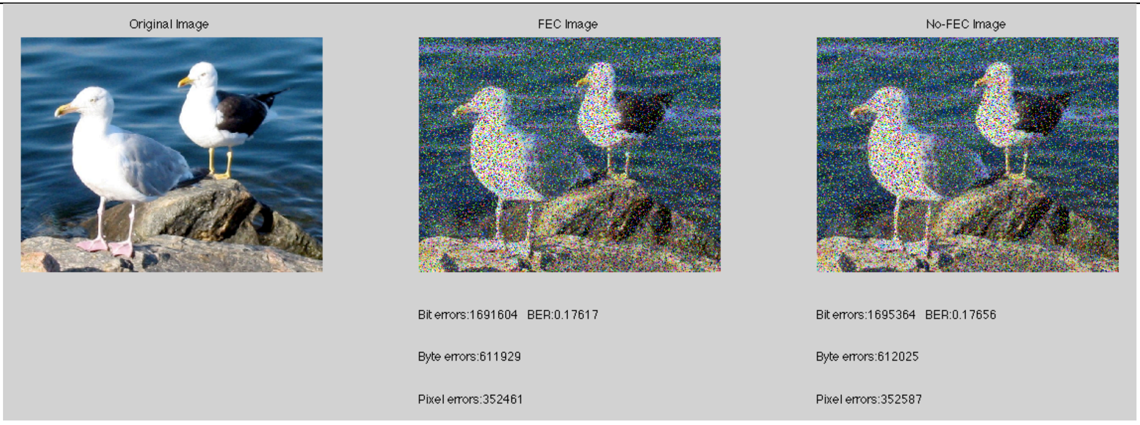
Figure 4.8.1. FEC + 8b/10b model working with a SNR of 8.5dB.

Figure 4.8.2. 8b/10b + FEC model working with a SNR of 8.5dB.



Bit errors:1700874 BER:0.17714
Byte errors:613905
Pixel errors:340113

Bit errors:1702364 BER:0.17729
Byte errors:613989
Pixel errors:340746



```
Bit errors:1691604   BER:0.17617
Byte errors:611929
Pixel errors:352461

Bit errors:1695364   BER:0.17656
Byte errors:612025
Pixel errors:352587
```

Figure 4.9.1. FEC + 8b/10b model working with a SNR of 5dB.

Figure 4.9.2. 8b/10b + FEC model working with a SNR of 5dB.

To be able to compare the results easily, the table below shows the comparison between the BER in three different models according to the SNR. The three models are the system using only FEC, the FEC + 8b/10b system and the 8b/10b + FEC system.

SNR (dB)	**BER**				
	-----------	**FEC**	**8b/10b**	**FEC + 8b/10b**	**8b/10b + FEC**
10dB	$2.0829*10^{-7}$	0	0.0014392	$1.6559*10^{-5}$	0
8.5dB	0.00020496	0	0.0014628	$4.1866*10^{-5}$	0
7dB	0.006138	0.0055916	0.026283	0.026258	0.024933
4dB	0.10455	0.10459	0.2661	0.26607	0.26564
0dB	0.3.855	0.3088	0.44263	0.44285	0.44237

Table 4.1. Comparison chart of the BER obtained in all the models.

As expected, when the FEC codifications are combined with the 8b/10b encoding and decoding of the Ethernet, the BER worsens considerably. The BER is a little bit better on the second case, but not much better.

Another result that can be observed in the pictures is the number of bytes and pixels with errors. As normally the pictures, unless the black and white ones, uses RGB, a pixel is constructed with 3 bytes. This mean that if any of these bytes has an error, the pixel has an error even the others are correct. So if there is some luck 3 wrong bytes could be 1 wrong pixel but it could also be 3. And the same applies from the bits to bytes, with the difference that due to the 8b/10b it is more probable to find wrong bits together.

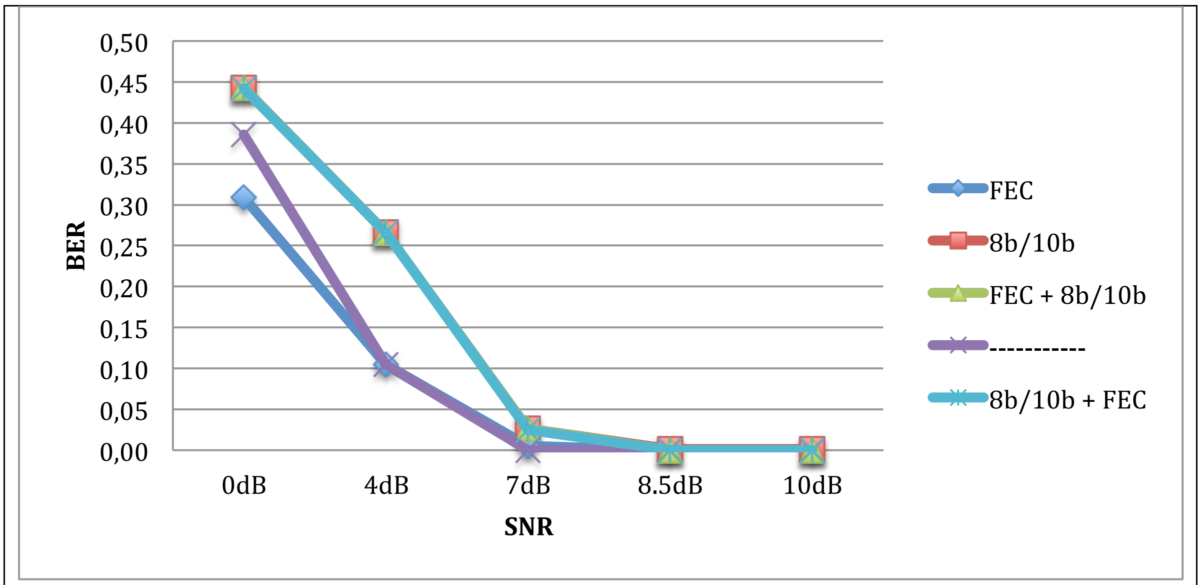


Figure 4.10. Comparison graph of all the models.

As can be observed, the configurations with FEC are far better than without FEC (in this graph cannot be notified so clear due to the reduced number of values as well as that the scale is in linear instead of logarithmic, what would help to analyse it better). Even though if it is look carefully to the values, and if a zoom is done in the values with the BER higher, it can be seen that the 8b/10b codification on its own is far lower than the other two.

Bandwidth effect:

The table below shows the bandwidth available of all the previous models that have been analysed:

	-----------	**FEC**	**8b/10b**	**FEC+8b/10b**	**8b/10b+FEC**
Rate at output of the receiver	1 Gb/s	1 Gb/s	1.25 Gb/s	1.25 Gb/s	1.3337 Gb/s
Rate of payload	1Gb/s	0.9331 Gb/s	1 Gb/s	0.9331 Gb/s	1 Gb/s

Table 4.2. Bandwidth effect of all the models analysed.

As can be seen, when the FEC is being used, the overhead due to its redundancy reduces the useful rate. This rate even worsens with the last configuration, though the payload rate is not reduced, maybe not all the systems are able to transmit at 1.3337 Gb/s because normally Ethernet is prepared to work at 1Gb/s or 1.25Gb/s if it is after the 8b/10b codification.

5. Budget

To compute with certain accuracy the cost of the system first of all is important to know the cost of all components. On the table below, an approximate cost of each of the lab components can be seen:

Device	Price
TP-Link MC200CM Gigabit Media Converter.	51.66 $ = 38.00 €
Unipro GbE So o	2560.28 $ = 1883.33 €
23818-S5-V2 (OLT1G)	299.89 $ = 220.60 €
Alcatel-Lucent1000BASE-T SFP	59.00 $ = 43.40 €
Adapter SC	1.31 € + IVA = 1.59 €
dapte LC	1.88 € + IVA = 2.28 €
Optical fibre (450x1.2m)	5.00 $ = 3.68 €
SC-SC cable	2.50 $ = 1.84 €
L -LC duplex cable	6.99 $ = 5.14 €
Optical attenuator (5 dB	18.10 $ = 13.31 €

Table 5.1. List of the laboratory items with each correspondent price.

Once known that, the cost of a prototype can be estimated, taking into account the elements that would be used.

Concept	Price
P ysical resources:	
2xMedia Conv rt r	2x38€=76€
2xSFP	2x43.40€=86.80€
4xSC-SC cable	4x1.84€=7.36€
1xOptical attenuator	1x13.31€=13.31€
1xTx/Rx/Analyser	1x1883.33€=1883.33€
Human resources:	

2xSenior engineer (50€/hour)	50€/hour*8hour*5days*20weeks=40000€
2xJunior engineer (20€/hour)	20€/hour*8hour*5days*20weeks=16000€
Software licences:	
2xMatlab licence	2x500 € = 1000€
<u>TO AL:</u>	59066.80 €

Table 5.2. Calculus of the prototype cost (physical resources + human resources + software licences).

Taking into account that this project is aimed to take advantage of the current installations of optical networks, the real cost would be only the money inverted on software and on human resource. This sums a really high percentage of the total, but as it has not physical requirements (because all the facilities are already there), can be inverted as a long-term project if the company does not have the money at the current moment.

But in order to ensure the real viability of the project it has to take into account the number of products that can be sold. The following equation shows the price that would have the product:

User Cost = Unit Cost (materials) + Unit Cost (fabrication) + R&D cost / (Number of units)

As said before, if it is used the current facilities the Unit Cost would be 0 and this R&D cost / (Number of units) would be translated in economical terms into an increase of the price of the service divided per the number of users. So with a number of users large enough this project can be viable.

6. Environment Impact

Building telecommunications systems has always had a huge impact against the environment. When it is necessary to be built an infrastructure like deploying and optical network, it sometimes has to be dig under earth which contributes on destroying the earth. Some others are not under earth, but what happens with those is that they destroy the "visual" environment and also have to be held somehow and these holding points also destroy the environment.

When there is an advance on technology, not only on the systems that have to be deployed but also on the technology used on the computers, this normally requires another installation.

The aim of this project is to use the current installations already deployed with some relatively new technology and at more distances. Due this project does not require any special facilities, it is allowing to use new features without creating any impact on the environment at the same time as saving the money that those modifications would have required.

Another thing to mention is that this system would allow HD streaming or videoconferences. This would indirectly reduce the need of having to commute to assist personally to those conferences which in environmental terms would be translated as a reduction of the pollution done by the travels.

On the other hand, this project is not totally eco-friendly because even though these telecommunication systems are already deployed as well as all the components of the system, it does not do anything to help the environment (apart from that indirect way of reducing the pollution). Creating these items has done certain impact on the environment, which is not repaired with this project.

To sum up, this project does not worsen the environment but it does not do anything to repair it directly (though it helps a little bit indirectly), so it is only "half-eco-friendly".

7. Conclusions and future development:

As a brief summary of the work done on the project, the use of ud-WDM requires some extra encoding methods to correct the errors obtained at the receiver; otherwise the distances it would be possible to send information using this method would be really short, or even impossible due to the BER floors imposed by the phase of the noise of the conventional lasers used in these new networks (DFBs with few MHz spectral linewidths).

This extra codification is called FEC and according to the standards, the most recommended to be used is RS(255,239). Another important aspect to remember is that Ethernet does another encoding called 8b/10b to reduce the continuous component of the signal and also to help the clock recovery at the receiver.

When combining both encodings, two different possible models appear: doing first the FEC encoding and afterwards the 8b/10b or just the opposite, start encoding with the 8b/10b and finish with the FEC.

Both models have several advantages from the other but also some drawbacks, but once seen the results the final recommendation is to encode first with the FEC and then the 8b/10b.

This has been developed in a simulating layer and it will soon be tested in a real case. However it would be really interesting this to be implemented in the real facilities that are already deployed, even if it is an application that allows to do video streaming codifying it with FEC or if it is a more universal update of all the system (which is quite improbable).

Bibliography:

[1] Norris, Mark. *Gigabit Ethernet technology and applications*. Boston: Artech House, 2003. Print.

[2] Held, Gilbert. *Ethernet networks: design, implementation, operation, management*. 4th ed. Chichester: J. Wiley, 2003. Print.

[3] Cunningham, David G., and William G. Lane. *Gigabit Ethernet networking*. Indianapolis, IN: Macmillan Technical Pub., 1999. Print.

[4] "Conectar MATLAB y Simulink a Hardware." *MATLAB and Simulink for Technical Computing*. <http://www.mathworks.es/>.

