R-DVB:

Software Defined Radio implementation of DVB-T signal detection functions for digital terrestrial television.

Contents

Introduction

Current Solution: Full Hardware Implementation.

As a well known and largely spread technology ETSI (European Telecommunication Standard Institute) DVB (Digital Video Broadcasting) is a project developed by an European-based industry consortium, with more than 270[1] members, which has been developing specifications for digital television broadcasting since 1992, many of them now used all around the world from south America to Australia.

Actual standard implementation of this technology rely completely upon hardware components: hardware is the modulator, hardware is the receiver, while software is relegated in small platforms that may be used as substitute performing some plain TV-related functions such as channel switching or video recording. Nowadays digital audio-video streams are sent on air through modulators built into satellites or terrestrial base-stations and received with dedicated hardware set top boxes (STBs) connected to common analogue television, or with built in digital-TV.

To be a winning technology, this standard had to allow a variety of transmission both over air and over cable, with a complete whole of parameters that may be set to fit peculiar countries' television needs and habits, showing a great deal of flexibility. As an example “2k mode” (2048 carriers) is suitable for single transmitter operation in small single frequency network (SFN) with limited transmitter distances, while the “8k mode” is commonly used for the normal digital television broadcasting, be it terrestrial (DVB-T) or by satellite (DVB-S), while the “4k mode” , exclusively for use in the DVB-H, offers an additional flexibility hybrid feature.

As said, at the moment of writing almost each and every operation needed to convert an electromagnetic wave into a video stream (and vice-versa) are executed trough dedicated hardware, projected and set to implement the peculiar DVB functions. While this is really handy and reasonable when we are dealing just with feasibility problems it soon becomes an heavy, limiting burden when it comes to face with market inertia, personalization of characteristics, IP-TV competition and possibility of updates. It is quite paradigmatic of this the lack of dynamism shown by technological oriented markets based on people unwillingness to change their hardware (and habits) without a really strong reason to. Consumers, in fact, seem to wonder why should they pay money to renew their own hardware when the old one is still perfectly working, and the new one does not offer great improvement. Of course this is quite a rational point of view, but the obvious consequence is that even good standard like ISDN or DAB dealt with insurmountable difficulties that turned smart and possible widespread solutions into niche technologies. Even DVB, though being honestly quite innovative compared to old TV standards like PAL or NTSC, is experimenting some of the same difficulties with market inertia: at the time on a whole of almost one billion TV householders there are only 143 million digital receivers, with a ratio of one every seven[2]. Of course one might object that it is not bad as diffusion, but it is undeniable it is still far away from the speed of brand new software spread.

This inertia is a double blade knife, it deals damages not only to the users that may be willing to switch to new technologies but are kept back and forced to use old technologies, and also to developers frustrated for seeing their ideas not having the hoped success.

Now, let's focus: what if one can easily and in a free (or definitely cheap) way upgrade his own technological equipment to get a better service, or to add new functions? What if this could be done remotely from services providers on demand? It is even too easy to forecast that almost every one would be willing to let his system to improve without any cost! It has not to be stressed much that nothing of this is really possible with full hardware components where it would be easily obtained adopting software solutions.

Being already available a completely software DVB-T modulator, developed by Vincenzo Pellegrini [3] and presented at the WSR Karlsruhe conference in March 2008 (http://www-int.etec.uni-karlsruhe.de/seiten/conferences/wsr08/Program_WSR08.pdf), aim of this thesis will be the creation of a prototype DVB-T software receiver, from the ADC (Analog to Digital Converter) to the MPEG-2 (Motion Picture Expert Group) decoder, without focussing on the channel estimation and the timing synchronization, trying to be as near as possible to real time performance.

Software Defined Radio : Benefits And Drawbacks.

As said the main aim of replacing hardware component with software ones is to win market inertia and let evolutionary not just revolutionary technologies access the market. Other great benefits comes when we start talking about research. The chance of experimenting researched solutions in a software-oriented systems is in all way extremely much cheaper and easier than in hardware ones, and, as an example, may boost the ability of researchers to find and implement better and faster decoding or channel estimation algorithm. Of course this does not come fro free.

Very complicated real time algorithm, as we need to make software radio, are computationally very demanding and functions that may require only a 300 MHz ASIC could become too hard even for a 3GHz CPU general purpose computer. This is especially true at the receiver side, where the Forward Error Correction (FEC) work is computed and the channel and timing must be estimated.

But is this a real insurmountable bottleneck? Of course it controversial, but we strongly believe it is not. Smart code writing, threads deserialization on multi-CPU machines, and faster processors may be much of help making even heavy programs good for running at real time.

Benefits of developing Software Define Radio (SDR) are not ended here. Another great feature their intrinsic portability and flexibility: a good, performing and fast code has to be written just one time, then its copies are done free of charge, tearing down the comprehensive cost of hardware based components.

There's an additional point of view. Nowadays traditional media have another growing up competitor: IP (Internet Protocol) TV. In order to not being eliminated by the natural-user selection of technologies, traditional media broadcasting have to keep or even boost its advantages over IP infrastructure, pointing not only toward their intrinsic strong point such as scalability, but also trying to give the user services that may be given form an IP TV, consequentially it is imperative to provide the end user with a state of the art multimedia products, otherwise the consumer will switch for a PC CODEC easily downloaded from the Internet.

Moreover, we are assisting to the diffusion of small cellular phone who perform TV decoding functions, letting people look at what they want whenever they want ans wherever they are. Having a strong tested software able of demodulating a DVB-T signal on common users CPU, would let the technology owner be able to turn any normal Laptop into a TV receiver probably taking back home the “smart-phone” TV market.

State Of The Art : Soft-DVB Modulator

As a first step let us point out the state of art about software defined radio. At the time of writing the open source community has released the 3.1.3 version of “GNU-Radio”, an ensemble of tools created in order to help developers to “translate” hardware operations into soft ones.

GNU-Radio had been used as the framework for SDR real time DVB-T modulator made by Vincenzo Pellegrini. This piece of software is able to put in air a 2 Mbps DVB-T signal, with a 2/3 puncturing convolutional channel coding, perfectly receivable by any DVB-T receiver. An useful feature in creating its “dual” was Soft-DVB's ability -of course not present within the hardware world- of creating dump intermediate files for testing single demodulator parts. This modulator has been largely tested and used, and showed great reliability with contained computational cost that allowed itself to be run even over very low profile desktop and laptop computers.

Of course we all know, by daily experience, that listening is harder than talking (especially if you are trying to listen to a single person inside a noisy crowd form a certain not negligible distance...), and this is also reflected in communication devices by a major amounts of functions and by a major computational weigh of each one beside its dual. As a consequence, the receiver implemented in this thesis will not be real time, but it will still try to be as fast as possible, so it will be work for others to to speed it up.

Lastly, to connect the “hyperuranium” ideas world of software to the material real world Soft-DVB, and consequentially my demodulator, uses a Universal Software Radio Peripheral (USRP) interface, connected via a fast USB port to a common Desktop PC. USRP duties are not really complicated, it must perform a simple conversion with a DAC (Digital Analog Converter), filtering and a translation to Radio Frequency, while all the mathematical operation, coding, scrambling, (i.e. baseband DSP) are computed by software by a general purpose machine.

Chapter 1

OFDM, DVB-T Standard

1.1 Main Features

Digital Video Broadcasting is the, most widely deployed system to deliver both standard definition and high definition video to digital TV users. It is defined as an ensemble of functional blocks performing the modulation of the baseband TV signals from the output of the MPEG-2 coder into the terrestrial channel. Optionally, it is possible to transmit two (high and low priority) MPEG-2 transport streams in hierarchical mode and/or data channel. Video distribution over Single Frequency Network is supported too.

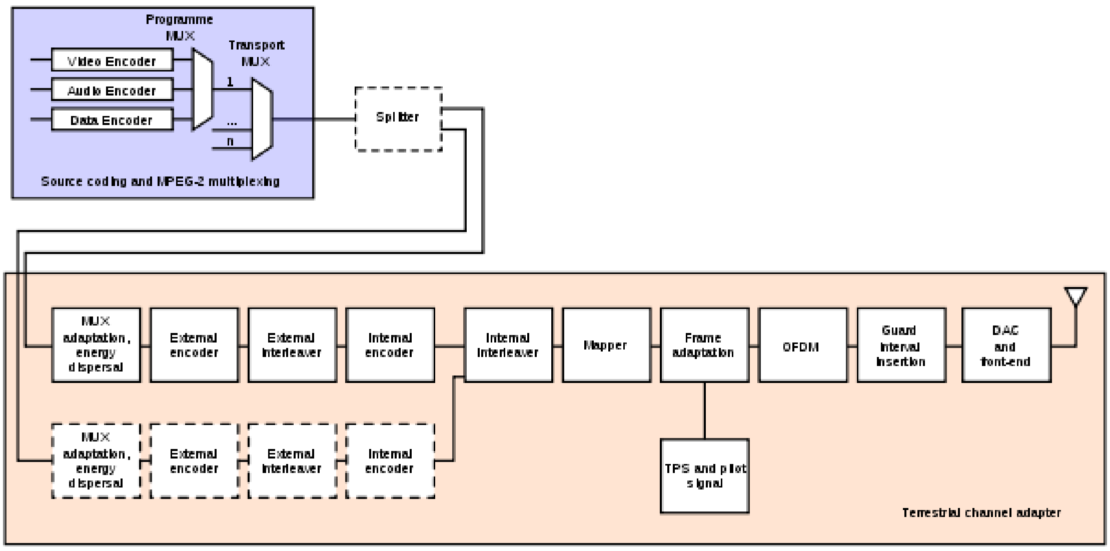


Fig. 1.1: ETSI DVB-T modulator blocks

good DVB needs a typical bandwidth of 8 MHz. Using such a large spectrum we can neither assume nor even hope to experience an AWGN (Additive White Gaussian Noise) flat channel, where a fading, multi-path selective one is more likely.

To avoid the channel problems, rather than carrying the data on a single radio frequency carrier, OFDM (Orthogonal Frequency Division Multiplexing) works by splitting the digital data stream into a large number of slower digital streams, each of which digitally modulate a large number of closely-spaced orthogonal *sub-carriers* are used to carry data. Orthogonality between carriers guarantees the smallest inter-carriers interference while minimizing the space from carrier to carrier thus maximizing the spectral efficiency. In the case of DVB-T, there are two choices for the number of carriers known as 2K-mode or 8K-mode. These are actually 1705 or 6817 active carriers (respectively 2048 and 8192 considering the "virtual" suppressed ones) that are approximately 4 kHz or 1 kHz apart.

Built to be used as an high performing video standard the OFDM signal has to be well protected from errors due to the noisy channel. The the standard provides two error protection codes, an inner and an external one. For DVB-T they would be a convolutional punctured code and a Reed-Solomon coding algorithm.