[5] "Coconut Project" *Home*. Web. <http://www.ict-coconut.eu/>.

[6] Lai, Cheah Cheng, P'Ng Won Tiang, Mohd Khazani Abdullah, Borhanuddin Mohd Ali, and Mohd Adzir Mahdi. "FEC Performance Analysis Based on Poisson and Bursty Error Patterns for SDH and OTN Systems." *Photonic Network Communications* 11.3 (2006): 265-270. Print.

[7] *IEEE Standard for Ethernet. Section III. Clause 36.2.4. 8B/10B transmission code*. IEEE Std 802.3-2012.

[8] *IEEE Standard for Information technology. Telecommunications and information exchange between systems Local and metropolitan area networks. Specific requirements. Part 3: Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications. Section V*. Clause 65.2.3. Forward Error Correction. IEEE Std 802.3-2008.

[9] ITU-T G.975. Error Correction in Reception for Submarine Systems. 2004.

[10] ITU-T G.984.2. Gigabit-capable Passive Optical Networks (G-PON). Physical Media Dependent (PMD) layer specification. 2003.

[11] ITU-T G.984.3. Gigabit-capable Passive Optical Networks (G-PON). Transmission convergence layer specification. 2008.

[12] SMPT 2022-1. Forward Error Correction for Real-Time Video/Audio Transport Over IP Networks. 2007.

[13] N. Kiran Babu, P. S. Srinivas Babu. "Design of Physical Coding Sublayer using 8b/10b Algorithm." *International Journal of Recent Technology and Engineering (IJRTE).* May (2013) Print.

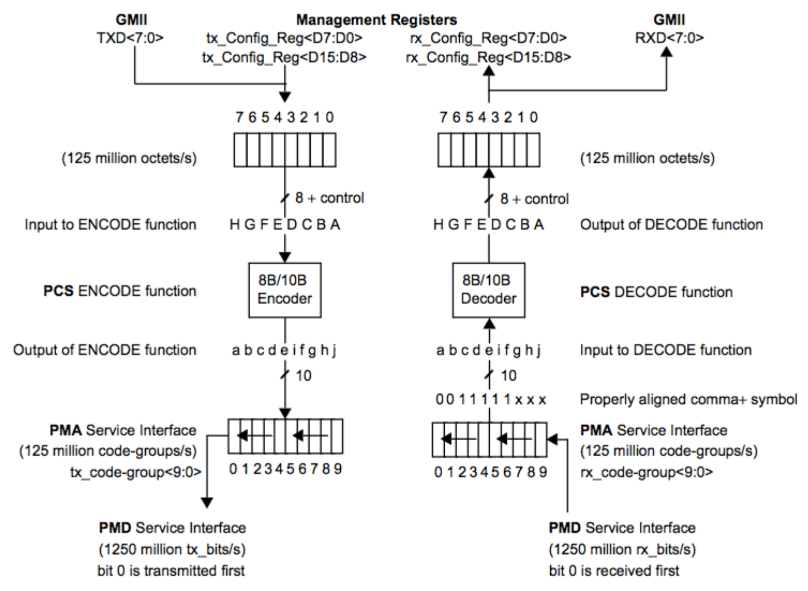
Annexes:

ANNEX 1. STANDARD SUMMARY TABLES:

Below here there is a brief summary of the standards that are somehow related with this project (Ethernet, FEC…) for further information seek for the standard named at the left side of the tables:

SUMMARY STANDARD CHART (8b10b)

IEEE

Section III. Clause 36.2.4.	• 8B/10B transmission codes specified in the standard have high transition density, run-length limited and have DC balance. • The system of 8B/10B codification and decodification is illustrated in the following figure:  GMII TXD<7:0> Management Registers tx_Config_Reg<D7:D0> tx_Config_Reg<D15:D8> rx_Config_Reg<D7:D0> rx_Config_Reg<D15:D8> GMII RXD<7:0> 7 6 5 4 3 2 1 0 (125 million octets/s) 8 + control Input to ENCODE function H G F E D C B A Output of DECODE function PCS ENCODE function 8B/10B Encoder 8B/10B Decoder PCS DECODE function Output of ENCODE function a b c d e i f g h j Input to DECODE function 10 0 0 1 1 1 1 1 x x x Properly aligned comma+ symbol PMA Service Interface (125 million code-groups/s) tx_code-group<9:0> rx_code-group<9:0> 0 1 2 3 4 5 6 7 8 9 PMD Service Interface (1250 million tx_bits/s) bit 0 is transmitted first (1250 million rx_bits/s) bit 0 is received first **Figure 36–3—PCS reference diagram** Figure x.1. Physical codding sub layer diagram. • The non-codified octets only have a control variable Z that can be D or K, each of it refers to a different code-group. • Non-codified word: A,B,C,D,E,F,G,H. • Codified word: a,b,c,d,e,i,f,g,h,. • /Dx.y/ for 256 valid data. • /Kx.y/ for special control. • The first block transmitted is the special code-group to distinguish the ordered_set. • The first code-group of each multi-code-group (group of codes-groups that are sent together) is transmitted in an even number code-group position. • The contents are transmitted sequentially starting with the SPD, then the packets and then the EPD. • The running disparity value has to be used and is re-calculated for every code-group transmitted. • The running disparity is also contained on the receiver, and if it

	matches with the code-group received is considered valid and decoded, otherwise it is considered invalid. • However this does not mean that the error is in that code-group, maybe it has happened before and computed a wrong running disparity. • Even with errors the running disparity is kept computed. • The number of invalid codes is related with the BER and counting the number of invalid groups can perform error monitoring.

SUMMARY STANDARD CHART (FEC)

IEEE

Section V. Clause 65. 2.3. (2005)	• Optional. • BER objective 10^{-12} at PCS. • BER objective 10^{-4} at FEC sublayers.
Section V. Clause 65. 2.3.1. (2005)	• FEC specifications in ITU-T G.975 • FEC code used is a linear cyclic bloc code RS(255,239,8) over a GF(2^8). (Cyclic means that a cyclic shift of any codeword is another codeword). • RS based on the generating polynomial $G(x) = \prod_{i=0}^{15}(x - \alpha^i)$ where α is 0x02 and root of $x^8+x^4+x^3+x^2+1$
Section V. Clause 65. 2.3.2. (2005)	• The 16 parity symbols are added at the end of the 239 symbols block. • If the last block is shorter the data is on D_{238} to D_r and the rest of the symbols are filled with 0. The symbols are sent without the 0s and at the receiver are added again to do the recovery. • Ethernet frame markers are not protected by FEC. Special start /S_FEC/ and stop /T_FEC/ markers are added at the beginning and end of the FEC coded frame.
Section V. Clause 65. 2.3.3.	• The following figures show the transmitter and receiver schemes:

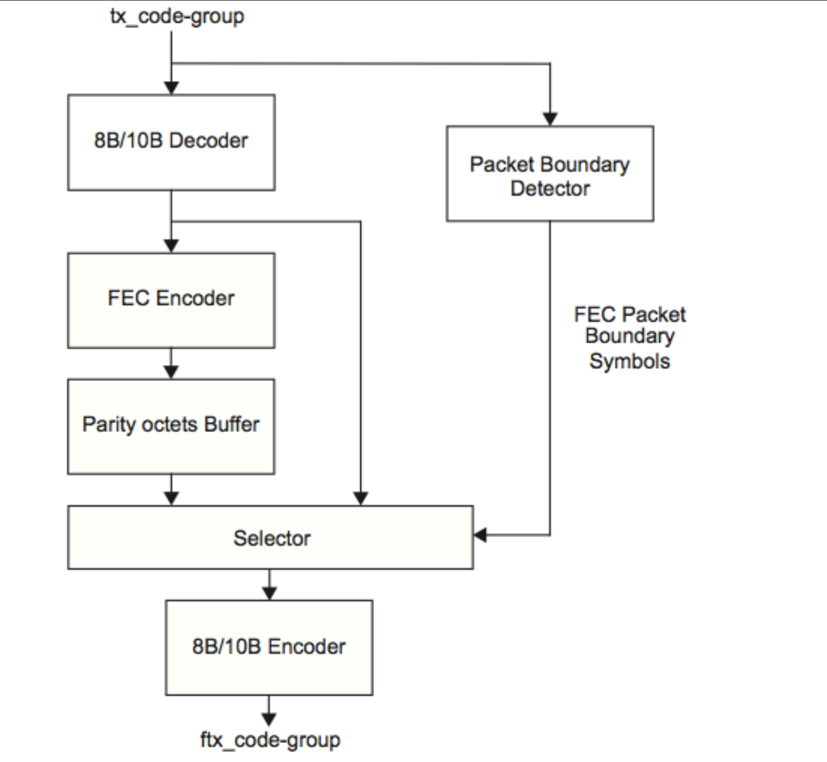


Figure x.2. Transmit block diagram.

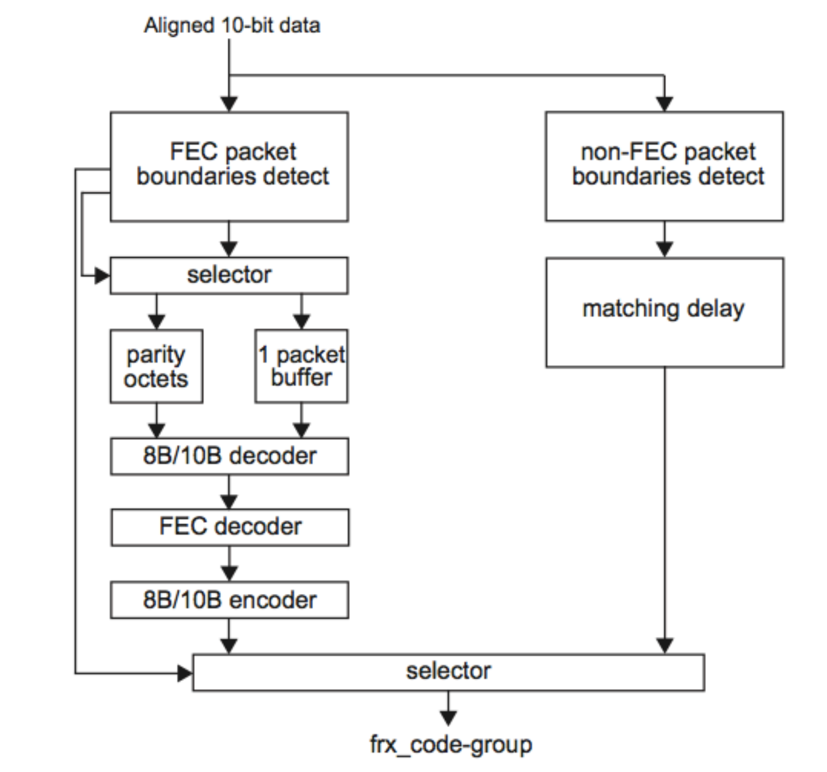
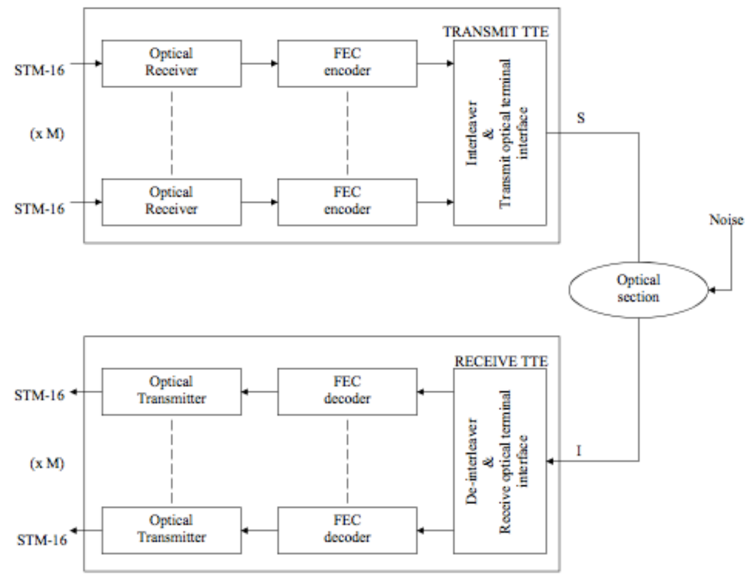
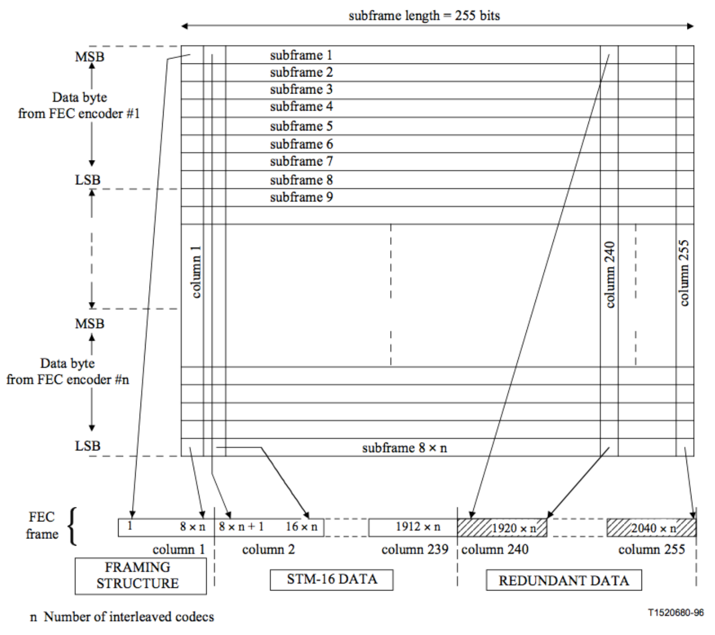


Figure x.3. Receiver data block diagram.

ITU

ITU-T G.975 (2000)	• This figure shows a system using FEC:  TRANSMIT TTE STM-16 (x M) STM-16 Optical Receiver FEC encoder Optical Receiver FEC encoder Interleaver & Transmit optical terminal interface S Noise Optical section RECEIVE TTE STM-16 (x M) STM-16 Optical Transmitter FEC decoder Optical Transmitter FEC decoder De-interleaver & Receive optical terminal interface I Figure x.4. Block diagram of a submarine system using FEC. • $BER_{input}=BER_c+BER_{output}$ (BER_c are the number of bits corrected). • The system works properly for $10^{-3}<BER_{input}<10^{-15}$. • Blocks of N = K + R where N is the number of symbols, K the symbols with information and R the redundancy. • Codification used: RS(255,239) with 8 bits words. • When applying interleaving the frame structure is the following:  subframe length = 255 bits MSB Data byte from FEC encoder #1 LSB MSB Data byte from FEC encoder #n LSB subframe 1 subframe 2 subframe 3 subframe 4 subframe 5 subframe 6 subframe 7 subframe 8 subframe 9 column 1 column 240 column 255 subframe 8 × n FEC frame 1 8 × n 8 × n + 1 16 × n 1912 × n 1920 × n 2040 × n column 1 column 2 column 239 column 240 column 255 FRAMING STRUCTURE STM-16 DATA REDUNDANT DATA n Number of interleaved codecs T1520680-96 Figure x.5. FEC frame construction. • Each subframe has the following structure:

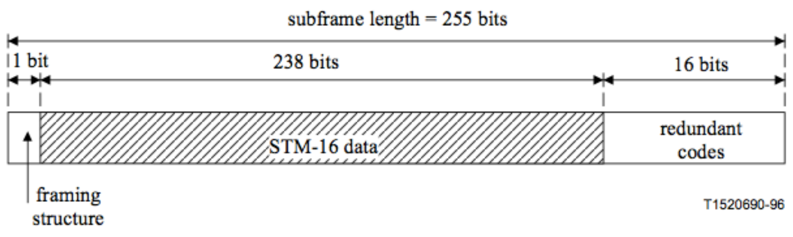


Figure x.6. Subframe of the FEC frame.

- Scrambling not compulsory.
- Theoretical results:

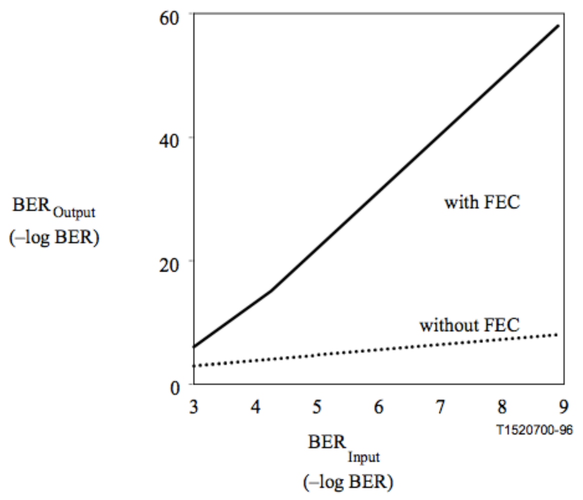


Figure x.7. Theoretical BERs.

BER_{input}	BER_{output}
10^{-4}	$5 \cdot 10^{-15}$
10^{-5}	$6.3 \cdot 10^{-24}$
10^{-6}	$6.4 \cdot 10^{-33}$

Table x.1. Theoretical BERs.

ITU-T REC G.975.1 (2004)

- For DWDM use super FEC instead of normal FEC.
- The following figure shows a super FEC scheme:

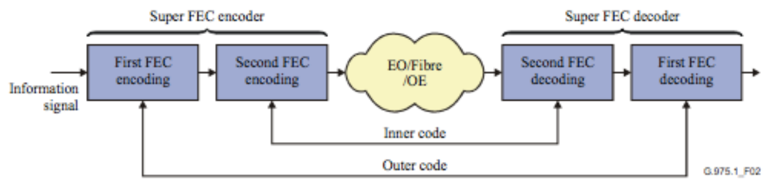


Figure x.8. Outer code and inner code of super FEC.

- Super FEC has 5.6dB of Net Codding Gain at 10^{-12} BER_{output} (Net Coding Gain is the gain between BER_{input} and BER_{output}).
- In the next figure can be found the equations of the BERs and Net Coding Gain:

$$Coding\_Gain = 20\log_{10}\left[erfc^{-1}\left(2B_{ref}\right)\right] - 20\log_{10}\left[erfc^{-1}\left(2B_{in}\right)\right] \quad (\mathrm{dB})$$

$$Net\_Coding\_Gain = 20\log_{10}\left[erfc^{-1}\left(2B_{ref}\right)\right] - 20\log_{10}\left[erfc^{-1}\left(2B_{in}\right)\right] + 10\log_{10} R \quad (\mathrm{dB})$$

$$Q = \frac{\mu_1 - \mu_0}{\sigma_1 + \sigma_0} \qquad BER = \frac{1}{2} erfc\left(\frac{Q}{\sqrt{2}}\right)$$

Figure x.9. Net equations.

- This last figure is a comparison between the BERs with no FEC, with normal FEC and with super FEC:

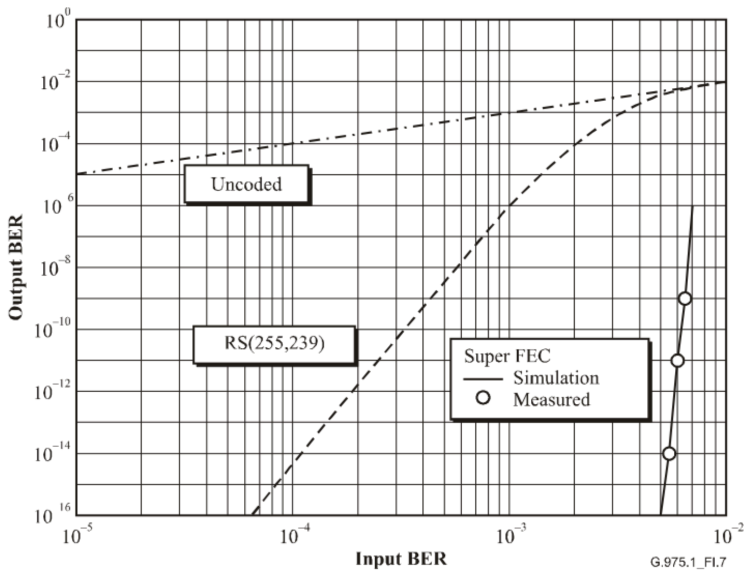


Figure x.10. BER characteristics.

- The redundancy ratio of super FEC is 24.48%.

ITU-T G.984.2. (2003)	• In the following table can be seen the parameters required for 1Gbit/s connections:

Items	Unit	Single fibre			Dual fibre		
		ONU Transmitter (optical interface O_{ru})					
Nominal bit rate	Mbit/s	1244.16			1244.16		
Operating wavelength	nm	1260-1360			1260-1360		
Line code	–	Scrambled NRZ			Scrambled NRZ		
Mask of the transmitter eye diagram	–	Figure 3			Figure 3		
Maximum reflectance of equipment, measured at transmitter wavelength	dB	less than –6			less than –6		
Minimum ORL of ODN at O_{ru} and O_{rd} (Notes 1 and 2)	dB	more than 32			more than 32		
ODN Class		A	B	C	A	B	C
Mean launched power MIN	dBm	–3 (Note 5)	–2	+2	–3 (Note 5)	–2	+2
Mean launched power MAX	dBm	+2 (Note 5)	+3	+7	+2 (Note 5)	+3	+7
Launched optical power without input to the transmitter	dBm	Less than Min sensitivity –10			less than Min sensitivity –10		
Maximum Tx Enable (Note 3)	bits	16			16		
Maximum Tx Disable (Note 3)	bits	16			16		
Extinction ratio	dB	more than 10			more than 10		
Tolerance to transmitter incident light power	dB	more than –15			more than –15		
MLM Laser – Maximum RMS width	nm	(Note 5)			(Note 5)		
SLM Laser – Maximum –20 dB width (Note 4)	nm	1			1		
If SLM Laser – Minimum side mode suppression ratio	dB	30			30		
Jitter transfer	–	Figure 4			Figure 4		
Jitter generation from 4.0 kHz to 10.0 MHz	UI p-p	0.33			0.33		
		OLT Receiver (optical interface O_{lu})					
Maximum reflectance of equipment, measured at receiver wavelength	dB	less than –20			less than –20		
Bit error ratio	–	less than 10^{-10}			less than 10^{-10}		
ODN Class		A	B	C	A	B	C
Minimum sensitivity	dBm	–24 (Note 6)	–28	–29	–24 (Note 6)	–28	–29
Minimum overload	dBm	–3 (Note 6)	–7	–8	–3 (Note 6)	–7	–8

Table x.2. Optical interface parameters for 1Gbit/s upstream.

- The next figure shows the definition of the optical gain G using FEC:

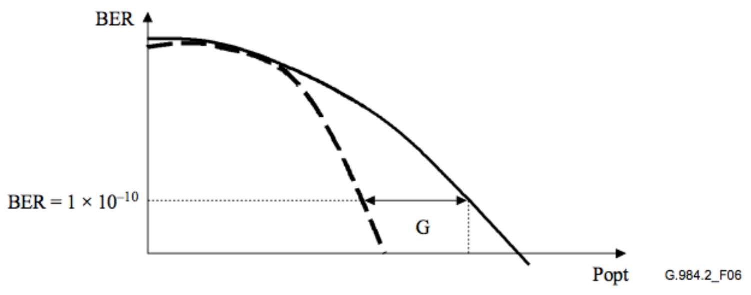


Figure x.11. Effective optical gain G achieved with FEC.

ITU-T G.984.2. (2006)

- In the following table there are the loss budget for G-PON systems:

Items	Unit	Single fibre
Minimum optical loss at 1490 nm	dB	13
Minimum optical loss at 1310 nm	dB	13
Maximum optical loss at 1490 nm	dB	28
Maximum optical loss at 1310 nm	dB	28

Table x.3. Loss budget for the G-PON system.

ITU-T G.984.3. (2004)

- RS(255,239).
- In the following figure there is a FEC frame with the last word shorter than the others:

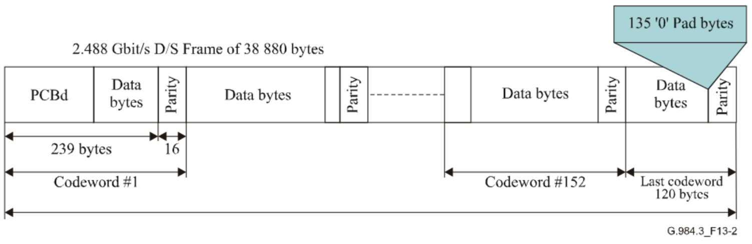


Figure x.12. D/S frame with FEC and last codeword shorter.