1.2 Propagation Channel Modelling : Ricean And Rayleigh Channel

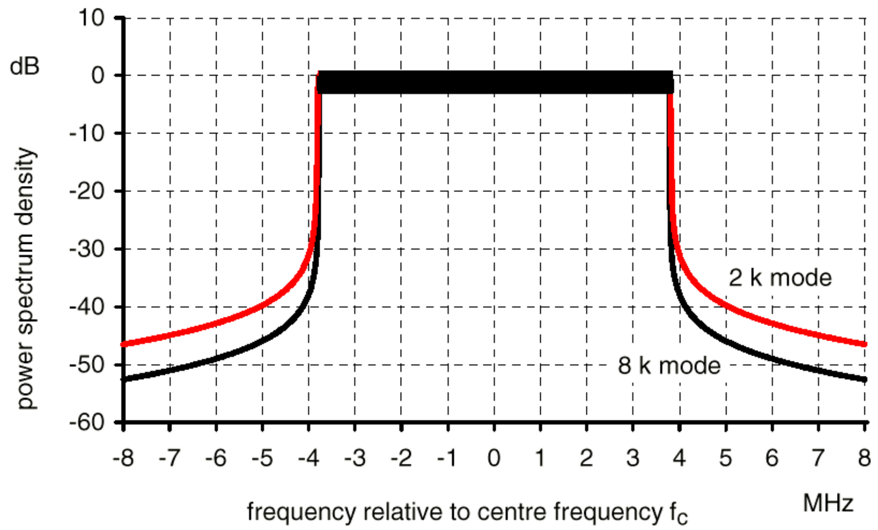


Fig. 1.2: COFDM spectrum

As already pointed out DVB-T typical propagation channel is multi-path, both Line-of-Sight (LoS) and No Line of Sight (N-LoS). The main consequence of the reflection, refraction and scattering of the electro-magnetic wave, beside of the time-variant channel is fading in and echoes.

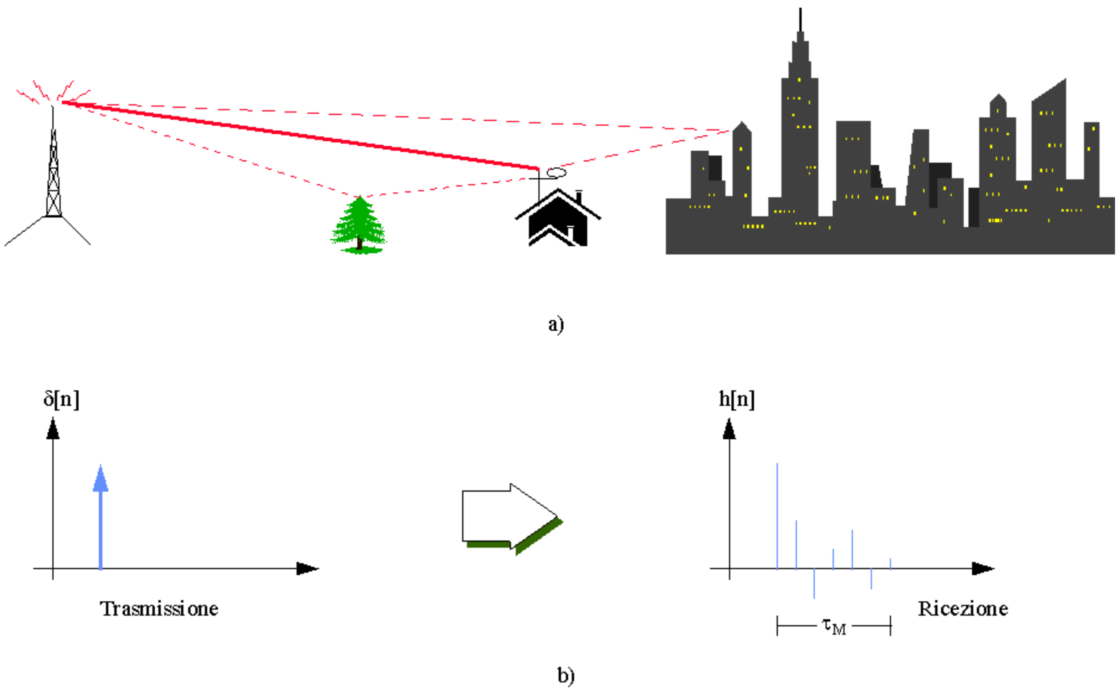


Fig. 1.3 : Multi-path scenario

Designed to provide digital high definition TV services to both urban and rural areas, the DVB-T standard has been developed in order to express good performance in LoS and N-LoS multipath channels. The system was validated by ETSI against the typical multipath channel model, with both Rayleigh (N-LoS) and Ricean (LoS) fading.

In No Line of Sight condition, typically urban areas where we can assume not having a direct ray from the transmitting antenna and the receiving one, channel is statistically modelled as it follows:

• Path magnitude is an aleatory variable R_i with density probability function:

$$f_{R_i}(\rho)=2\rho * e^{-\rho^2} u(\rho)$$

• Path phase is an aleatory variable uniformly distributed:

$$f_{R_i}(\phi)=\frac{1}{2\pi} * rect\left(\frac{\phi-\pi}{2\pi}\right)$$

Otherwise in a more optimistic scenario when it is possible to assume a direct ray, like in rural areas, we may use the Ricean model:

• Path magnitude :

$$f_{R_i}(\rho)=2\rho(k-1)e^{-(k+1)\rho^2-k} I_0\left(2\rho\sqrt{k(k+1)}\right)u(\rho) \quad ;$$

• Path phase:

$$f_{R_i}(\phi)=\frac{1}{2\pi} * rect\left(\frac{\phi-\pi}{2\pi}\right) \quad .$$

Where k is the "Rice factor" $k=\left(\frac{LoS_{RXPower}}{NLoS_{RXPower}}\right)$ and I_0 is the modified Bessel function of the first kind with order zero:

$$I_0(z)=1 \text{over} \, 2\pi \int_0^{2\pi} e^{z\cos(\theta)} d\theta$$.

To ensure a Quasi Error Free condition at the receiver MPEG-2 decoder, the system behaviour has been tested in terms of required Carrier to Noise ratio (C/N). Test result are shown in Table [1.1]. They clearly show the flexibility obtainable with different modulation options, in order to allow transmissions over the various condition of propagation scenarios. ETSI standard also suggests that in order to achieve the QEF condition we must provide enough SNR and enough bit-protection to have an error probability of $P_e=2\text{x}10^{-4}$ after the first error correction function (Viterbi).

It is easy to state DVB-T can put in play really good performance, especially in N-LOS multipath environments such as densely populated urban areas, this characteristic was of great importance in allowing ETSI DVB-T to outperform its American competitor, namely the Advanced Television Systems Committee Standard (ATSC). In fact ATSC relies upon a different modulation system, namely 8-VSB which is much less robust to multipath propagation than DVB-T's OFDM.

Modulation	*Convolutional Rate*	*Required C/N, Gaussian Channel*	*Required C/N, Ricean Channel*	*Required C/N, Reyleigh Channel*
QPSK	1/2	3.1	3.6	5.4
QPSK	2/3	4.9	5.7	8.4
QPSK	3/4	5.9	6.8	10.7
QPSK	5/6	6.9	8.0	13.1
QPSK	7/8	7.7	8.7	16.3
16-QAM	1/2	8.8	9.6	11.2
16-QAM	2/3	11.1	11.6	14.2
16-QAM	3/4	12.5	13.0	16.7
16-QAM	5/6	13.5	14.4	19.3
16-QAM	7/8	13.9	15.0	22.8
64-QAM	1/2	14.4	14.7	16.0
64-QAM	2/3	16.5	17.1	19.3
64-QAM	3/4	18.0	18.6	21.7
64-QAM	5/6	19.3	20.0	25.3
64-QAM	7/8	20.1	21.0	27.9

Table 1.1 Required Channel to Noise ratio to have BER=2x10^-4 after Viterbi decoding.

1.3 Modulator Functional Blocks

Before getting in the heart of receiving functions, it is worth to focus a while on the transmission side and have a look to standard' s directives on functional blocks shown in Fig [1.1]. Of course, not being the main spot of this thesis, I am not going deep inside the modulator blocks that will be briefly summarized :

1. *Multiplex adaptation for energy dispersal* (MAED). It is a stream byte-scrambling unit with the purpose of removing time correlation between bits in the MPEG-2 transport streams by performing a bit-wise XOR with a proper defined PRBS. It also inverts the first byte (namely the SYNC byte) every 8 MPEG-2 frames

2. *Outer encoder.* A typical Reed-Solomon (204-188) encoding procedure derived from a common (255-239) by inserting 51 null bytes in the head of the frame. Its main purpose is to protect the audio and video stream from Viterbi burst errors. As specified by the standard, and better explained later, the Galois Field polynomial generator is: $p(x)=x^8+x^4+x^3+x^2+1$, while the code polynomial is generated by: $g(x)=\prod_{i=0}^{i=15}(x+\lambda^i)$ where $\lambda=02_{HEX}$

3. *Outer interleaver.* A convolutional byte oriented interleaving block based on the Forney approach. Its purpose is removing correlation between casual errors due to a Viterbi failure at the decoding part.

4. *Inner coder.* A widely used convolutional encoder, built from a mother of ½ it is possible to rise the rate with puncturing technique from 2/3

to 7/8. Convolutional generators are: $G_1 = 171_{OCT}$ and $G_2 = 133_{OCT}$. The convolutional is the main error correction block, its purpose is to protect bits form noisy channels in order to obtain the QEF condition.

5. *Inner interleaver.* Composed by a DEMUX and two different interleaver (the first working on bits the second on "words") is needed to avoid time correlation in the errors at the input of the Viterbi decoding block and to avoid, in transmission, to certain bit to be sent on air always in the same carriers with bad Signal to Noise ratio.

6. *Mapper.* It map the bit stream into symbols. It is possible to chose between QPSK, 16-QAM and 64-QAM.

7. *OFDM modulation.* Perform the virtual carriers, TPS and pilot carriers insertion into the signal before computing the IFFT.

8. *DAC, Digital to Analog Converter.* As the name suggest, it performs the conversion from deigital samples to analogue signal by means of interpolation.

9. *Radio Frequency front end.* It shifts the base-band signal to its proper frequency for the desired TV channel and sends it to the aerial.

Chapter 2

GNU-Radio Framework

2.1 Python C/C++ Architecture

GNU-Radio is a free software development tool-kit created to build and test and defined radios. Its main characteristic is to provide the signal processing runtime, the flow control between implemented "blocks" (where the typical communication functions happen) and to handle bufferization and the exchange of data. The use of GNU-Radio allow to implement the user with a strong communication background and a good knowledge of C/C++ languages to create software defined radios using readily-available, low-cost external RF hardware and general purpose commodity processors.

GNU Radio applications are primarily written using the high level scripting language Python, which main scope is providing GNU-Radio a data flow abstraction. Its fundamental atoms are "signal processing blocks", implemented in C/C++, doted of one or more input and one or more output ports. These blocks, are pre-implemented classes where the developer must, generally speaking, override some member functions in order to obtain the desired work. Their positions and their connections are organized into a "flow-graph". Besides, the Python has the purpose of dealing with the USRP

(Universal Software Radio Peripheral: our external Radio frequency terminal), in order to do so the Python runs the C/C++ classes needed for the USRP to work. Thus, being all the hard (computationally heavy!) work done by the C/ C++ code the developer is able to implement real-time, high-throughput radio systems in a simple-to-use, rapid-application-development environment.