- In the IDEN field there is a FEC indication bit (0 if FEC off and 1 if FEC on).

SMPTE

St 2022-1 (2007)

- RTP shall be required.
- The error correcting function is XOR.
- FEC header has 12 bits and is like that:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         SNBase low bits         |        Length Recovery         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|E| PT recovery |                    Mask                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          TS recovery                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|N|D|type |index|    Offset     |      NA       |SNBase ext bits|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure x.13. Definition of the FEC header.

ANNEX 2. INITIAL WORK PACKAGES:

Project: IFECOEC	WP ref: 1
Major constituent: Standards and Current System Study	Sheet 1 of 1
Short description: The current system must be studied carefully and in detail of all the components.	Planned start date: 10/03/2014 Planned end date: 06/04/2014

The standards related with the system to implement must be studied as well.	Start event: Opt. Tx study End event: FEC for fibre study	
Internal task T1: General study (Optical Ethernet Tx, EPON and 10GPON, 8b10b coding…). Internal task T2: FEC for optical fibre study.	Deliverables:	Dates: T1: 10/03/14 to 23/03/14 T2: 24/03/14 to 13/04/14

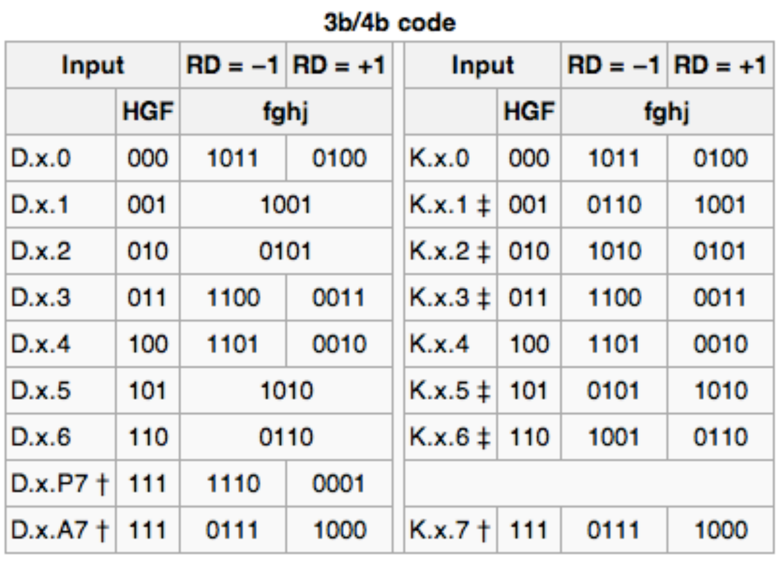
Project: IFECOEC	WP ref: 2	
Major constituent: Design and Simulation	Sheet 1 of 1	
Short description: The aim in this work package is to create some different designs that accomplish the requirements of the system. It also aims to simulate each of the designs.	Planned start date: 07/04/14 Planned end date: 30/04/14 Start event: Design systems End event: CDR	
Internal task T1: Design different possible systems. Internal task T2: Simulate and choose the best system.	Deliverables: T2: System chosen info. + CDR	Dates: T1: 01/04/14 to 20/04/14 T2: 14/04/14 to 30/04/14

Project: IFECOEC	WP ref: 3	
Major constituent: Implementation (SW)	Sheet 1 of 1	
Short description: The system chosen after being simulated must be implemented using a matlab toolbox. Evaluation of its implementation with FPGA for real time operation.	Planned start date: 28/04/14 Planned end date: 25/05/14 Start event: CDR End event: Implementation	
Internal task T1: Implement the FEC code of the system.	Deliverables: T1: FEC implementation	Dates: T1: 28/04/14 to 25/05/14

Project: IFECOEC	WP ref: 4	
Major constituent: Test (whole system as a prototype)	Sheet 1 of 1	
Short description: Test the implemented design on its own. Test the implemented design with the whole system. Integrate the design to the whole system and test it again, demonstrating it with multimedia content.	Planned start date: 26/05/14 Planned end date: 6/07/14 Start event: FEC test End event: FEC integration	
Internal task T1: Tests of FEC alone and in the system. Internal task T2: FEC integration.	Deliverables: T1: FEC results. T2: System demonstration.	Dates: T1: 26/05/14 to 29/06/14 T2: 23/06/14 to 06/05/14

ANNEX 3. 8b-10b ENCODING TABLES:

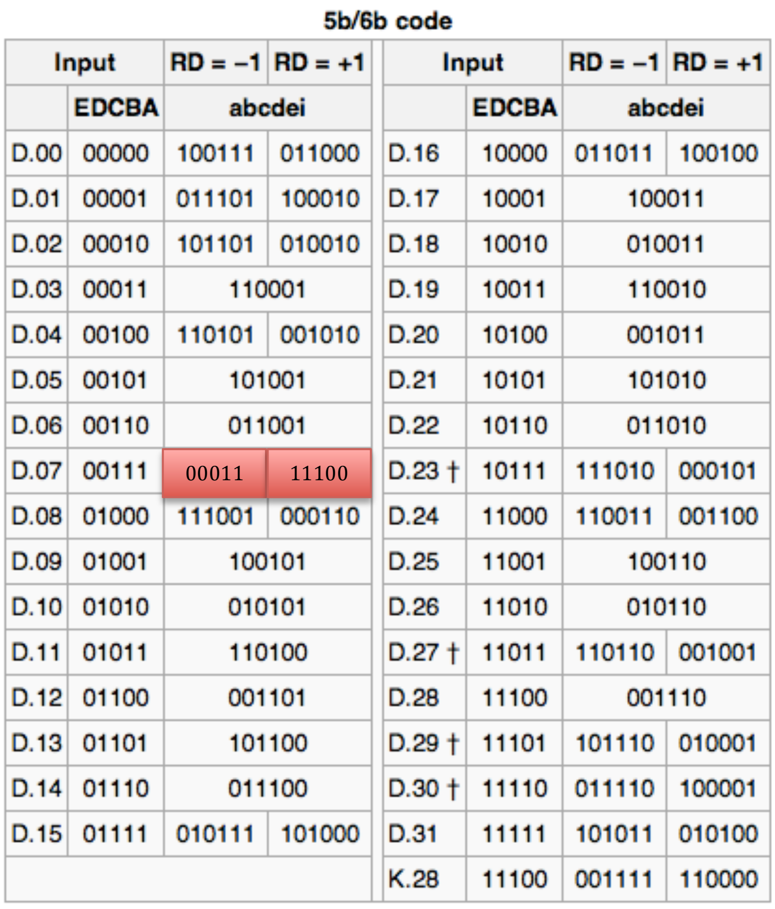
The 8b-10b encoding is divided into 2 different encoding: the lowers 5 bits using the first table and the top 3 using the second table. These tables can be found through the Internet to avoid having to use the longest tables of the standards, however it has an important mistake (marked in red on the table) because it has a couple of values swapped from the disparity:



3b/4b code

Input		RD = −1	RD = +1	Input		RD = −1	RD = +1
	HGF	fghj			HGF	fghj	
D.x.0	000	1011	0100	K.x.0	000	1011	0100
D.x.1	001	1001		K.x.1 ‡	001	0110	1001
D.x.2	010	0101		K.x.2 ‡	010	1010	0101
D.x.3	011	1100	0011	K.x.3 ‡	011	1100	0011
D.x.4	100	1101	0010	K.x.4	100	1101	0010
D.x.5	101	1010		K.x.5 ‡	101	0101	1010
D.x.6	110	0110		K.x.6 ‡	110	1001	0110
D.x.P7 †	111	1110	0001				
D.x.A7 †	111	0111	1000	K.x.7 †	111	0111	1000

Table x.4. 3b/4b encoding table.

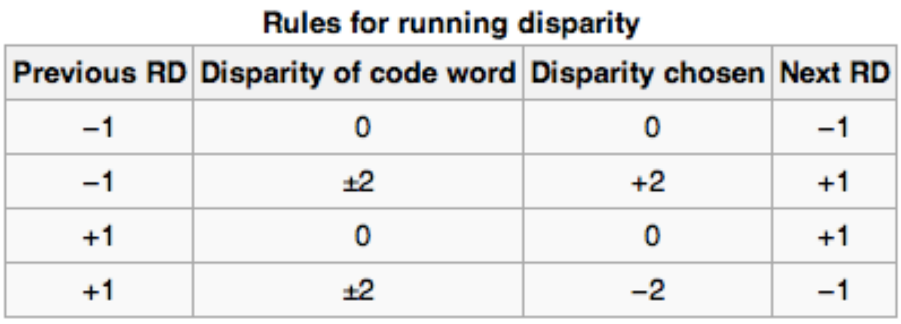


5b/6b code

Input		RD = −1	RD = +1	Input		RD = −1	RD = +1
	EDCBA	abcdei			EDCBA	abcdei	
D.00	00000	100111	011000	D.16	10000	011011	100100
D.01	00001	011101	100010	D.17	10001	100011	
D.02	00010	101101	010010	D.18	10010	010011	
D.03	00011	110001		D.19	10011	110010	
D.04	00100	110101	001010	D.20	10100	001011	
D.05	00101	101001		D.21	10101	101010	
D.06	00110	011001		D.22	10110	011010	
D.07	00111	00011	11100	D.23 †	10111	111010	000101
D.08	01000	111001	000110	D.24	11000	110011	001100
D.09	01001	100101		D.25	11001	100110	
D.10	01010	010101		D.26	11010	010110	
D.11	01011	110100		D.27 †	11011	110110	001001
D.12	01100	001101		D.28	11100	001110	
D.13	01101	101100		D.29 †	11101	101110	010001
D.14	01110	011100		D.30 †	11110	011110	100001
D.15	01111	010111	101000	D.31	11111	101011	010100
				K.28	11100	001111	110000

Table x.5. 5b/6b encoding table.

The RD (Running Disparity) depends on how many 1s or 0s in a row had been and also depends on the last RD:



Rules for running disparity

Previous RD	Disparity of code word	Disparity chosen	Next RD
−1	0	0	−1
−1	±2	+2	+1
+1	0	0	+1
+1	±2	−2	−1

Table x.6. Rules for running disparity.

ANNEX 4. MATLAB FUNCTIONS:

In this annex it can be found all the Matlab functions required to do all the simulations explained on the chapter 3 and 4 of this project.

This first function is the function that converts a vector with the bits into its byte value:

```
function B = bit2byte(b)

B=0;
for i=1:8
    B=B+b(9-i)*(2^(i-1));
end
```

The following function is in charge of converting bytes into a vector of the corresponding bits:

```
function b = byte2bit(B)

b=zeros(1,8);
b(8)=mod(B,2);
aux=floor(B/2);
b(7)=mod(aux,2);
aux=floor(aux/2);
b(6)=mod(aux,2);
aux=floor(aux/2);
b(5)=mod(aux,2);
aux=floor(aux/2);
b(4)=mod(aux,2);
aux=floor(aux/2);
b(3)=mod(aux,2);
aux=floor(aux/2);
b(2)=mod(aux,2);
b(1)=floor(aux/2);
```

The function below does the FEC encoding to a certain input, taking into account the length of the symbols as an input too:

```
function D255 = FECcod(D239,M)
% D239: data to encode.
%M: number of bits per word.
%RS(239,255)

Daux=gf(D239,M);
D255aux=rsenc(Daux',255,239);
D255=(double(D255aux.x))';
```

This function reverses the previous function and has also as inputs the length of the words as well as what is desired to decode:

```
function D239 = FECdec(D255,M)
% D255: data to decode.
%M: number of bits per word.
%RS^-1(239,255)

Daux=gf(D255,M);
D239aux=rsdec(Daux',255,239);
D239=(double(D239aux.x))';
```

The following function does the 8b/10b encoding, the inputs are a single value in bytes and is reconverted to its corresponding value of 10 bits in a vector:

```
function [b10,RD] = enc8b10b(b8,RD)

aux=byte2bit(b8);
H=aux(1);
G=aux(2);
F=aux(3);
E=aux(4);
D=aux(5);
C=aux(6);
B=aux(7);
A=aux(8);
```

```
EDCBA=[E,D,C,B,A];
HGF=[H,G,F];

%5b6b
if RD==1
    if EDCBA==[0,0,0,0,0]
        abcdei=[0,1,1,0,0,0];
        RD=-1;
    elseif EDCBA==[0,0,0,0,1]
        abcdei=[1,0,0,0,1,0];
        RD=-1;
    elseif EDCBA==[0,0,0,1,0]
        abcdei=[0,1,0,0,1,0];
        RD=-1;
    elseif EDCBA==[0,0,0,1,1]
        abcdei=[1,1,0,0,0,1];
        RD=1;
    elseif EDCBA==[0,0,1,0,0]
        abcdei=[0,0,1,0,1,0];
        RD=-1;
    elseif EDCBA==[0,0,1,0,1]
        abcdei=[1,0,1,0,0,1];
        RD=1;
    elseif EDCBA==[0,0,1,1,0]
        abcdei=[0,1,1,0,0,1];
        RD=1;
    elseif EDCBA==[0,0,1,1,1]
        abcdei=[0,0,0,1,1,1];
        RD=-1;
    elseif EDCBA==[0,1,0,0,0]
        abcdei=[0,0,0,1,1,0];
        RD=-1;
    elseif EDCBA==[0,1,0,0,1]
        abcdei=[1,0,0,1,0,1];
        RD=1;
    elseif EDCBA==[0,1,0,1,0]
        abcdei=[0,1,0,1,0,1];
        RD=1;
    elseif EDCBA==[0,1,0,1,1]
        abcdei=[1,1,0,1,0,0];
        RD=1;
    elseif EDCBA==[0,1,1,0,0]
        abcdei=[0,0,1,1,0,1];
        RD=1;
    elseif EDCBA==[0,1,1,0,1]
        abcdei=[1,0,1,1,0,0];
        RD=1;
    elseif EDCBA==[0,1,1,1,0]
        abcdei=[0,1,1,1,0,0];
        RD=1;
    elseif EDCBA==[0,1,1,1,1]
        abcdei=[1,0,1,0,0,0];
        RD=-1;
    elseif EDCBA==[1,0,0,0,0]
        abcdei=[1,0,0,1,0,0];
        RD=-1;
    elseif EDCBA==[1,0,0,0,1]
        abcdei=[1,0,0,0,1,1];
        RD=1;
    elseif EDCBA==[1,0,0,1,0]
        abcdei=[0,1,0,0,1,1];
        RD=1;
    elseif EDCBA==[1,0,0,1,1]
        abcdei=[1,1,0,0,1,0];
        RD=1;
    elseif EDCBA==[1,0,1,0,0]
        abcdei=[0,0,1,0,1,1];
        RD=1;
    elseif EDCBA==[1,0,1,0,1]
        abcdei=[1,0,1,0,1,0];
        RD=1;
    elseif EDCBA==[1,0,1,1,0]
        abcdei=[0,1,1,0,1,0];
        RD=1;
    elseif EDCBA==[1,0,1,1,1]
        abcdei=[0,0,0,1,0,1];
```

```
            RD=-1;
        elseif EDCBA==[1,1,0,0,0]
            abcdei=[0,0,1,1,0,0];
            RD=-1;
        elseif EDCBA==[1,1,0,0,1]
            abcdei=[1,0,0,1,1,0];
            RD=1;
        elseif EDCBA==[1,1,0,1,0]
            abcdei=[0,1,0,1,1,0];
            RD=1;
        elseif EDCBA==[1,1,0,1,1]
            abcdei=[0,0,1,0,0,1];
            RD=-1;
        elseif EDCBA==[1,1,1,0,0]
            abcdei=[0,0,1,1,1,0];
            RD=1;
        elseif EDCBA==[1,1,1,0,1]
            abcdei=[0,1,0,0,0,1];
            RD=-1;
        elseif EDCBA==[1,1,1,1,0]
            abcdei=[1,0,0,0,0,1];
            RD=-1;
        elseif EDCBA==[1,1,1,1,1]
            abcdei=[0,1,0,1,0,0];
            RD=-1;
        end
    elseif RD==-1
        if EDCBA==[0,0,0,0,0]
            abcdei=[1,0,0,1,1,1];
            RD=1;
        elseif EDCBA==[0,0,0,0,1]
            abcdei=[0,1,1,1,0,1];
            RD=1;
        elseif EDCBA==[0,0,0,1,0]
            abcdei=[1,0,1,1,0,1];
            RD=1;
        elseif EDCBA==[0,0,0,1,1]
            abcdei=[1,1,0,0,0,1];
            RD=-1;
        elseif EDCBA==[0,0,1,0,0]
            abcdei=[1,1,0,1,0,1];
            RD=1;
        elseif EDCBA==[0,0,1,0,1]
            abcdei=[1,0,1,0,0,1];
            RD=-1;
        elseif EDCBA==[0,0,1,1,0]
            abcdei=[0,1,1,0,0,1];
            RD=-1;
        elseif EDCBA==[0,0,1,1,1]
            abcdei=[1,1,1,0,0,0];
            RD=1;
        elseif EDCBA==[0,1,0,0,0]
            abcdei=[1,1,1,0,0,1];
            RD=1;
        elseif EDCBA==[0,1,0,0,1]
            abcdei=[1,0,0,1,0,1];
            RD=-1;
        elseif EDCBA==[0,1,0,1,0]
            abcdei=[0,1,0,1,0,1];
            RD=-1;
        elseif EDCBA==[0,1,0,1,1]
            abcdei=[1,1,0,1,0,0];
            RD=-1;
        elseif EDCBA==[0,1,1,0,0]
            abcdei=[0,0,1,1,0,1];
            RD=-1;
        elseif EDCBA==[0,1,1,0,1]
            abcdei=[1,0,1,1,0,0];
            RD=-1;
        elseif EDCBA==[0,1,1,1,0]
            abcdei=[0,1,1,1,0,0];
            RD=-1;
        elseif EDCBA==[0,1,1,1,1]
            abcdei=[0,1,0,1,1,1];
            RD=1;
        elseif EDCBA==[1,0,0,0,0]
            abcdei=[0,1,1,0,1,1];
            RD=1;
        elseif EDCBA==[1,0,0,0,1]
            abcdei=[1,0,0,0,1,1];
            RD=-1;
        elseif EDCBA==[1,0,0,1,0]
            abcdei=[0,1,0,0,1,1];
            RD=-1;
        elseif EDCBA==[1,0,0,1,1]
            abcdei=[1,1,0,0,1,0];
            RD=-1;
        elseif EDCBA==[1,0,1,0,0]
            abcdei=[0,0,1,0,1,1];
            RD=-1;
        elseif EDCBA==[1,0,1,0,1]
            abcdei=[1,0,1,0,1,0];
            RD=-1;
        elseif EDCBA==[1,0,1,1,0]
            abcdei=[0,1,1,0,1,0];
            RD=-1;
        elseif EDCBA==[1,0,1,1,1]
            abcdei=[1,1,1,0,1,0];
            RD=1;
        elseif EDCBA==[1,1,0,0,0]
            abcdei=[1,1,0,0,1,1];
            RD=1;
        elseif EDCBA==[1,1,0,0,1]
            abcdei=[1,0,0,1,1,0];
            RD=-1;
        elseif EDCBA==[1,1,0,1,0]
            abcdei=[0,1,0,1,1,0];
            RD=-1;
        elseif EDCBA==[1,1,0,1,1]
            abcdei=[1,1,0,1,1,0];
            RD=1;
        elseif EDCBA==[1,1,1,0,0]
            abcdei=[0,0,1,1,1,0];
            RD=-1;
        elseif EDCBA==[1,1,1,0,1]
            abcdei=[1,0,1,1,1,0];
            RD=1;
        elseif EDCBA==[1,1,1,1,0]
            abcdei=[0,1,1,1,1,0];
            RD=1;
        elseif EDCBA==[1,1,1,1,1]
            abcdei=[1,0,1,0,1,1];
            RD=1;
        end
    end

    %3b4b
    if RD==1
        if HGF==[0,0,0]
            fghj=[0,1,0,0];
            RD=-1;
        elseif HGF==[0,0,1]
            fghj=[1,0,0,1];
        elseif HGF==[0,1,0]
            fghj=[0,1,0,1];
        elseif HGF==[0,1,1]
            fghj=[1,1,0,0];
            RD=-1;
        elseif HGF==[1,0,0]
            fghj=[0,0,1,0];
            RD=-1;
        elseif HGF==[1,0,1]
            fghj=[1,0,1,0];
        elseif HGF==[1,1,0]
            fghj=[0,1,1,0];
        elseif HGF==[1,1,1]
            if abcdei(5:6)==[0,0]
                fghj=[1,0,0,0];
            else
                fghj=[0,0,0,1];
            end
        end
    elseif RD==-1
        if HGF==[0,0,0]
            fghj=[1,0,1,1];
            RD=1;
        elseif HGF==[0,0,1]
            fghj=[1,0,0,1];
        elseif HGF==[0,1,0]
            fghj=[0,1,0,1];
        elseif HGF==[0,1,1]
            fghj=[0,0,1,1];
            RD=1;
        elseif HGF==[1,0,0]
            fghj=[1,1,0,1];
            RD=1;
        elseif HGF==[1,0,1]
            fghj=[1,0,1,0];
        elseif HGF==[1,1,0]
            fghj=[0,1,1,0];
        elseif HGF==[1,1,1]
            if abcdei(5:6)==[1,1]
                fghj=[0,1,1,1];
            else
                fghj=[1,1,1,0];
            end
        end
    end

    b10=[abcdei,fghj];
```