Moreover, the framework comprehends a list of pre-written blocks to perform basilar and common telecommunication functions. This is comprehensive of FIR filers or FFT transform, in addiction to blocks needed to handle the data structure in the graph.

While not a simulation tool, GNU Radio does support development of signal processing algorithms using pre-recorded or generated data stored into files, avoiding the need for actual RF hardware usage. This comes in handy when you have to validate and test performances of single or group of implemented processing blocks before being able to use it with Radio Frequency real signals. As an example, any new idea for implementing a demodulation function could be just implemented and tested firstly by itself, with the proper input and then with all the systems.

```
import threading
from gnuradio.gr import firdes
import Numeric



class qa_howto(gr.top_block):

   def __init__(self):
       gr.top_block.__init__(self)

       src = gr.file_source (16384, "/home/Luca/Scrivania/script/input.dat",False)
       dst = gr.file_sink (1, "/home/Luca/Scrivania/script/output.dat")
       dst_fft = gr.file_sink (16384, "/home/Luca/Scrivania/script/output.dat")
#        dst1 = gr.file_sink (8, "/home/Luca/Scrivania/script/output.dat")
#        dst = gr.null_sink(1)
       inner  int = rdvb.innerinterleaver  bb()
```

Fig 2.1 : Pyhton code example

2.2 Gnu-radio Blocks And Functional Blocks

As already shown in previous chapter, ETSI DVB-T standard determines a number of functional block for the modulation part which make up the simple MPEG-2 audio-video stream ready for the RF, but, of course, it does not tell anything for what is about the receiver part. The path to follow, then, will be of implementing into each GNU-Radio block a dual for every functional block in the modulator. The philosophy of using a GNU-Radio block for each standard function has shown herself to be the best trade-off between speed throughouts (that would anyway be slightly incremented by using just one block to perform all the decodification process) and code readability and portability. This doing will result in the "chain" of blocks shown in Fig 2.2. Of course in projecting these duals there is high degree of freedom particularly in choosing the best algorithms to perform the needed functions.

Having in mind the goal of taking the signal from the antenna to the monitor with a test bench receiver (this means avoiding the channel estimation and the synchronization, functions that may as well be introduced later), the difference between riding the wave (and consequentially implementing from the next-to-antenna block) or going backward from the one nearest to the MPEG-2 decoder is just a matter of strategy. Both paths present their peculiar advantages and drawbacks. Going the straight way, surely would have allowed to set the entire data structure step by step, and furthermore it is somewhat more "natural", but on the other side it would be quite an effort to check the correctness and the functionality of each function. As an example it would not at all be easy to understand if the channel estimation and the timing synchronization were done correctly, having to wait until the last block to be really sure. Otherwise the going backward option is surely more useful in debugging operation. This because the only thing to do to check if a new block works or not is to connect the entire system, run it, and have a look to the decoded video. The bad part of this implementation strategy relies in the synchronization between blocks, in fact it is quite hard to think and implement a working systems while not aware of what will definitely trigger everything on. In fact, as will be better explained later, there are some parts that cannot work in "stream" mode, but need a vectorized data structure, they need, in simple words, to work on groups on N bytes. But this opens a problem: when they must start to consider a byte at their input a valid byte? When the receiving byte will be stream byte and not just noise? The problem has been solved by looking some transport stream known byte, (namely the SYNC and inverted SYNC bytes), but still it would be usefull, when everything will be ready, to define an inner way to trigger on and to trigger off each demodulator part.

All summed up, and considering GNU-Radio pre-implemented blocks, which can make setting up of the data structure neither difficult nor really effective on performance, estimating the benefits would outmatch the drawbacks we adopted the reverse way strategy, from the video to the antenna.

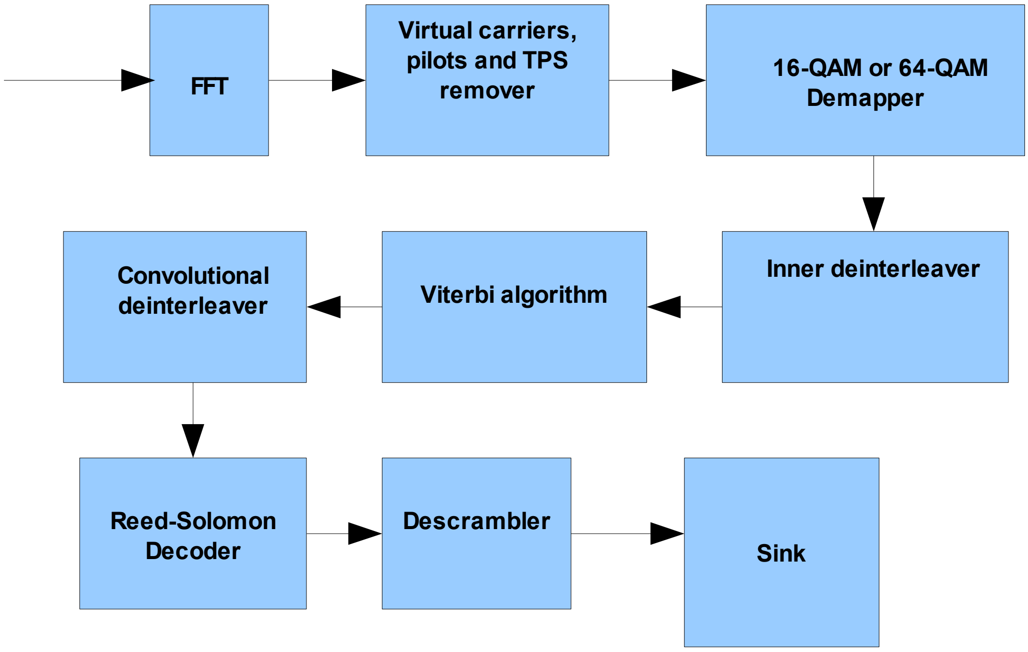


Figure 2.2 : Receiver functional blocks

Chapter 3

Demodulator Blocks : R-dvb.

3.1 Descrambler

Before letting the base-band video stream be processed by the MPEG-2 decoder it must be de-randomized. The randomization process takes part in DVB standard to perform a M.A.E.D. (*multiplex adaptation for energy dispersal*). In order to do so the scrambler computes a bit-XOR between the video stream and a PRBS (*pseudo random binary sequence*) generated with a linear feedback shift register (LFSR) by the generator polynomial: $p(x)=1+x^{14}+x^{15}$ initialized with the sequence: 100101010000000.

The MPEG-2 transport packet is composed of 187 bytes + the SYNC byte (0x47) at its head, following ETSI directive to provide an initialization signal to the descrambler the SYNC byte every 8 transport packet is bit-wise inverted from 0x47 to 0xB8. The SYNC bytes had not be randomized, thus they must not be de-randomized but just inverted one every eight, this has been easily achieved by computing the bit-XOR operation with 0xFF.

Finally every eight transport packets the initialization sequence, the seed, must be reloaded into the LFSR, doing so will result in our PRBS having a periodicity of 1504 bytes. By this point of view, being the XOR the base operation a descrambler is almost the same of a scrambler, except for the synchronization matter.

The inner periodicity of this descrambling operation suggest a fast and easy implementation strategy: pre-calculate during initialization the 1504 bytes composing the PRBS and store them in a vector which will be used when needed to perform the XOR.

As can easily be deducted, the synchronization between the first SYNC byte (the first 0xB8) and the descrambling operation is all-important, so it is demanded to the block to recognize a good sync byte and line up its PRBS.

As an additional feature, the Descrambler implements a BER-o-METER to simply evaluate the signal corruption level after the FEC decoding states. The estimation is done by working out the hamming distance between the received sync byte (after synchronization has been recovered) and the expected byte (both 0x47 and 0xB8).

Moreover this BER estimation is used to avoid false SYNC alignment. It is more than obvious that mistaking an inverted SYNC would result in a BER very similar to ½, thus if the BER goes over a limit the block stops its descrambling work and starts looking for a new inverted SYNC byte.

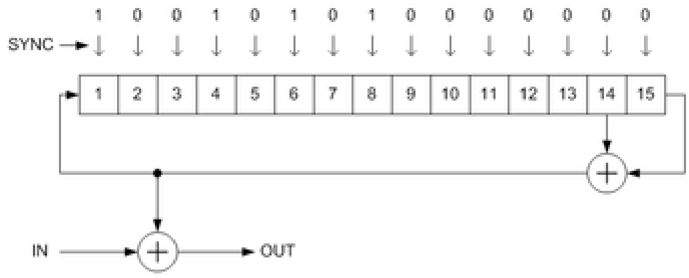


Fig. 3.1 : Descrambler schematic block

3.2 Reed Solomon Decoder

It is well known that an MPEG-2 video stream is very compressed, but its efficiency in terms of information bit for binary symbol had a price: fragility and great susceptibility to errors. As a consequence any corrupted bit may comport substantial degradation in the video quality and has to be avoided.

To provide the end user a high definition video experience, the standard expects the system to be able to put out a QEF (*Quasi Error Free*) MPEG video stream, where the QEF condition is obtained when $BER < 10^{-11}$. As a direct consequence, the system must have some very good error correction algorithm, without inserting too much redundancy. The concanetion of Viterbi (convolutional) and Reed Solomon (RS) FEC is the solution adopted by ETSI.

Reed Solomon coder is a systematic code (this means that a portion of output word includes the input in its original form) with little insertion of parity bytes (in DVB-T RS rate is only 1,085 16 parity bytes every 188 information bytes). At any rate its main interesting characteristic, being byte oriented, is its intrinsic ability to perform well against "burst" errors, the typical kind of mistake a Viterbi decoder might do. In fact from an RS point of view a byte which has just a single erroneous bit and a byte completely mistaken are “wrong” to the same extent.

The Reed Solomon implemented in DVB-T is a shortened (204,188) code, built from a classical (255,239) where the 51 remainder bytes are all set to zero during the coding procedure and consequentially not transmitted. The first assignment of a decoder then will be re-inserting this 51 zeros, and then compute the decoding process. Such a code is able to correct up to 8 erroneous bytes put everywhere inside each 204 bytes word.

In order to understand how this code works we must afore have an introduction in Galois (or finite) Field Algebra on which relies cyclic code. For each prime p does exist a Galois Field GF(p) made of p elements, this field may be extended in field GF(p^m) where “m” is an integer greater than one. The Galois field we need for our Reed Solomon code is GF(2^8) so we can arrange binary words of 8 bit (in other words: a byte).

In a Galois field must be defined two operations: sum and product, with the following properties:

• closure (if a and b are elements of GF(m) then also a + b and a x b are elements of GF(m)

• associativity, commutativity, distributivity

• existence of the neutral element

Beside the elements 0 and 1, whose existence is given by definition, there will be a *primitive element* a such each and every not null element f the GF may be represented as a power of a.

As we can see, then, we will have rings of sum and multiplication such as,given a element of GF $a^{p^m-1}=1=a^0$. For our need it becomes $a^{255}=1$, while the addiction will easily be the binary XOR.

A class of polynomials called *primitive polynomials* is of interest as such functions define the finite fields GF(2^m) that in turn are needed to define RS codes, in our case we shall have $1+X^2+X^3+X^4+X^8=0$ as our field generator polynomial and $a=0x02=00000010$ as our base element.