The next function does the decoding of the previous function, but this one also have an extra output notifying if there has been an error detected:

```
function [b8,RD,ERR10b] = dec10b8b(b10,RD)

a=b10(1);
b=b10(2);
c=b10(3);
d=b10(4);
e=b10(5);
i=b10(6);
f=b10(7);
g=b10(8);
h=b10(9);
j=b10(10);

abcdei=[a,b,c,d,e,i];
fghj=[f,g,h,j];
ERR6b=1;
ERR4b=1;

%Arbitrary value for error case
HGF=[0,0,0];
EDCBA=[0,0,0,0,0];

%5b6b
if RD==1
    if abcdei==[0,1,1,0,0,0]
        EDCBA=[0,0,0,0,0];
        RD=-1;
        ERR6b=0;
    elseif abcdei==[1,0,0,0,1,0]
        EDCBA=[0,0,0,0,1];
        RD=-1;
        ERR6b=0;
    elseif abcdei==[0,1,0,0,1,0]
        EDCBA=[0,0,0,1,0];
        RD=-1;
        ERR6b=0;
    elseif abcdei==[1,1,0,0,0,1]
        EDCBA=[0,0,0,1,1];
        RD=1;
        ERR6b=0;
    elseif abcdei==[0,0,1,0,1,0]
        EDCBA=[0,0,1,0,0];
        RD=-1;
        ERR6b=0;
    elseif abcdei==[1,0,1,0,0,1]
        EDCBA=[0,0,1,0,1];
        RD=1;
        ERR6b=0;
    elseif abcdei==[0,1,1,0,0,1]
        EDCBA=[0,0,1,1,0];
        RD=1;
        ERR6b=0;
    elseif abcdei==[0,0,0,1,1,1]
        EDCBA=[0,0,1,1,1];
        RD=-1;
        ERR6b=0;
    elseif abcdei==[0,0,0,1,1,0]
        EDCBA=[0,1,0,0,0];
        RD=-1;
        ERR6b=0;
    elseif abcdei==[1,0,0,1,0,1]
        EDCBA=[0,1,0,0,1];
        RD=1;
        ERR6b=0;
    elseif abcdei==[0,1,0,1,0,1]
        EDCBA=[0,1,0,1,0];
        RD=1;
        ERR6b=0;
    elseif abcdei==[1,1,0,1,0,0]
        EDCBA=[0,1,0,1,1];
        RD=1;
        ERR6b=0;
    elseif abcdei==[0,0,1,1,0,1]
        EDCBA=[0,1,1,0,0];
        RD=1;
        ERR6b=0;
    elseif abcdei==[1,0,1,1,0,0]
        EDCBA=[0,1,1,0,1];
        RD=1;
        ERR6b=0;
    elseif abcdei==[0,1,1,1,0,0]
        EDCBA=[0,1,1,1,0];
        RD=1;
        ERR6b=0;
    elseif abcdei==[1,0,1,0,0,0]
        EDCBA=[0,1,1,1,1];
        RD=-1;
        ERR6b=0;
    elseif abcdei==[1,0,0,1,0,0]
        EDCBA=[1,0,0,0,0];
        RD=-1;
        ERR6b=0;
    elseif abcdei==[1,0,0,0,1,1]
        EDCBA=[1,0,0,0,1];
        RD=1;
        ERR6b=0;
    elseif abcdei==[0,1,0,0,1,1]
        EDCBA=[1,0,0,1,0];
        RD=1;
        ERR6b=0;
    elseif abcdei==[1,1,0,0,1,0]
        EDCBA=[1,0,0,1,1];
        RD=1;
        ERR6b=0;
    elseif abcdei==[0,0,1,0,1,1]
        EDCBA=[1,0,1,0,0];
        RD=1;
        ERR6b=0;
    elseif abcdei==[1,0,1,0,1,0]
        EDCBA=[1,0,1,0,1];
        RD=1;
        ERR6b=0;
    elseif abcdei==[0,1,1,0,1,0]
        EDCBA=[1,0,1,1,0];
        RD=1;
        ERR6b=0;
    elseif abcdei==[0,0,0,1,0,1]
        EDCBA=[1,0,1,1,1];
        RD=-1;
        ERR6b=0;
    elseif abcdei==[0,0,1,1,0,0]
```

```
            EDCBA=[1,1,0,0,0];
            RD=-1;
            ERR6b=0;
        elseif abcdei==[1,0,0,1,1,0]
            EDCBA=[1,1,0,0,1];
            RD=1;
            ERR6b=0;
        elseif abcdei==[0,1,0,1,1,0]
            EDCBA=[1,1,0,1,0];
            RD=1;
            ERR6b=0;
        elseif abcdei==[0,0,1,0,0,1]
            EDCBA=[1,1,0,1,1];
            RD=-1;
            ERR6b=0;
        elseif abcdei==[0,0,1,1,1,0]
            EDCBA=[1,1,1,0,0];
            RD=1;
            ERR6b=0;
        elseif abcdei==[0,1,0,0,0,1]
            EDCBA=[1,1,1,0,1];
            RD=-1;
            ERR6b=0;
        elseif abcdei==[1,0,0,0,0,1]
            EDCBA=[1,1,1,1,0];
            RD=-1;
            ERR6b=0;
        elseif abcdei==[0,1,0,1,0,0]
            EDCBA=[1,1,1,1,1];
            RD=-1;
            ERR6b=0;
        end
    elseif RD==-1
        if abcdei==[1,0,0,1,1,1]
            EDCBA=[0,0,0,0,0];
            RD=1;
            ERR6b=0;
        elseif abcdei==[0,1,1,1,0,1]
            EDCBA=[0,0,0,0,1];
            RD=1;
            ERR6b=0;
        elseif abcdei==[1,0,1,1,0,1]
            EDCBA=[0,0,0,1,0];
            RD=1;
            ERR6b=0;
        elseif abcdei==[1,1,0,0,0,1]
            EDCBA=[0,0,0,1,1];
            RD=-1;
            ERR6b=0;
        elseif abcdei==[1,1,0,1,0,1]
            EDCBA=[0,0,1,0,0];
            RD=1;
            ERR6b=0;
        elseif abcdei==[1,0,1,0,0,1]
            EDCBA=[0,0,1,0,1];
            RD=-1;
            ERR6b=0;
        elseif abcdei==[0,1,1,0,0,1]
            EDCBA=[0,0,1,1,0];
            RD=-1;
            ERR6b=0;
        elseif abcdei==[1,1,1,0,0,0]
            EDCBA=[0,0,1,1,1];
            RD=1;
            ERR6b=0;
        elseif abcdei==[1,1,1,0,0,1]
            EDCBA=[0,1,0,0,0];
            RD=1;
            ERR6b=0;
        elseif abcdei==[1,0,0,1,0,1]
            EDCBA=[0,1,0,0,1];
            RD=-1;
            ERR6b=0;
        elseif abcdei==[0,1,0,1,0,1]
            EDCBA=[0,1,0,1,0];
            RD=-1;
            ERR6b=0;
        elseif abcdei==[1,1,0,1,0,0]
            EDCBA=[0,1,0,1,1];
            RD=-1;
            ERR6b=0;
        elseif abcdei==[0,0,1,1,0,1]
            EDCBA=[0,1,1,0,0];
            RD=-1;
            ERR6b=0;
        elseif abcdei==[1,0,1,1,0,0]
            EDCBA=[0,1,1,0,1];
            RD=-1;
            ERR6b=0;
        elseif abcdei==[0,1,1,1,0,0]
            EDCBA=[0,1,1,1,0];
            RD=-1;
            ERR6b=0;
        elseif abcdei==[0,1,0,1,1,1]
            EDCBA=[0,1,1,1,1];
            RD=1;
            ERR6b=0;
        elseif abcdei==[0,1,1,0,1,1]
            EDCBA=[1,0,0,0,0];
            RD=1;
            ERR6b=0;
        elseif abcdei==[1,0,0,0,1,1]
            EDCBA=[1,0,0,0,1];
            RD=-1;
            ERR6b=0;
        elseif abcdei==[0,1,0,0,1,1]
            EDCBA=[1,0,0,1,0];
            RD=-1;
            ERR6b=0;
        elseif abcdei==[1,1,0,0,1,0]
            EDCBA=[1,0,0,1,1];
            RD=-1;
            ERR6b=0;
        elseif abcdei==[0,0,1,0,1,1]
            EDCBA=[1,0,1,0,0];
            RD=-1;
            ERR6b=0;
        elseif abcdei==[1,0,1,0,1,0]
            EDCBA=[1,0,1,0,1];
            RD=-1;
            ERR6b=0;
        elseif abcdei==[0,1,1,0,1,0]
            EDCBA=[1,0,1,1,0];
            RD=-1;
            ERR6b=0;
        elseif abcdei==[1,1,1,0,1,0]
            EDCBA=[1,0,1,1,1];
            RD=1;
            ERR6b=0;
        elseif abcdei==[1,1,0,0,1,1]
            EDCBA=[1,1,0,0,0];
            RD=1;
            ERR6b=0;
        elseif abcdei==[1,0,0,1,1,0]
            EDCBA=[1,1,0,0,1];
            RD=-1;
            ERR6b=0;
        elseif abcdei==[0,1,0,1,1,0]
            EDCBA=[1,1,0,1,0];
            RD=-1;
            ERR6b=0;
        elseif abcdei==[1,1,0,1,1,0]
            EDCBA=[1,1,0,1,1];
            RD=1;
            ERR6b=0;
        elseif abcdei==[0,0,1,1,1,0]
            EDCBA=[1,1,1,0,0];
            RD=-1;
            ERR6b=0;
        elseif abcdei==[1,0,1,1,1,0]
            EDCBA=[1,1,1,0,1];
            RD=1;
            ERR6b=0;
        elseif abcdei==[0,1,1,1,1,0]
```

```
        EDCBA=[1,1,1,1,0];
        RD=1;
        ERR6b=0;
    elseif abcdei==[1,0,1,0,1,1]
        EDCBA=[1,1,1,1,1];
        RD=1;
        ERR6b=0;
    end
end

%3b4b
if RD==1
    if fghj==[0,1,0,0]
        HGF=[0,0,0];
        RD=-1;
        ERR4b=0;
    elseif fghj==[1,0,0,1]
        HGF=[0,0,1];
        ERR4b=0;
    elseif fghj==[0,1,0,1]
        HGF=[0,1,0];
        ERR4b=0;
    elseif fghj==[1,1,0,0]
        HGF=[0,1,1];
        RD=-1;
        ERR4b=0;
    elseif fghj==[0,0,1,0]
        HGF=[1,0,0];
        RD=-1;
        ERR4b=0;
    elseif fghj==[1,0,1,0]
        HGF=[1,0,1];
        ERR4b=0;
    elseif fghj==[0,1,1,0]
        HGF=[1,1,0];
        ERR4b=0;
    elseif fghj==[1,0,0,0]
        HGF=[1,1,1];
        ERR4b=0;
    elseif fghj==[0,0,0,1]
        HGF=[1,1,1];
        ERR4b=0;
    end
elseif RD==-1
    if fghj==[1,0,1,1]
        HGF=[0,0,0];
        RD=1;
        ERR4b=0;
    elseif fghj==[1,0,0,1]
        HGF=[0,0,1];
    elseif fghj==[0,1,0,1]
        HGF=[0,1,0];
        ERR4b=0;
    elseif fghj==[0,0,1,1]
        HGF=[0,1,1];
        RD=1;
        ERR4b=0;
    elseif fghj==[1,1,0,1]
        HGF=[1,0,0];
        RD=1;
        ERR4b=0;
    elseif fghj==[1,0,1,0]
        HGF=[1,0,1];
        ERR4b=0;
    elseif fghj==[0,1,1,0]
        HGF=[1,1,0];
        ERR4b=0;
    elseif fghj==[0,1,1,1]
        HGF=[1,1,1];
        ERR4b=0;
    elseif fghj==[1,1,1,0]
        HGF=[1,1,1];
        ERR4b=0;
    end
end

if (ERR6b==1)||(ERR4b==1)
```

```
    ERR10b=1;
else
    ERR10b=0;
end

b8=bit2byte([HGF,EDCBA]);
```

These couple of functions below implement the mathematic model:

```
function [N_err, N_err_FEC] =
mathematic_model(msg,msg_FEC,m_FEC,n_FEC,k_FEC,P_tx,P_N,N_rx,att)
%SYSTEM PARAMETERS
%----------------------------------------------------------
N=length(msg);
msg_b=zeros(1,N*8);
%CONVERSION TO BITS OF msg and msg_FEC
for i=1:N
    msg_b((8*i-7):(8*i))=byte2bit(msg(i));
end
N_FEC=length(msg_FEC);
msg_FEC_b=zeros(1,8*N_FEC);
for i=1:N_FEC
    msg_FEC_b((8*i-7):(8*i))=byte2bit(msg_FEC(i));
end
N_b=length(msg_b); %Lenght of the transmited signal.
N_FEC_b=length(msg_FEC_b); %Lenght of the transmited signal.
%----------------------------------------------------------

%Simulation of the system without FEC
%----------------------------------------------------------
msg_tx=msg_b*P_tx; %Transmited signal.
Nrx=P_N*N_rx(1:N_b); %Noise at the receiver.
P_Srx=msg_tx/(10^(att/10)); %Received signal without noise.
Xrx=P_Srx+Nrx; %Received signal with noise.
threshold=0.5*P_tx/(10^(att/10)); %Threshold.
%Detection
S_Rx=zeros(1,N_b);
for i=1:N_b
    if Xrx(i)>threshold
        S_Rx(i)=1;
    end
end
%Restauration
S_det=zeros(1,N);
for i=1:N
    S_det(i)=bit2byte(S_Rx((8*i-7):(8*i)));
end
%BER_results
N_err=0;
for i=1:N
    if S_det(i)~=msg(i)
        N_err=N_err+1;
    end
end
%----------------------------------------------------------

%Simulation of the system with FEC
%----------------------------------------------------------
msg_FEC_tx=msg_FEC_b*P_tx; %Transmited signal.
Nrx=P_N*N_rx; %Noise at the receiver.
P_Srx=msg_FEC_tx/(10^(att/10)); %Received signal without noise.
Xrx=P_Srx+Nrx; %Received signal with noise.
threshold=0.5*P_tx/(10^(att/10)); %Threshold.
%Detection
S_Rx=zeros(1,N_FEC_b);
for i=1:N_FEC_b
    if Xrx(i)>threshold
        S_Rx(i)=1;
    end
end
%Restauration
S_det_FEC=zeros(1,N_FEC);
for i=1:N_FEC
    S_det_FEC(i)=bit2byte(S_Rx((8*i-7):(8*i)));
end
```

```
S_det_FEC=gf(S_det_FEC,m_FEC);
S_det=rsdec(S_det_FEC,n_FEC,k_FEC);
S_det=double(S_det.x);
%BER_results
N_err_FEC=0;
for i=1:N
    if S_det(i)~=msg(i)
        N_err_FEC=N_err_FEC+1;
    end
end
%----------------------------------------------------------
```

```
clear all;
close all;

m=8;
n=2^m-1;
k=239;
P_tx=100;
mean_N=0;
att=0;
P_N_min=1;
P_N_max=100;
err=zeros(1,P_N_max-P_N_min+1);
err_FEC=zeros(1,P_N_max-P_N_min+1);
SNR=zeros(1,P_N_max-P_N_min+1);
for i=P_N_min:P_N_max
        SNR(i-P_N_min+1)=10*log10(P_tx/i);
end
ERR=zeros(1,P_N_max-P_N_min+1);
ERR_FEC=zeros(1,P_N_max-P_N_min+1);
x=50;  %Number of sent packages

for j=1:x
    msg=randi([0 n],1,k);
    msg=gf(msg,m);
    msg_FEC=rsenc(msg,n,k);
    msg=double(msg.x);
    msg_FEC=double(msg_FEC.x);
    N_rx=randn(1,n*8)+mean_N;

    for i=P_N_min:P_N_max
        [err(i-P_N_min+1),err_FEC(i-
P_N_min+1)]=mathematic_model(msg,msg_FEC,m,n,k,P_tx,i,N_rx,att);
    end
    ERR=ERR+err;
    ERR_FEC=ERR_FEC+err_FEC;
end
BER=ERR/(k*x);
BER_FEC=ERR_FEC/(k*x);

semilogy(SNR,BER,'b');
hold on
semilogy(SNR,BER_FEC,'r');
hold off
xlabel('SNR (dB)');
ylabel('BER');
title('Mathematic Model');
legend('BER without FEC','BER with FEC');
```

The following function implements the transmitter, so it reads a picture from your computer (this picture must be on the same path as this function!), is divided into 255 and 239 bytes words' and it is given as three outputs (the one divided into 239 the one into 255 and the original image without being modified). The default format is *.png but it accepts any kind of picture if it is changed:

```
function [D,DATA_239,DATA_255] = transmitter()

clear all;
FILENAME = uigetfile('*.jpeg');
D=importdata(FILENAME);
```

```
imshow(D);
[Dx,Dy,Dz]=size(D);
V=zeros(1,Dx*Dy*Dz);
aux=1;
%DATA to vector
for i=1:Dx
    for j=1:Dy
        for k=1:Dz
            V(aux)=D(i,j,k);
            aux=aux+1;
        end
    end
end
%Fragmenting data parameters
I=zeros(1,6);
I(1)=floor(Dx/255);
I(2)=floor(Dy/255);
I(3)=floor(Dz/255);
I(4)=Dx-255*I(1);
I(5)=Dy-255*I(2);
I(6)=Dz-255*I(3);

V2send=[I,V];
aux=length(V2send);

%CASE 239
aux2=ceil(aux/239);
size2=aux2*239;
V239=zeros(1,size2);
for h=1:aux
    V239(h)=V2send(h);
end
DATA_239=reshape(V239,239,aux2);

%CASE 255
aux2=ceil(aux/255);
size2=aux2*255;
V255=zeros(1,size2);
for h=1:aux
    V255(h)=V2send(h);
end
DATA_255=reshape(V255,255,aux2);

%Clearing auxiliar variables
clear Dx; clear Dy; clear Dz; clear I; clear FILENAME; clear V; clear V239; clear V255;
clear V2send; clear aux; clear aux2; clear h; clear i; clear j; clear k;
clear l; clear o; clear p; clear size2;

%Saving data ready to send
save ready2send
```

The next function acts as the receiver, it reconstructs the pictures, counts the wrong bytes and pixels (the bits have to be counted before this) and plots the three pictures with each corresponding information:

```
function [R239,R255]=receiver(D239,D255,D0,NberrFEC,Nberr)

%Preparing figures:
close all;
figure('name','results');
hold on;
subplot(1,3,1);
imshow(D0);
title('Original Image');

%Vectorizing
%239
[a,b]=size(D239);
Dv239=reshape(D239,1,a*b);
%255
[a,b]=size(D255);
Dv255=reshape(D255,1,a*b);

%Extract parameters
```

```
%239
Dx239=255*Dv239(1)+Dv239(4);
Dy239=255*Dv239(2)+Dv239(5);
Dz239=255*Dv239(3)+Dv239(6);
%255
Dx255=255*Dv255(1)+Dv255(4);
Dy255=255*Dv255(2)+Dv255(5);
Dz255=255*Dv255(3)+Dv255(6);

%239 Reconstruction
R239=zeros(Dx239,Dy239,Dz239);
aux=7;
for i=1:Dx239
    for j=1:Dy239
        for k=1:Dz239
            R239(i,j,k)=Dv239(aux);
            aux=aux+1;
        end
    end
end
R239=uint8(R239);
subplot(1,3,2);
imshow(R239);
title('FEC Image');
text(0,Dx239+100,strcat('Bit errors: ',num2str(NberrFEC),'   BER:
',num2str(NberrFEC/(Dx239*Dy239*Dz239*8))));
ByteErr=0;
PixelErr=0;
for i=1:Dx239
    for j=1:Dy239
        for k=1:Dz239
            if D0(i,j,k)~=R239(i,j,k)
                ByteErr=ByteErr+1;
            end
        end
        if (D0(i,j,1)~=R239(i,j,1))||(D0(i,j,2)~=R239(i,j,2))||(D0(i,j,3)~=R239(i,j,3))
            PixelErr=PixelErr+1;
        end
    end
end
text(0,Dx239+200,strcat('Byte errors: ',num2str(ByteErr)));
text(0,Dx239+300,strcat('Pixel errors: ',num2str(PixelErr)));

%255 Reconstruction
R255=zeros(Dx255,Dy255,Dz255);
aux=7;
for i=1:Dx255
    for j=1:Dy255
        for k=1:Dz255
            R255(i,j,k)=Dv255(aux);
            aux=aux+1;
        end
    end
end
R255=uint8(R255);
subplot(1,3,3);
imshow(R255);
title('No-FEC Image');
text(0,Dx255+100,strcat('Bit errors: ',num2str(Nberr),'   BER:
',num2str(Nberr/(Dx255*Dy255*Dz255*8))));
ByteErr=0;
PixelErr=0;
for i=1:Dx255
    for j=1:Dy255
        for k=1:Dz255
            if D0(i,j,k)~=R255(i,j,k)
                ByteErr=ByteErr+1;
            end
        end
        if (D0(i,j,1)~=R255(i,j,1))||(D0(i,j,2)~=R255(i,j,2))||(D0(i,j,3)~=R255(i,j,3))
            PixelErr=PixelErr+1;
        end
    end
end
text(0,Dx255+200,strcat('Byte errors: ',num2str(ByteErr)));
text(0,Dx255+300,strcat('Pixel errors: ',num2str(PixelErr)));
```

```
Hold off;
```

With all this functions the model functions can also be implemented. First the model which contains only the FEC codification:

```
function [R239,R255,D] = systemFEC(SNR_dB)

%TX
[D,D239,D255] = transmitter();
DFEC=FECcod(D239,8);
hFEC=zeros(1,6);
h255=zeros(1,6);
for i=1:6
    hFEC(i)=DFEC(i,1);
    h255(i)=D255(i,1);
end

%CHANNEL
L=length(DFEC);
P_N=1;
P_Tx=P_N*10^(SNR_dB/10);
Noise=randn(L,255*8)*P_N;

%FEC case
DFEC_Rx=zeros(255,L);
for i=1:L
    v2send=DFEC(:,i);
    bits2send=zeros(1,255*8);
    for j=1:255
        bits2send((8*j-7):(8*j))=byte2bit(v2send(j));
    end
    v_Rx=bits2send*P_Tx+Noise(i,:);
    %DETECTOR------------------------------------------------
    threshold=0.5*P_Tx; %Threshold.
    S_Rx=zeros(1,255*8);
    for aux=1:(255*8)
        if v_Rx(aux)>threshold
            S_Rx(aux)=1;
        end
    end
    %--------------------------------------------------------
    for k=1:255
        DFEC_Rx(k,i)=bit2byte(S_Rx((8*k-7):(8*k)));
    end
end

%NO-FEC case
L=length(D255);
D255_Rx=zeros(255,L);
for i=1:L
    v2send=D255(:,i);
    bits2send=zeros(1,255*8);
    for j=1:255
        bits2send((8*j-7):(8*j))=byte2bit(v2send(j));
    end
    v_Rx=bits2send*P_Tx+Noise(i,:);
    %DETECTOR------------------------------------------------
    threshold=0.5*P_Tx; %Threshold.
    S_Rx=zeros(1,255*8);
    for aux=1:(255*8)
        if v_Rx(aux)>threshold
            S_Rx(aux)=1;
        end
    end
    %--------------------------------------------------------
    for k=1:255
        D255_Rx(k,i)=bit2byte(S_Rx((8*k-7):(8*k)));
    end
end

%RX
DFEC_ready=FECdec(DFEC_Rx,8);
%Bit error counter FEC-----------------------------------
L=length(DFEC_ready);
```

```
NberrFEC=0;
Nberr=0;
for i=1:L
    for j=1:239
        bitDFEC_Rx=byte2bit(DFEC_ready(j,i));
        bitAux=byte2bit(D239(j,i));
        for k=1:8
            if bitDFEC_Rx(k)~=bitAux(k)
                NberrFEC=NberrFEC+1;
            end
        end
    end
end
L=length(D255_Rx);
for i=1:L
    for j=1:255
        bitD255_Rx=byte2bit(D255_Rx(j,i));
        bitAux=byte2bit(D255(j,i));
        for k=1:8
            if bitD255_Rx(k)~=bitAux(k)
                Nberr=Nberr+1;
            end
        end
    end
end
%------------------------------------------------------------
for i=1:6
    DFEC_ready(i,1)=hFEC(i);
    D255_Rx(i,1)=h255(i);
end
[R239,R255] = receiver(DFEC_ready,D255_Rx,D,NberrFEC,Nberr);
```

The next function implements the model that computes the effect of the errors in 8b/10b:

```
function system8b10b()

close all;
figure('name','Errors in 8b/10b');

%Case 1: RD=1 i error in 10b
errors=zeros(1,2560);
detErr=zeros(1,2560);

for i=0:255
    v8b=byte2bit(i);
    [v10b,RD]=enc8b10b(i,1);
    for j=1:10
        if v10b(j)==1
            v10b(j)=0;
        else
            v10b(j)=1;
        end
        [res,RD,detErr(i*10+j)]=dec10b8b(v10b,1);
        result=byte2bit(res);
        for k=1:8
            if result(k)~=v8b(k)
                errors(i*10+j)=errors(i*10+j)+1;
            end
        end
    end
end

mErr=mean(errors);
meanErr=zeros(1,2560);
for i=1:2560
    meanErr(i)=mErr;
end

subplot(1,3,1);
hold on;
stem(errors,'b');
plot(meanErr,'r','linewidth',5);
title('Errors in 8b for 1 error in 10b (RD=1)');
ylabel('Number of errors (bits) per Byte');
legend('Errors','Error mean');
```

```
text(1800,7.2,strcat('Errors detected:
',num2str(100*sum(detErr)/2560),'%'),'BackgroundColor',[.7 .9 .7]);
hold off;

%Case 2: RD=-1 i error in 10b
errors=zeros(1,2560);
detErr=zeros(1,2560);

for i=0:255
    v8b=byte2bit(i);
    [v10b,RD]=enc8b10b(i,-1);
    for j=1:10
        if v10b(j)==1
            v10b(j)=0;
        else
            v10b(j)=1;
        end
        [res,RD,detErr(i*10+j)]=dec10b8b(v10b,-1);
        result=byte2bit(res);
        for k=1:8
            if result(k)~=v8b(k)
                errors(i*10+j)=errors(i*10+j)+1;
            end
        end
    end
end

mErr=mean(errors);
meanErr=zeros(1,2560);
for i=1:2560
    meanErr(i)=mErr;
end

subplot(1,3,2);
hold on;
stem(errors,'b');
plot(meanErr,'r','linewidth',5);
title('Errors in 8b for 1 error in 10b (RD=-1)');
ylabel('Number of errors (bits) per Byte');
legend('Errors','Error mean');
text(1800,7.2,strcat('Errors detected:
',num2str(100*sum(detErr)/2560),'%'),'BackgroundColor',[.7 .9 .7]);
hold off;

%Case 3: 1 error in RD
errors=zeros(1,512);
detErr=zeros(1,512);

for i=0:255
    v8b=byte2bit(i);
    for j=1:2
        [v10b,RD]=enc8b10b(i,(-1)^j);
        [res,RD,detErr(i*2+j)]=dec10b8b(v10b,(-1)^(j-1));
        result=byte2bit(res);
        for k=1:8
            if result(k)~=v8b(k)
                errors(i*2+j)=errors(i*2+j)+1;
            end
        end
    end
end

mErr=mean(errors);
meanErr=zeros(1,512);
for i=1:512
    meanErr(i)=mErr;
end

subplot(1,3,3);
hold on;
stem(errors,'b');
plot(meanErr,'r','linewidth',5);
title('Errors in 8b for 1 error in RD');
ylabel('Number of errors (bits) per Byte');
legend('Errors','Error mean');
text(350,4.5,strcat('Errors detected:
',num2str(100*sum(detErr)/512),'%'),'BackgroundColor',[.7 .9 .7]);
```

```
hold off;
```