As a consequence it is easy to state that every possible codeword can be mapped into a Galois Field element, so the RS decoder block, during its initialization phase, will generate the field and store this result into a vector, so that converting bytes into powers and powers into bytes could the easiest (and fastest) possible. During this initialization phase, that take place in the class constructor, the block will also create a “multiplication matrix” and an inverse vector, in this way every possible operation needed in the decoding phase will be copmleted with just a look-up.

3.2.1 Syndrome Computation

But why we need all this mathematical stuff? It is fast explained. The coder interprets the 188 byte as coefficients of a polynomial expressed with Galois Field's elements, naming d(x) the word to be coded (and thus the one we wish to decode..) we have: $d(x)=d_0+d_1*x+d_2*x^2+....d_{187}*x^{187}$

The coder now will use another polynomial namely the *code generator polynomial* g(x) that characterizes the code:

$$g(x)=(x-a^0)*(x-a^1)*(x-a^2)*...*(x-a^{15})=\prod_{i=0}^{15}(x-a^i) \quad .$$

to evaluate the code-word c(x):

$$c(x)=x^{16}*d(x)+p(x)$$

where p(x) is

$$p(x)=x^{16}*d(x) \, mod \, g(x)$$

Consequentially it is quite easy to state that $c(a^i)=0 \, \forall \, 0 \leq i \leq 15$ that is the condition we need in order to understand how correct is the received word.

Now, the first thing the decoder does is to compute the *Syndromes evaluation.* These Syndromes are obtained by just evaluating the received polynomial named R(x) in the roots of g(x) . In fact, if the received bytes are correct it will result $R(a^i)=c(a^i)=0$. Calculated syndromes are then organized into a polynomial in the following way: $S(i)=R(a^i)$.

If all these syndromes are equal to zero then the codeword is correct and the only function of the decoder will be eliminating all the parity symbols, else way it has to try to correct the errors.

3.2.2 Key Equation Solving: Berlekamp-Massey

Lets assume the Syndrome polynomial is not a null one. Next step is locating the errors position and evaluate the entity, the magnitude of this errors. In order to achieve this result we must solve the *key equation*, a non linear system that links syndromes to errors and their position. Solving by common way this system would be much an effort so the problem is split into 2 steps:

1. Evaluate an *error locator polynomial* C(x), whose roots are the positions of the errors.
2. From C(x) evaluate a *magnitude polynomial* .

In order to evaluate C(x) two algorithms have been mainly proposed, the Euclidean and the Berlekamp-Massey. The first is easier to implement but heavier, the latter then has been chosen to be implemented due to his lesser computational cost.

Berlekamp-Massey algorithm pseudo-code steps are the following:

1. Initialization of variables: $C(x)=1; D(x)=x; L=0; n=1$
2. Discrepancy computation:
$$\delta=S_n+\sum_{i=1}^{L} C_i * S_{(n-i)}$$
3. Discrepancy test, if $\delta=0$ go to step 8, else go to step 4
4. Error location polynomial modification:
$$\hat{C}(x)=C(x)-\delta D(x)$$
5. Registry length test: if $2L \geq n$ then go to step 7, else go to step 6
6. Registry length and correction term modification:
$$L=n-L$$
$$\hat{D}(x)=C(x)/\delta$$
7. Error locator polynomial update:
$$C(x)=\hat{C}(x)$$
8. Correction term update:

$$D(x) = \hat{D}(x)$$

9. Element counter update:

$$n = n + 1$$

10. Syndrome element number check: if n<16 go to 2, else stop.

If everything is done correctly, it will result in a polynomial C(x) which roots are the "position" of the errors, of course now it is time to find out where this roots are, in order to do this we use a well known algorithm known as the "Chien search".

3.2.3 Chien search

As soon as the demodulator completes the Berlekamp-Massey Algorithm it has to look for C(x) roots, to locate the errors in the codeword. Being working in Galois Field the easiest way to do it is an algorithm known as *Chien search.* This is nothing more than an exhaustive search: it just evaluate $C(a^i)$ for every possible i from 0 to 254: if it is zero then a^i is a root and consequentially "i" is the position of the error.

To save time, when eight errors have been found the Chien search will be stopped. Our Reed Solomon code can correct up to 8 errors, so it is of no use to go on searching.

3.2.4 Forney Algorithm

Now that we know where erroneous byte are we still lack information, in fact the decoder must be able to find the *magnitude* of the errors, to let correction be possible. This is achieved using the Forney Algorithm.

First of all we compute the *error magnitude polynomial* as it follows:

$$\Omega(x)=[(1+S(x))*C(x)]\,mod\,x^{17}$$

The second step is calculating the formal derivative of C(x). This is quite easy, not only because C(x) is a polynomial but also because, due to the XOR nature of the addictive operation in Galois Field algebra, the even powers will always have null derivative.

At this point our block must evaluate the error amplitude:

$$e_k=X_k\frac{[\Omega(X_k^{-1})]}{[C'(X_k^{-1})]}$$

where C'(x) is the upper defined formal derivative and X_k is the k-th root of C(x).

Once iterated the Forney algorithm for each X_k it is time to correct our code word, as said the position of the error will be $degree(X_k)$ and its entity e_k .

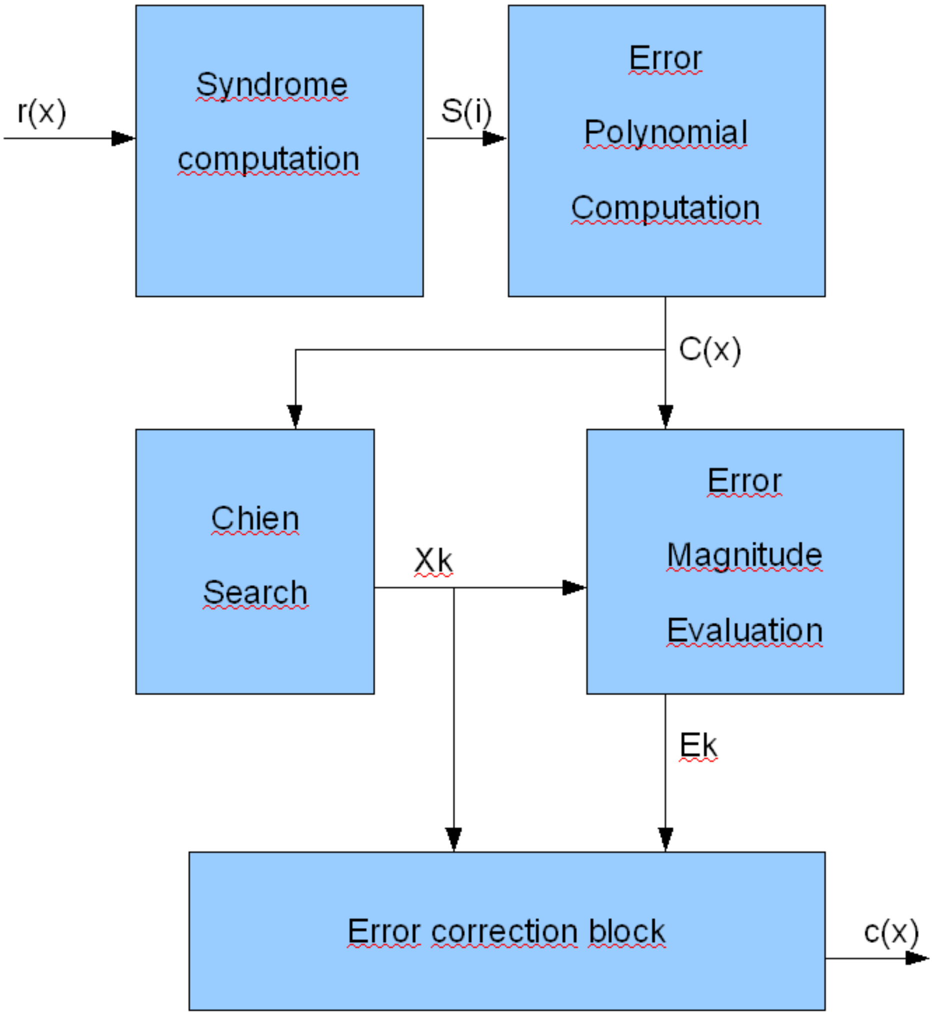
Now, the only remaining steps are to sum (or better XOR) the corrupted byte with the correspondent “error”, to discard the 16 parity bytes, and our corrected word is ready to be de-scrambled.

3.2.5 Considerations

As previously stated, this Reed Solomon decoder can correct up to 8 errors, where errors are corrupted bytes whose position in the codeword and amplitude are both unknown. But what does happen when there are more than 8 erroneous byte in the codeword? There are two possibilities.

The first one happens when the decoder "understand" it is trying to correct something that is over its correcting ability and then give it up, copying in its output the 188 information bytes without performing any correction and hoping most of the errors were in the parity bytes. This acknowledgement happens when the formal derivative of the error position polynomial evaluated in the inverse of C(x) roots during the Forney algorithm becomes zero.

The second one happens when the decoding algorithm is completely fooled by the errors, and the received codeword looks like another valid codeword. In this case, which usually happens with lots of errors, the decoder would not limit itself to neutrality, but it would even introduce newer errors, by "correcting" the not corrupted bytes. At any rate this chance is not critical. Whenever a signal is so bad to confuse the Reed Solomon it is probably too noisy also for the other blocks, and we should not forget that the system is thought to give the user a QEF video stream, so the unlucky event should be avoided by other means.



Legenda

r(x)=received codeword
S(i)=Syndromes Polynomial
C(x)=Error locator Polynomial
Xk=k-th C(x) roots
Ek=k.th roots erros
c(x)=recovered word

Fig. 3.2 : Reed Solomom decoding blocks

3.3 Outer Interleaver

Between the two error protection blocks, the Vietrbi and the Reed Solomon decoders, the DVB-T standard provides a state of *convolutional interleaving.* The rationale behind this becomes quite clear once stated that typical Viterbi errors are burst errors, and that Reed Solomon really improve its performance in presence of "uncorrelated" corrupted bytes.

Interleaving is a technique commonly used in communication systems to overcome correlated channel noise such as burst error or fading. It rearranges input data such that consecutive data are split among different blocks so that the latter error correction block may be capable of making them up.

As shown in Fig[3.3], the de-interleaver is composed by 12 FIFO (First In first Out) shift registers (namely from 0 until 11) which are cyclically connected to the byte stream, both as input and as output. The first shift register of the interleaving function does not have buffers, as a consequence the interleaver results conservative about SYNC (0x47) and inverted SYNC (0xB8) bytes. This is very important during the dual function. In fact, would the de-interleaver mistake the first inverted SYNC byte and put it in a incorrect line, it would result in an completely unreadable and unrecoverable video stream. Knowing this the de-interleaver must keep an eye on the first shift register and expect to see there a SYNC every 17 bytes and an inverted SYNC every 8 SYNCs.

The latter function is implemented through a BER-qatch over the expected SYNC bytes. The difference of course relies on the threshold, being before the Reed Solomon block we are forced to tolerate an higher error ratio, thus whenever this BER goes over 10^{-2} the block will assume to be not correctly aligned and will perform an inverted SYNC search in the stream.