The following function is the model that combines first the FEC and afterwards the 8b/10b encoding:

```
function [R239,R255,D] = system_FEC_8b10b(SNR_dB)

%TX
[D,D239,D255] = transmitter();
DFEC=FECcod(D239,8);
hFEC=zeros(1,6);
h255=zeros(1,6);
for i=1:6
    hFEC(i)=DFEC(i,1);
    h255(i)=D255(i,1);
end

%CHANNEL
L=length(DFEC);
P_N=1;
RD=1;
P_Tx=P_N*10^(SNR_dB/10);
Noise=randn(L,255*10)*P_N;

%FEC case
DFEC_Rx=zeros(255,L);
for i=1:L
    v2send=DFEC(:,i);
    bits2send=zeros(1,255*10);
    for j=1:255
        %8b/10b encoding---------------------------------------
        [bits2send((10*j-9):(10*j)),RD]=enc8b10b(v2send(j),RD);
        %-----------------------------------------------------
    end
    v_Rx=bits2send*P_Tx+Noise(i,:);
    %DETECTOR-------------------------------------------------
    threshold=0.5*P_Tx; %Threshold.
    S_Rx=zeros(1,255*10);
    for aux=1:(255*10)
        if v_Rx(aux)>threshold
            S_Rx(aux)=1;
        end
    end
    %-------------------------------------------------------
    RD=1;
    for k=1:255
        %8b/10b decodding--------------------------------------
        [DFEC_Rx(k,i),RD,errs]=dec10b8b(S_Rx((10*k-9):(10*k)),RD);
        %-----------------------------------------------------
    end
end

%NO-FEC case
L=length(D255);
RD=1;
D255_Rx=zeros(255,L);
for i=1:L
    v2send=D255(:,i);
    bits2send=zeros(1,255*10);
    for j=1:255
        %8b/10b encoding---------------------------------------
        [bits2send((10*j-9):(10*j)),RD]=enc8b10b(v2send(j),RD);
        %-----------------------------------------------------
    end
    v_Rx=bits2send*P_Tx+Noise(i,:);
    %DETECTOR-------------------------------------------------
    threshold=0.5*P_Tx; %Threshold.
    S_Rx=zeros(1,255*10);
    for aux=1:(255*10)
        if v_Rx(aux)>threshold
            S_Rx(aux)=1;
        end
    end
    %-------------------------------------------------------
    for k=1:255
        %8b/10b decodding--------------------------------------
        [D255_Rx(k,i),RD,errs]=dec10b8b(S_Rx((10*k-9):(10*k)),RD);
        %------------------------------------------------------
    end
end

%RX
DFEC_ready=FECdec(DFEC_Rx,8);
%Bit error counter FEC---------------------------------------
L=length(DFEC_ready);
NberrFEC=0;
Nberr=0;
for i=1:L
    for j=1:239
        bitDFEC_Rx=byte2bit(DFEC_ready(j,i));
        bitAux=byte2bit(D239(j,i));
        for k=1:8
            if bitDFEC_Rx(k)~=bitAux(k)
                NberrFEC=NberrFEC+1;
            end
        end
    end
end
L=length(D255_Rx);
for i=1:L
    for j=1:255
        bitD255_Rx=byte2bit(D255_Rx(j,i));
        bitAux=byte2bit(D255(j,i));
        for k=1:8
            if bitD255_Rx(k)~=bitAux(k)
                Nberr=Nberr+1;
            end
        end
    end
end
%------------------------------------------------------------
for i=1:6
    DFEC_ready(i,1)=hFEC(i);
    D255_Rx(i,1)=h255(i);
end
[R239,R255] = receiver(DFEC_ready,D255_Rx,D,NberrFEC,Nberr);
```

This last function implements the model that combines the 8b/10b first of all and then the FEC:

```
function [R239,R255,D] = system_8b10b_FEC(SNR_dB)

%TX
[D,D239,D255] = transmitter();
hFEC=zeros(1,6);
h255=zeros(1,6);
for i=1:6
    hFEC(i)=D239(i,1);
    h255(i)=D255(i,1);
end
%8b/10b encoding--------------------------------------------------
%----------------------------------------------------------------
[aux1,aux2]=size(D239);
D239b10=zeros(aux1,aux2);
RD=1;
for i=1:aux1
    for j=1:aux2
        [aux3,RD]=enc8b10b(D239(i,j),RD);
        B=0;
        for k=1:10
            B=B+aux3(11-k)*(2^(k-1));
        end
        D239b10(i,j)=B;
    end
end
[aux1,aux2]=size(D255);
D255b10=zeros(aux1,aux2);
RD=1;
```

```
for i=1:aux1
    for j=1:aux2
        [aux3,RD]=enc8b10b(D255(i,j),RD);
        B=0;
        for k=1:10
            B=B+aux3(11-k)*(2^(k-1));
        end
        D255b10(i,j)=B;
    end
end
%-----------------------------------------------------------------
%-----------------------------------------------------------------
DFEC=FECcod(D239b10,10);

%CHANNEL
L=length(DFEC);
P_N=1;
P_Tx=P_N*10^(SNR_dB/10);
Noise=randn(L,255*10)*P_N;

%FEC case
D239_Rx=zeros(255,L);
for i=1:L
    v2send=DFEC(:,i);
    bits2send=zeros(1,255*10);
    for j=1:255
        %baud2bit---------------------------------------------
        B=v2send(j);
        b=zeros(1,10);
        b(10)=mod(B,2);
        aux=floor(B/2);
        b(9)=mod(aux,2);
        aux=floor(aux/2);
        b(8)=mod(aux,2);
        aux=floor(aux/2);
        b(7)=mod(aux,2);
        aux=floor(aux/2);
        b(6)=mod(aux,2);
        aux=floor(aux/2);
        b(5)=mod(aux,2);
        aux=floor(aux/2);
        b(4)=mod(aux,2);
        aux=floor(aux/2);
        b(3)=mod(aux,2);
        aux=floor(aux/2);
        b(2)=mod(aux,2);
        b(1)=floor(aux/2);
        bits2send((10*j-9):(10*j))=b;
        %---------------------------------------------------
    end
    v_Rx=bits2send*P_Tx+Noise(i,:);
    %DETECTOR-----------------------------------------------------
    threshold=0.5*P_Tx; %Threshold.
    S_Rx=zeros(1,255*10);
    for aux=1:(255*10)
        if v_Rx(aux)>threshold
            S_Rx(aux)=1;
        end
    end
    %---------------------------------------------------------
    for k=1:255
        %bit2baud---------------------------------------------
        B=0;
        b=S_Rx((10*k-9):(10*k));
        for l=1:10
            B=B+b(11-l)*(2^(l-1));
        end
        D239_Rx(k,i)=B;
        %---------------------------------------------------
    end
end

%NO-FEC case
L=length(D255b10);
D255_Rx=zeros(255,L);
for i=1:L
    v2send=D255b10(:,i);
    bits2send=zeros(1,255*10);
    for j=1:255
        %baud2bit-------------------------------------------
        B=v2send(j);
        b=zeros(1,10);
        b(10)=mod(B,2);
        aux=floor(B/2);
        b(9)=mod(aux,2);
        aux=floor(aux/2);
        b(8)=mod(aux,2);
        aux=floor(aux/2);
        b(7)=mod(aux,2);
        aux=floor(aux/2);
        b(6)=mod(aux,2);
        aux=floor(aux/2);
        b(5)=mod(aux,2);
        aux=floor(aux/2);
        b(4)=mod(aux,2);
        aux=floor(aux/2);
        b(3)=mod(aux,2);
        aux=floor(aux/2);
        b(2)=mod(aux,2);
        b(1)=floor(aux/2);
        bits2send((10*j-9):(10*j))=b;
        %---------------------------------------------------
    end
    v_Rx=bits2send*P_Tx+Noise(i,:);
    %DETECTOR-------------------------------------------------
    threshold=0.5*P_Tx; %Threshold.
    S_Rx=zeros(1,255*10);
    for aux=1:(255*10)
        if v_Rx(aux)>threshold
            S_Rx(aux)=1;
        end
    end
    %--------------------------------------------------------
    for k=1:255
        %bit2baud--------------------------------------------
        B=0;
        b=S_Rx((10*k-9):(10*k));
        for l=1:10
            B=B+b(11-l)*(2^(l-1));
        end
        D255_Rx(k,i)=B;
        %---------------------------------------------------
    end
end

%RX
D239_Rx=FECdec(D239_Rx,10);
%10b/8b decoding----------------------------------------------
%-------------------------------------------------------------
[aux1,aux2]=size(D239_Rx);
RD=1;
for i=1:aux1
    for j=1:aux2
        %baud2bit-------------------------------------------
        B=D239_Rx(i,j);
        b=zeros(1,10);
        b(10)=mod(B,2);
        aux=floor(B/2);
        b(9)=mod(aux,2);
        aux=floor(aux/2);
        b(8)=mod(aux,2);
        aux=floor(aux/2);
        b(7)=mod(aux,2);
        aux=floor(aux/2);
        b(6)=mod(aux,2);
        aux=floor(aux/2);
        b(5)=mod(aux,2);
        aux=floor(aux/2);
        b(4)=mod(aux,2);
        aux=floor(aux/2);
        b(3)=mod(aux,2);
        aux=floor(aux/2);
        b(2)=mod(aux,2);
        b(1)=floor(aux/2);
        %--------------------------------------------------
        [aux3,RD,errs]=dec10b8b(b,RD);
        D239_Rx(i,j)=aux3;
    end
end
[aux1,aux2]=size(D255_Rx);
RD=1;
for i=1:aux1
    for j=1:aux2
        %baud2bit------------------------------------------
        B=D255_Rx(i,j);
        b=zeros(1,10);
        b(10)=mod(B,2);
        aux=floor(B/2);
        b(9)=mod(aux,2);
        aux=floor(aux/2);
        b(8)=mod(aux,2);
        aux=floor(aux/2);
        b(7)=mod(aux,2);
        aux=floor(aux/2);
        b(6)=mod(aux,2);
        aux=floor(aux/2);
        b(5)=mod(aux,2);
        aux=floor(aux/2);
        b(4)=mod(aux,2);
        aux=floor(aux/2);
        b(3)=mod(aux,2);
        aux=floor(aux/2);
        b(2)=mod(aux,2);
        b(1)=floor(aux/2);
        %--------------------------------------------------
        [aux3,RD,errs]=dec10b8b(b,RD);
        D255_Rx(i,j)=aux3;
    end
end
%----------------------------------------------------------------
%----------------------------------------------------------------
%Bit error counter FEC----------------------------------------
L=length(D239_Rx);
NberrFEC=0;
Nberr=0;
for i=1:L
    for j=1:239
        bitDFEC_Rx=byte2bit(D239_Rx(j,i));
        bitAux=byte2bit(D239(j,i));
        for k=1:8
            if bitDFEC_Rx(k)~=bitAux(k)
                NberrFEC=NberrFEC+1;
            end
        end
    end
end
L=length(D255_Rx);
for i=1:L
    for j=1:255
        bitD255_Rx=byte2bit(D255_Rx(j,i));
        bitAux=byte2bit(D255(j,i));
        for k=1:8
            if bitD255_Rx(k)~=bitAux(k)
                Nberr=Nberr+1;
            end
        end
    end
end
%------------------------------------------------------------
for i=1:6
    D239_Rx(i,1)=hFEC(i);
    D255_Rx(i,1)=h255(i);
end
[R239,R255] = receiver(D239_Rx,D255_Rx,D,NberrFEC,Nberr);
```

Glossary

FEC: Forward Error Correction.

PCS: Physical Codding Sublayer.

RS: Reed-Solomon.

GF: Galois Field.

BER: Bit Error Rate.

RTP: Real-time Transport Protocol.

WDM: Wavelength Division Multiplexing.

ud-WDM: ultra dense – Wavelength Division Multiplexing.

PON: Passive Optical Network.

OLT: Optical Line Terminal.

ONU: Optical Network Unit.

LAN: Local Area Network.

WBS: Work Breakdown Structure.

SFP: Small Form-factor pluggable.