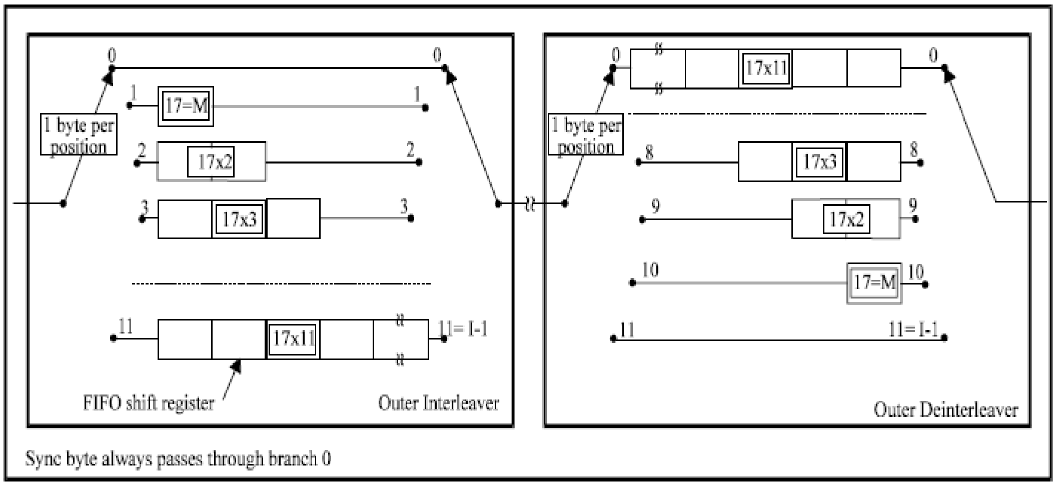


Fig. 3.3 : Convolutional Interleaver and Deinterleaver

3.4 Viterbi Decoder

Followoing the DVB-T directives the channel error protection code is a convolutional code with constraint length L=7 and generator polynomials $G_x = 171_{OCT}\, G_y = 133_{OCT}$, which conceptual scheme is reported in the following figure.

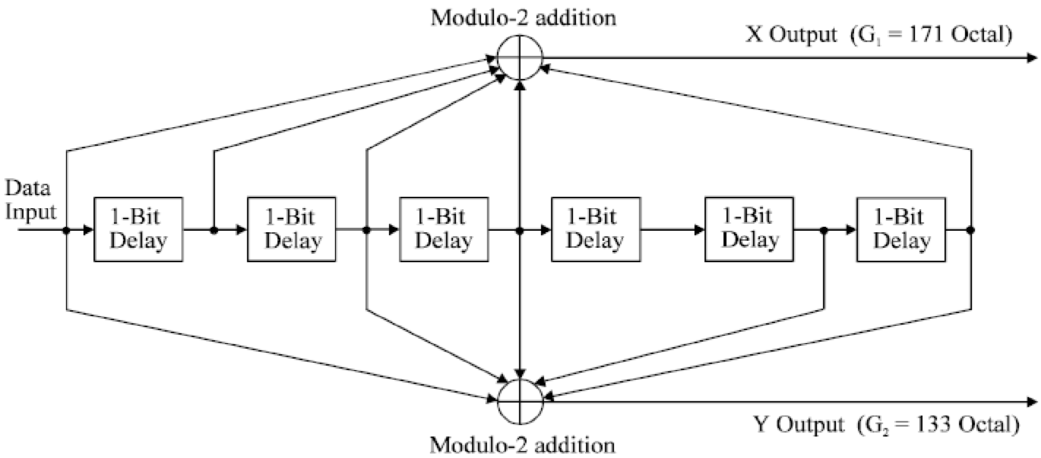


Fig. 3.4 : Mother convolutional with rate 1/2

The classical method of decoding convolutional codes has been proposed by Andrew James Viterbi. The Viterbi Algorithm (VA) is a recursive process which consists in finding the most-likely state transition sequence in a state diagram, given a sequence of symbols. In practice, the VA is observed as a finite-state Markov process whose representation is either a state transition diagram or a trellis.

The mother code rate imposed by DVB-T standards is 1/2. However, higher code rates such as $\frac{2}{3}, \frac{4}{5}, \frac{5}{6}, \frac{6}{7} or \frac{7}{8}$ can be derived from the mother code, by simply introducing "puncturing".

On the encoder side, the so called puncturing consists in deleting a certain number of bits in the encoded stream according to perforation patterns (as shown in Fig [3.4]) which indicate the positions for bits to be deleted.

It is important to stress that the decision depth is lengthened to almost 15*L, where L indicates the constrain length of the convolutional code (for DVB compliant convolutional L=7), then the classical Viterbi Algorithm can be applied. Thus in the implemented Viterbi was needed a decision depth of 105, but as already affirmed we use 64-bits registers to memorize path , as a consequence the used memory must be an integer multiple of 64. For such reason it has been decided to choose 128-bits memory register realized with a vector of two 64-bits buffer.

The coding rate selected in Soft-DVB modulator is most common in commercial use, and also the one selected for Italian DVB-T transmission. It is a good trade off between error protection and redundancy: 2/3. On the decoder side, depuncturing can be obtained following at least two strategies.

The first way consists in inserting, in place of the deleted symbols, an a-priori-known symbol equidistant from both "0" and "1". This means to substitute the standard Hamming distance with a doubled one where distance from 1 to 0 is 2 and distance between the known symbol and 1 (or 0) is one. It is easy to think at this symbol as at ½. In order to do so without wasting time the block sets a distance matrix by pre-calculating it during initialization phase.

Another strategy is the one of considering the punctured 2/3 as an actual real 2/3 convolutional and then decode it in the very standard way.

Both path have been tried and tested, while they have shown the very same attitude against noise the latter Viterbi has shown to be slightly faster and it looks also a cleaner approach than the first, so it was chosen as the convolutional decoder implementation.

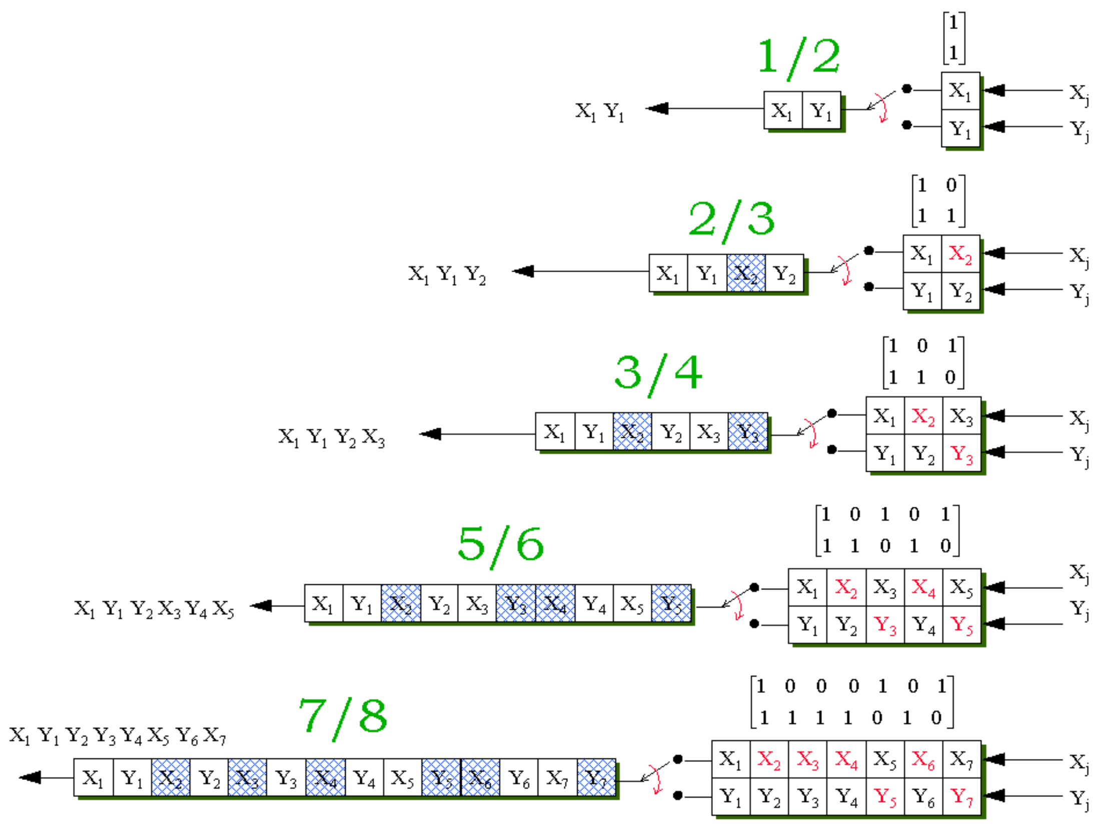


Figure3.4 : Puncturing scheme

3.4.1– Initialization and butterfly creation.

The Viterbi algorithm has shown to be quite computationally expensive, even if enormously better than an exhaustive strategy. This means that the first thought when implementing a Viterbi is to pre-calculate everything that can be pre-calculated. This will, of course, slightly slow down the starting phase but it will really boost the decoding time.

During this start-up then the implementation will build up the distance matrix (a simple matrix whose inputs are all the possible labels and all the possible triples of bits and with the Hamming distance as output), and will create the "butterfly", a sort of finite state machine which stores all details about Viterbi states (paths, branches label..). An example of a simple butterfly for a 4 states Viterbi is shown in Fig[3.5].

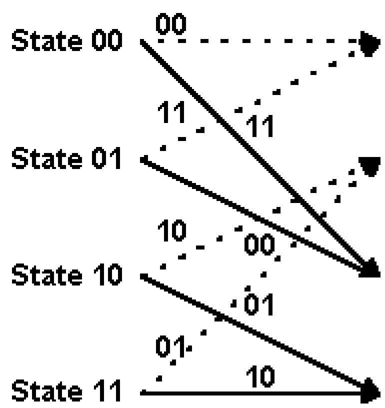


Fig. 3.5 : Four states butterfly with labels

As provided from the standard, and easily derivable from the convolutional scheme, the encoder will have 6 shift registers and this will result into a $2^6 = 64$ states butterfly.

3.4.2 Branch Metrics

Once passed the start up phase, the branch metrics computation is quite straight forward. The decoder must just take the three bits at the input of the Viterbi and evaluate the Hamming distance with the pre-calculated labels, using both the input and then label as input for the distance matrix instantiated in the previous phase.

3.4.3- Add Compare Select (ACS)

Once computed the branch metrics it comes to add those to the previous state accumulated metric with the goal of selecting, for each state, the smallest one.

Once done, each state will update "path", 64 two elements vectors made of 64-bits registers who stores the Viterbi-path in the trellis in the form of output decoded couple of bits. When this buffers are full, the one with the smallest metric associated will be the one selected to send to the output interface the decoded couple of symbols.

3.4.4- Consideration

As can easy be derived from the description, Viterbi algorithm implemented is a classical Hard-Viterbi, this means that it may take as input only already de-mapped symbols, and indeed it computes the branch metrics by calculating an Hamming distance while an Euclidean one would be needed to realize a soft Viterbi.

Of course it would be possible to implement a soft version which should provide us of the known further 2 dB in coding gain, but this would coerce us to evaluate no longer just an Hamming distance, but an euclidean one probably becoming quite too heavy for the CPU that has already being showing great stress with the “normal” weight of the hard Viterbi.

3.5 Inner De-Interleaver

It's common knowledge that Viterbi decoders have their weak point in strongly correlated errors patterns, in other words they have difficulties in recovering a bunch of erroneous bits the one too close to the other. Bearing in mind the channel fading effect and the nature of the noise, particularly in broad band multi-path communications, as well as the possible interference from other source (intentional or casual) it is really expected that errors will likely occur in burst.

The purpose of the inner interleaving, then, is shuffling the bit stream so that dangerous and destructive events would not tear down the whole system performance.

This block is the first block who changes his behaviour in accordance with the transmission mode, it means it considerable changes if you are using 2k or 8k mode, and it also is mapping-dependent. As stated two demodulation modes were implemented, 16-QAM 2k, and 64-QAM 8k, both in not-hierarchical mode. The functional scheme of the two possible interleaving functions is shown in Fig[3.6] and in Fig [3.7].

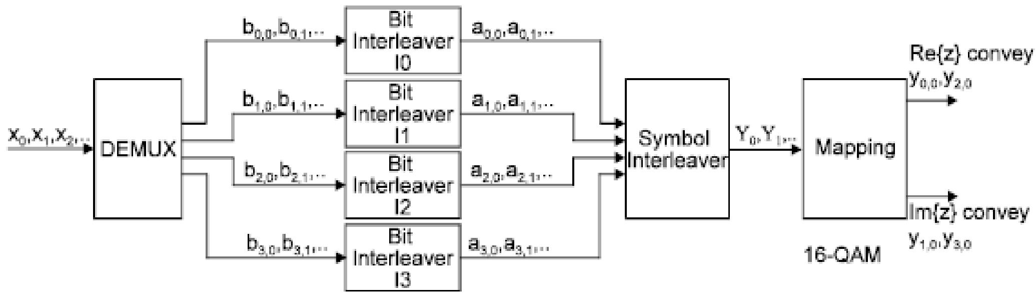


Fig. 3.6 : Inner Interleaver, Mode 2k 16-QAM Not-hirarchical

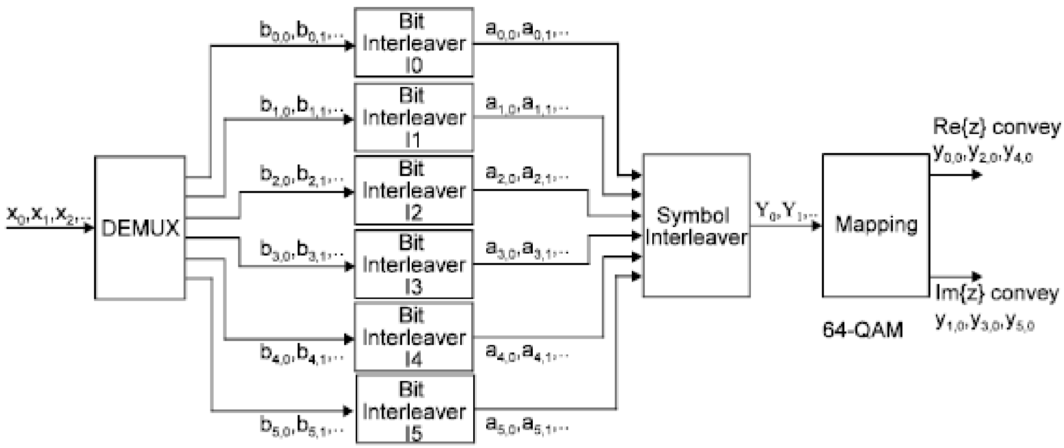


Fig. 3.7 : Inner Interleaver, Mode 8k 64-QAM Not-hirarchical

There are, of course, several possible strategies to implement a dual block of such an interleaver, one could be reversing each and every function starting from the last to the first function, but such solution being simple it has its pay-off in computational weigh. Thus, the selected option has been observing the interleaving function mapping a generic bit at its input into a place at its output, and then inverting the formula.

A simple mathematical analysis of the functional blocks tells us it has got a 3024 (2k mode) or 12096 (8k mode) bit periodicity. In fact it is the periodicity of the last function, the symbol interleaving, while bit interleaving blocks have a 126 periodicity and the de-mux a 4 (2k) or 6 (8k) one. At this point, the implemented system will read form the input buffer “vectors” of right dimension (3024 or 12096 bits) so it may de-shuffle their elements.

3.5.1 Formula evaluation

So let us assume the generic bit at the input of the interleaver position to be “n”. Due to the interleaver parts different periodicity, it is necessary to divide the total bit ensemble in different dimension group.

First of all, the Demultiplexer works every v bit, where v=4 for 2k mode and 6 for 8k, we need to know which place the bit occupies inside group of v bits and this is achieved by calculating e0=(n mod v). The de-mux state, then, will put the bit in the right branch “b” (b may vary from 0 to v-1) according to the standard's rule:

16-QAM	64-QAM
e0=0 => e1=0	e0=0 => e1=0
e0=1 => e1=2	e0=1 => e1=2
e0=2 => e1=1	e0=2 => e1=4
e0=3 => e1=3	e0=3 => e1=1
	e0=4 => e1=3
	e0=5 => e1=5

Let us call this demultiplexing function Hd, so that will easily result e1=Hd(e0).

Once done, the bit will enter into the bit-interleaving state, whose permutation formula is depending on the branch selected.

The possible formulas are:

$H_0(w) = w \, mod \, 126$

$H_1(w) = (w + 63) \, mod \, 126$

$H_2(w) = (w + 105) \, mod \, 126$

$H_3(w) = (w + 42) \, mod \, 126$

$H_4(w) = (w + 21) \, mod \, 126$

$H_5(w) = (w + 84) \, mod \, 126$

Of course the subscript indexing these permutations corresponds to the upper calculated e1. Clearly, if we are using the 2k mode, it cannot be over three, in this case only the first four permutations (from H_0 to H_3) will be used.

Now we have the problem of linking the position inside the e1 branch called "w" with n. It happens it is quite easy! In fact, lets name gr0=n/6, where "/" is the integer division, gr0 is telling us the bit 's position inside the branch, bethinking that we are going to perform a 126 bits periodicity interleaving, it is useful to calculate also "w0" as w0=gr0 mod 126. Naming w1 the exit position it will result $w1 = H_{e1}(w0)$ and the absolute new position inside the branch $gr1 = gr0 + w1 - w0$.

As it results clear from the figure, those branches flow together into the symbol interleaver. This block computes a vector permutation following a DVB standard defined function $H_q(q)$. Just like the bit interleaver one, this permutation is calculated once for all in the class constructor, during initialization phase.

The pseudo code algorithm for $H_q(q)$ is the following.

Set Mmax=2048 for 2k mode and Mmax=8192 for 8k mode, then set $N_r = log2(M_{max})$ and i = 0 , 1 $R_i^{'}[N_r-2,N_r-3,.......,1,0]=0,0,0,0,0..0$

i = 2 $R_i^{'}[N_r-2,N_r-3,.......,1,0]=0,0,0,0,0..0,1$

2 < i < Mmax: $R_i^{'}[N_r-3,N_r-4,.......,1,0]=R_i^{'}[N_r-2,N_r-3,.......,1]$

2k mode: $R_i^{'}[9]=R_{i-1}^{'}[0]\,xor\,R_{I-1}^{'}[3]$

8k mode: $R_i^{'}[11]=R_{i-1}^{'}[0]\,xor\,R_{i-1}^{'}[1]\,xor\,R_{i-1}^{'}[4]\,xor\,R_{i-1}^{'}[6]$

Now we must derive Ri from R'i by bit permutation given in Tab.

Bit permutations for the 2K mode

R'i bit positions	9	8	7	6	5	4	3	2	1	0
Ri bit positions	0	7	5	1	8	2	6	9	3	4

Bit permutations for the 8K mode

R'i bit positions	11	10	9	8	7	6	5	4	3	2	1	0
Ri bit positions	5	11	3	0	10	8	6	9	2	4	1	7

Finally, $H_q(q)$ is defined:

q=0

0<i<Mmax :{ $(i\,mod\,2)*2^{N_r-1}+\sum_{j=0}^{N_r-2} R_i[j]*2^j$;

$if\,(H_q(q)<N_{max})\,q=q+1$; }

Lately, the symbol interleaver will take our bit in “gr1” position in the “e1” branch and put it in $H_q(gr1)$ position when the OFDM symbol is even (this occurs when n<1512 for 2k or when n<6048 for 8k) it will instead put it in $H_q^{-1}(gr1)$ if it happens to be odd.

In the end, we can state that the whole inner interleaver will put the generic n-th bit in position “o” where

$o=gr1*v+e1$.

Now, the hard part being done off-line, to invert this formula and performing a de-interleaver it is sufficient to take the o-th bit at its input and map it at its proper output position.

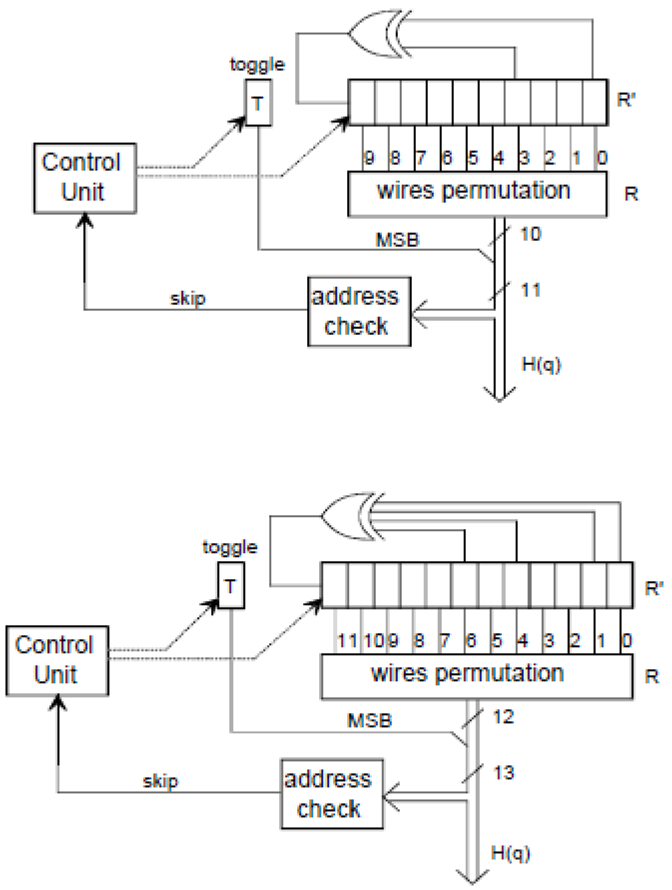


Figure 3.7 : Symbol Interleaver addres computation scheme for 2k (upper) and 8k (lower) modes

3.6 De-mapper

When the signal comes to this block it is composed of complex baseband symbols embodied by two floating point characters representing respectively symbol's real part (in phase) and imaginary part (quadrature).

This block has been implemented for both 2k-16QAM and 8k-64QAM transmission modes with a common threshold decision following the constellations shown in Fig[3.8] and Fig[3.9]. The output, in order to be consistent with the previous block, must be a bit stream.

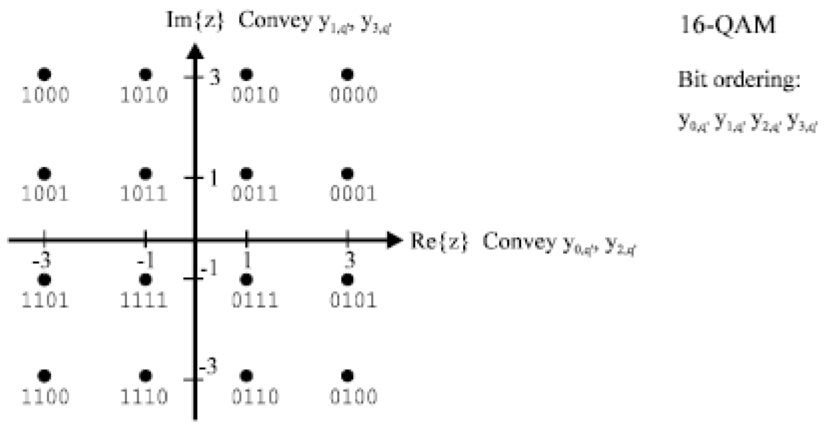


Fig. 3.8 : Uniform 16-QAM Mapping and bit patterns

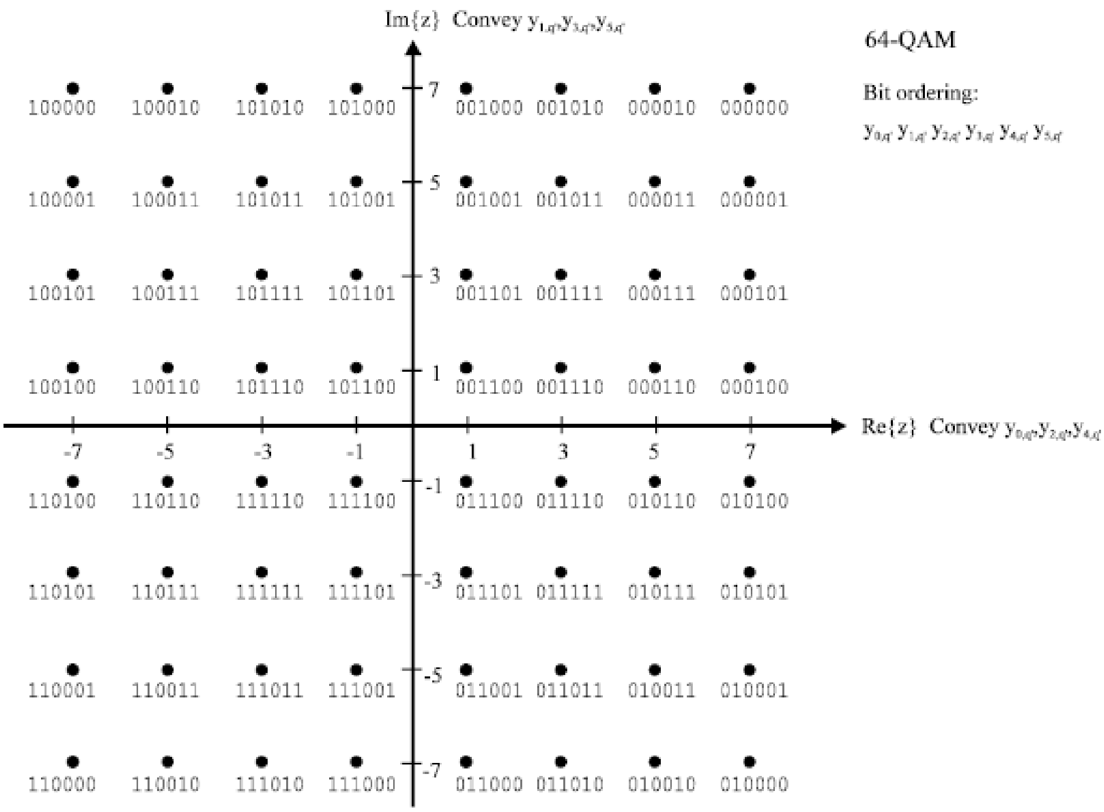


Fig. 3.9 : Uniform 64-QAM Mapping and bit patterns

3.7 Not Information Carriers Removal

The transmitted signal is organized in frames, each with duration of Tf consisting of 68 OFDM symbols. Four frames constitute a super-frame. Each OFDM symbols is made up of a set of 6817 carriers in the 8k mode and 1705 for the 2k one, and it is transmitted with a duration of Ts. To resist to multipath channel the OFDM structure need some carriers not to carry video or audio information but to be pilots that allow a correct channel estimation. Pilots may be fixed or scattered, the latter being useful to avoid a multipath channel notch to delete all the information about a single pilot. Other carriers are needed for synchronization purposes and to transport transmission parameters to the receiver such as used constellation, frame number, hierarchy informations. Moreover after the 1705 (6816) information and pilot carriers the DVB-T's OFDM provide the insertion of 343 (1375) "virtual carriers", which are suppressed carriers useful to put under control RF spectrum profile.

Not being the goal of this thesis to recover synchronization and transmission information, and being the virtual carrier obviously completely aimless for video decoding we need just to cut off the fruitless carriers and send to the de-mapper the right symbol stream. Besides, carriers must be normalized according to normalization factors for data symbols. There is a set of values available for both 16-QAM and 64-QAM provided by the DVB standard, the two options useful for our goals are $\sqrt{10}$ for the 2k mode and $\sqrt{42}$ for the 8k one.

3.7.1 Scattered, continual Pilots and TPS

While the continual pilots and TPS positions, inside the OFDM symbol, are fixed by the standard and set in the block as in Figure [3.10], the scattered pilot carriers are inserted according the following rule. Being k the carriers index, the ones which k belongs to the subset $k = K_{MIN} + 3x(l \bmod 4) + 12p$ with p integer grater or equal to 1 and $k \in [K_{MIN}; K_{MAX}]$ are scattered pilots.

Continual pilot carrier positions (index number k)													
2K mode							8K mode						
0	48	54	87	141	156	192	0	48	54	87	141	156	192
201	255	279	282	333	432	450	201	255	279	282	333	432	450
483	525	531	618	636	714	759	483	525	531	618	636	714	759
765	780	804	873	888	918	939	765	780	804	873	888	918	939
942	969	984	1 050	1 101	1 107	1 110	942	969	984	1 050	1 101	1 107	1 110
1 137	1 140	1 146	1 206	1 269	1 323	1 377	1 137	1 140	1 146	1 206	1 269	1 323	1 377
1 491	1 683	1 704					1 491	1 683	1 704	1 752	1 758	1 791	1 845
							1 860	1 896	1 905	1 959	1 983	1 986	2 037
							2 136	2 154	2 187	2 229	2 235	2 322	2 340
							2 418	2 463	2 469	2 484	2 508	2 577	2 592
							2 622	2 643	2 646	2 673	2 688	2 754	2 805
							2 811	2 814	2 841	2 844	2 850	2 910	2 973
							3 027	3 081	3 195	3 387	3 408	3 456	3 462
							3 495	3 549	3 564	3 600	3 609	3 663	3 687
							3 690	3 741	3 840	3 858	3 891	3 933	3 939
							4 026	4 044	4 122	4 167	4 173	4 188	4 212
							4 281	4 296	4 326	4 347	4 350	4 377	4 392
							4 458	4 509	4 515	4 518	4 545	4 548	4 554
							4 614	4 677	4 731	4 785	4 899	5 091	5 112
							5 160	5 166	5 199	5 253	5 268	5 304	5 313
							5 367	5 391	5 394	5 445	5 544	5 562	5 595
							5 637	5 643	5 730	5 748	5 826	5 871	5 877
							5 892	5 916	5 985	6 000	6 030	6 051	6 054
							6 081	6 096	6 162	6 213	6 219	6 222	6 249
							6 252	6 258	6 318	6 381	6 435	6 489	6 603
							6 795	6 816					

Figure 3.10 : Continual pilot carriers position

3.8 Fast Fourier Transform

The last implemented block in my receiver is the FFT. GNU radio framework includes in its toolbox a FFT block which implements the Cooley-Tukey algorithm, one of the most common and fast way to speed up the DFT based on the divide-and-conquer approach.

Being optimally written and resulting extremely fast (or at least really weightless in the chain time economy) it fitted and worked perfectly.

Chapter 4

Optimization

4.1 Real Time Horizon

Talking about software defined radio cannot be done without having in mind to make all the effort fruit into working real time system, receiving real electromagnetic waves and demodulating real signals. Even really rough implementation is generally much more difficult than just talking. To achieve real time performances we need to force the whole system to work faster than the video stream, this means, in other words, that in order to demodulate a 10 minutes video we want the whole system to need less than 10 minutes, at least in its average. Besides it is correct to state that the system built at this point still lacks the synchronization and channel estimation parts that may realistically be quite onerous for the system. As a consequence it is possible to foresee that, to be abreast with real time goal, the system made so far should do its work in something less than the video stream time.

So, how far are we from the real time horizon?

It results that for demodulating, from the FFT to the MPEG-2 decoder, a 20 seconds video the system need almost 3' and 18'', which means nearly 10 factor between the decoding time and the video time.

4.2 Viterbi Computational Problems

As it has already been suggested, the system has a great bottleneck: the Vieterbi algorithm. In spite of a really fast Reed Solomon decoder the Viterbi still lacks speed, in fact while the former take almost the 5% of all the time the latter approaches the 90% of time consuming. It is a natural consequence to wonder how it is possible that two error correcting algorithms are so different in time consumption. The answer relies in two main differences.

First of all, as already said, the Reed Solomon decoder has got as its first task a Syndrome calculation and evaluation. If these are equal to zero, the decoder knows it is in presence of a correct codeword and thus it just tears apart the parity bytes, with a very low computational cost. Viterbi, on the other side, completely lack this "correctness-check" function and must accomplish its work on every triple of bits, be them correct or wrong. Moreover, Reed Solomon accomplishes its whole work every 188*8=1504 bits, while the Viterbi does everything every 2 bits, thus the operations-per-decoded-bit rate difference is clearly prominent.

In order to make it faster it is important, anyway, to focus on which part of the Viterbi consumes the most time. Tests done by stopping (achieved by commenting in the C/C++ code) some of the Viterbi inner part showed that the main performance absorber is the add compare select (ACS), which for each state, every three bits, evaluate the best path and stores the survivors.

4.3 Possible Patching

Of course we cannot let this problem stop us in our path through the real time, so we come to think some of possible patches that can be applied to make the implementation run faster.

As always, for performances related matters, the easiest way is to wait for the industry to provide the market with better machines with faster chip-sets so that they may be able to run the whole program real time. It has to be underlined that, due to the 64 bits nature of the path-registers, a 64 bits machine with a 64 bits Operating System has been able to almost halve the overall CPU time required for decoding compared to of a 32 bit one, actually we went from 5' and 57'' to the 3' and 18''. Luckily, other paths do exist.

In fact, state of art common user machine are supplied with multi CPUs. The obvious consequence is that being able to "split" the algorithm in several different threads will result in a relevant performance boost without having to wait for hardware to improve. Thanks to its intrinsic nature Viterbi is an algorithm that could be deserialized with no real effort from the developer. Indeed it just needs to set a certain number of states for each thread (for example 16 states for 4 thread) to possibly speed up the system. Surely, even splitting the work in four pats would not realistically give us a four factor gain in speed because of the obvious overhead yielded by multi-threading, it is anyway not very drastic to assume to be able to lower from 10 to 3 the ratio between decoding and video time.

With regard to the implementation the nowadays GNU-radio framework does not support thread parallelization inside single blocks! It is true, however, that its latest release allows an inter-block deserialized scheduler called TPB (Thread Per Block) against the common STS (Single Thread Scheduler). But while this is a good option (although still to be really tested) when the problem is an ensemble of uniform computational weight blocks, it is of really no use when you must deal with a single great bottleneck while all the rest is running at the speed of light.

Another possible way is, of course, trying to replace the common full Viterbi algorithm with something faster but not too worse in error protection. As an example a “reduced-state” Viterbi may offer a great speed boost. In fact it is really plethoric to show the strong correlation (nearly proportional) between the number of states taken in consideration as next eligible state and computational cost, thus is crystal clear that half the states will (almost) half the time.

It is also true that lessening the states' number will realistically comport a negative coding gain, or alternatively a minor error correction ability of the decoder, but the are sufficiently strong clues that it would not be a great one. In fact it has been shown [4] that going from 16 to 8 state would just result in a 0.3 dB loss. Moreover it is quite typical of this kind of strategy to perform their best, in our case it means not working too worse compared to the full Viterbi, with reasonably good SNR, and we are acknowledged that the whole DVB-T system works only in presence of good SNR. Consequentially thinking to a reduced state Viterbi as something that will not work too worse than a full one is not a too limiting hypothesis.

Additionally it should be stressed that having ready a bench software receiver is allowing us to implement quite easy and, hopefully, not too computationally heavy soft-Viterbi with its 2 dB of power gain , which might result in a possible re-gaining the loss yielded by the reduced states.

Functional block	Implemented GNU-Radio block name	Percentage of time consuming
Descrambler	rdvb.descrambler_bb	1.1%
Reed-Solomon Decoder	rdvb.rs_bb	3.2%
Outer Interleaver	rdvb.deinterleaver_bb	1.5%
Viterbi algorithm	rdvb.punctviterbi_bb	90.8%
Inner Interleaver	rdvb.innerinterleaver_bb	1%
Demapper 16-QAM	rdvb.demapper16qam_cb	1.3%
Remove not informative carriers	rdvb.removevirtual_cc	0.5%
FFT	gr.fft_vcc	0.5%

Chapter 5

Implementation Results

5.1 Demodulator Validation Test

Whenever a block had been implemented it had to be validated on a real input to be sure it would do his job correctly, and once everything had gone the right way, it had to be tested within the whole demodulation block sequence. In this view it has been fundamental to have signal “dumps” to perform these tests. Dumps are just files built from the modulation chain in Soft-DVB truncated at right position, those, once elaborated through the dual demodulation block (or blocks) have been byte checked with the original generator files.

As a newbie I thought that the chain concatenation tests would have been somewhat redundant once the single block were working perfectly as single, but it turned out I was definitely wrong. Synchronization of all functions is critical to the behaviour of the entire systems: each and every block must understand the precise moment to start working, and it is never stressed enough that failing in this task always results into a complete failure of the demodulation (at least until the system manages to re-synchronize its blocks!).

A demodulator as the one done, with two error correction block, oriented toward a very strong video and audio quality assurance cannot be considered tested without checking its error correction abilities. In order to see if the implemented Reed-Solomon and Viterbi decoders were able to fulfil their duties an ensemble of error corrupted files have been used. Those corrupted files were done off-line with a self made dedicated C++ application able to corrupt bits with a user defined error probability (and as a major checkpoint with a post evaluated BER).

The “perfect” file's bits, then, were corrupted, and the noisy files given to the chain to demodulate. The results of these tests are reported in following table.

BER1	**BER2**	**BER3**
5,00E-003	0	0
1,00E-002	7,00E-006	0
2,50E-002	2,80E-006	0
3,00E-002	4,80E-004	9,80E-005
3,50E-002	1,00E-003	6,11E-004

Legenda:

BER1 = BER at the input of the Viterbi decoder

BER2 = BER at the output of the Viterbi and thus at the input of the Reed-Solomon.

BER3 = BER at the output of Reed-Solomon decoding thus at the input of MPEG-2 decoder.

0 = BER is below measurable limits.

To validate these results, we can compare the performance of the Viterbi implemented in my software decoder with an hardware one. As we can see from Fig [5.1] (obtained from the Institute of Radio Electronics FECT from the Brno University of Technology [5]) they are almost superposable.

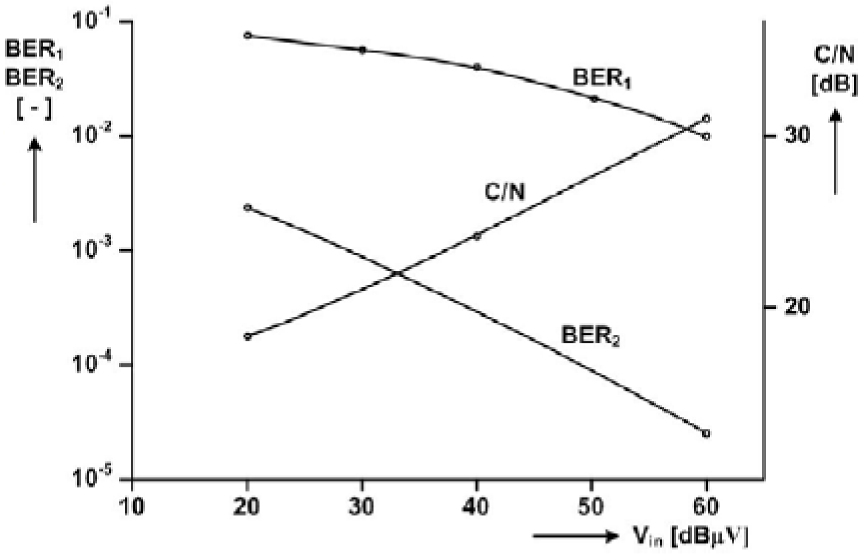


Fig. 5.1 : Hardware Viterbi Decoder perfomance [5]
Legenda:
BER1 = BER before Viterbi decoder
BER2 = BER after Viterbi decoder
C/N = Carrier to Noise ratio

Moreover, the Viterbi output BER required for having a QEF streams at the output of Reed Solomon is, according to the ETSI standard 2e-4, which is completely in accordance with the receiver error correction capability.

In order to test the Reed Solomon correction efficiency, a random number (between 1 and 8 for each 204 bytes block) of bytes have been corrupted in a random way. The decoder has been always able to correct the artificial errors, besides, when the required proper BER is at its input port, it is perfectly capable to perform in accordance to DVB-T standard requirements.

Validation of other implemented blocks has been quite easy. Because of their nature their functionality was of the "on-off" kind thus the only real test was connecting the entire revelation system and make tests on the output file. Since there was no difference (in a byte to byte comparison) between the test and the demodulated file when an error-free coded video was put at the input of the FFT, since the BER calculation with the BER-o-METER at the output of the Descrambling function was congruent with the one outside of the Reed--Solomon and, lastly, since the video was perfectly playable by a MPEG-2 video player those blocks have been considered validated.

Chapter 6

Conclusions

In this master thesis work a prototype, fully-software, low-cost, DVB-T receiver has been implemented. It was developed using open source GNU-Radio framework under GPL (General Public License) license. In order to be able to turn any common desktop or laptop computer into a DVB-T compliant receiver there is still some work to be done: lowering the computational cost in order to achieve real time performance and complete the demodulators chain.

This receiver wants to be a starting point to take all the DVB-T transmission systems form hardware to software implementation, following the trend of developing new multimedia distribution systems based on software defined radios. It also aims at exploring new strategies in developing demodulator functional blocks thanks to the peculiarity of its software nature.

For what concerns the feasibility of the whole project although the time consumption is quite far from what we would like it to be,it is not discouraging and it is possible to consider it as a proof of feasibility, as well as the to forsee that it will be done with just some more work and effort.

It is never stressed enough how much software defined radios bears unprecedented opportunities for service providers, developers and lastly users, both on small and nation wide networks. While service providers would be experiencing cost reduction by substituting hardware components with software ones, developers could create and deploy new technologies faster and wider, while the end user would benefit from always up to date systems without the need to constantly change hardware and habits.

List of Figures

List of acronyms

ADC	Analogue to Digital Conversion
ATSCT	Advanced Television Systems Committee
BER	Bit Error Rate
BM	Berlekamp Massey
CN	Carrier to Noise Ratio
COFDM	Coded OFDM
CPU	Central Processing Unit
DAC	Digital to Analogue Conversion
DFT	Discrete Fourier Transform
DSP	Digital Signal Processing
ETSI	European Telecommunication Standards Institute
FIFO	First In First Out
FFT	Fast Fourier Tansform
GF	Galois Field
GNU	GNU's Not Unix
GPL	General Public License
HW	Hardware
LoS	Line of Sight
MAED	Multiplex Adaptation for Energy Dispersal
MPEG	Motion Picture Expert Group
NLoS	Not Line of Sight
OFDM	Orthogonal Frequency Division Multiplexing
QAM	Quadrature Amplitude Modulation
QEF	Quasi Error Free
RF	Radio Freqeuncy
RS	Reed Solomon
SDR	Software Defined Radio
SFN	Single Frequency Network
SNR	Signal to Noise Ratio

STB	Set Top Box
STS	Single Thread Scheduler
SW	Software
TPB	Thread Per Block
USRP	Universal Software Radio Peripheral
VA	Viterbi algorithm

Bibliography and notes

[1] http://www.etsi.org/WebSite/AboutETSI/AboutEtsi.aspx

[2] http://en.wikipedia.org/wiki/DVB-T

[3] Soft-DVB: http://etd.adm.unipi.it/theses/available/etd-08222008-002954/unrestricted/SoftDVB.pdf

[4] http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00511109
"It is obvious that the eight-state decoder is a good tradeoff because the performance loss compared to the 16 state Viterbi decoder is only 0.3dB and the number of processors in the Viterbi decoder is reduced to eight"

[5] http://www.urel.feec.vutbr.cz/ra2008/archive/ra2006/abstracts/025.pdf

Prof. Marco Luise: "introduzione all'OFDM"

Digital Video Broadcasting (DVB); DVB specication for data broadcasting, ETSI Std. EN 301 192 V1.4.1, June 2004.

Digital Video Broadcasting (DVB); Framing Structure, channel coding and modulation for digital terrestrial television, ETSI Std. EN 300 744 V1.5.1, Nov. 2004.

GNURadio website. Available at: http://gnuradio.org/trac

Bernard Sklar Reed Solomon paper:
http://www.informit.com/content/images/art_sklar7_reed-solomon/elementLinks/art_sklar7_reed-solomon.pdf