Demodulation and error correction

Romain Clédat

***Abstract*— This paper describes the techniques used in demodulation and error correction. It presents where demodulation and error correction occur in the transmission-reception chain. It will also explain techniques used for forward error correction. A brief note about hardware implementation of these codes will also be made.**

***Index Terms*— Modulation, forward error correction, demodulation, Reed-Solomon, bit error rate**

I. Introduction: Context of demodulation

DEMODULATION and error correction occur at the receiving part of the transmission chain. Its function is to regenerate the original signal transmitted with the most accuracy. To transmit information over a certain medium, the signal is first encoded and sent over a medium, in our case, a fiber; it then has to be decoded and, if necessary, it has to be corrected to best reproduce the original signal.

a) Medium of transmission: We will not interest ourselves with the medium in this paper. Many phenomena occur during transmission of the signal over a fiber but we will not take them into account. We will only be preoccupied with how to get the signal back as close to the original as possible. Therefore this paper will focus on methods that correct the problems that may occur in a transmission line rather than on the causes of the problems and ways to reduce or nullify these causes.

b) Modulation: We will however briefly discuss modulation. This is important to understand demodulation. Only the basics of modulation needed to grasp what demodulation is all about will be presented.

A. From transmitter to receiver

The complete chain from a transmitted signal to the received signal is:

- The electrical signal to transmit will modulate the optical emitter (a LASER, a LED, etc.). This modulation is the subject of I-B. The information contained in the signal will thus be transfered to the optical carriers;
- The modulated optical signal is then transmitted over a length of fiber. Dispersion, loss and other problems can occur in this fiber. These phenomena are not the subject of this paper;
- The receiver must then process the signal it receives and generate an output electrical signal identical to the original transmitted signal. The receiver must thus demodulate the signal and make sure it is correct (error correction). What happens in this receiver will be discussed in more detail throughout this paper.

Romain Clédat is a student at the Georgia Institute of Technology

B. Brief presentation of modulation

Modulation can be defined as

> Altering the characteristics of a carrier wave to convey information. Modulation techniques include amplitude, frequency, phase, plus many other forms of on-off digital coding.[1]

This definition makes it clear that there are many ways to modulate a signal.

1) Modulation for LASERs: Two different types of modulation are commonly used for transmitting over an optical fiber:

- *On-Off key modulation*: With this type of modulation, the LASER (or other light emitting device) is turned on and off to encode zeros and ones. This is the most common type of modulation;
- *Subcarrier modulation*: With this type of modulation, another signal is modulated in the microwave range and then controls the light emitting device. This method is particularly useful when one wants to do multiplexing.

The most common modulation scheme, and the one we will consider here, is on/off keying (OOK) modulation.

a) OOK modulation: With this modulation, the optical signal is turned on or off to encode zeros and ones. It is a binary coding with two values for the transmitted signal: either it is on full and codes a one or it is completely off and codes a zero.

To turn the signal on or off, you may either directly act on the device emitting the light by turning it on or off or you can use an external device to cut the light off from getting into the fiber. The second method is preferred as it introduces less chirp and the transmitted signal is less prone to dispersion.

Different signal formats are also used with this type of modulation. The two main ones are non-return to zero (NRZ) and return to zero (RZ). For the former scheme, the pulse occupies the entire bit slot. In the latter scheme, the pulse is shorter than the bit slot (there is no fixed percentage of occupation of the bit slot and this can vary) [Fig. 1]. The importance of signal format and other aspects of it will be developed in II-D. See also I-B.2 for a brief presentation.

b) Other types of modulations: Since OOK modulation is the most commonly used modulation format, we will not describe the other schemes in detail as this is the not the goal of this paper. Suffice to know that other schemes exist but are not widely implemented. The OOK modulation is currently the most widespread and most used. It works with digital signals very nicely and is thus easy to deal with in electronics. Other types of modulation are being experimented on today and may become predominant in the future however.

2) Important aspects of modulation: While the principle of modulation is simple, the implications of how modulation can

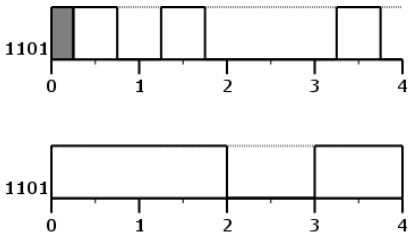


Fig. 1. RZ format (in the top figure) and NRZ format (in the bottom figure) for the data 1101

affect the quality of the final received signal are important to understand. Not all modulated signals are equal. Some signals are better and will be recovered with more accuracy than others. This section briefly presents the main factors to look for in modulation that are important to recovering the signal.

a) DC balance: When deciding whether a received bit is a zero or a one, the receiver compares the power received to a threshold power and decides if it is over the threshold (in that case, the bit is a one) or if it is under (and in that case the bit is a zero). The receiver cannot presume to know what the threshold should be as it will depend on the original power injected in the transmission line and the loss characteristics of the transmission line. It must thus decide, with the first bits that it receives, where this threshold should be. If the received signal has a constant average, that is, if the ones and zeros are statistically as likely, then it will be easy to determine a threshold. It will simply be this constant average power. Statistically, this power will correspond to the mean of the one power and the zero power and will thus optimally differentiate between zeros and ones. This is called a *DC balanced* signal. Different techniques exist to obtain a DC balanced signal. Basically, you must ensure that the signal transmitted has as many ones as it has zeros. You can code the signal or scramble it. These techniques will be discussed later and their impact on demodulation and error correction will also be discussed.

b) Clock synchronization: The receiver is, as most devices used with an optical system, digital, and thus works by taking samples. It will thus take one sample per bit slot for example and determine if the bit is a zero or a one based on the value of this sample. Since the receiver cannot presume to know the frequency of the transmitted signal, it must synchronize on it and thus recover it from the signal. You can either separately transmit the clock to the receiver (but this introduces other problems) or you can try to recover the clock from the transmitted signal by looking at the transitions between zeros and ones. It is thus important to have a signal that does not have long strings of zeros or ones because in that case the clock may be lost. Phased lock loops help recover the clock signal.

c) Other considerations: Other aspects are also important to consider when modulating a signal (such as how much spectrum does the modulated signal take, etc.) but they do not come into play when considering demodulation. We will thus not discuss them. The two main points above are what is most interesting to us when considering demodulation and error correction.

C. The importance of demodulation

As the previous section shows, data transmitted over a channel is modulated in a specific way and undergoes other transformations to make it more adapted to transmission. On the receiving end, it must be possible to undo these modifications and generate the original data; that is the goal of demodulation. It will take the received signal from the receiver and try to:

- Identify the zeros and ones. To do that, it will of course have to correctly determine the bit slots and other parameters such as the threshold value;
- Undo the transformations the signal underwent to be transmitted (de-scramble or decode);
- Apply an optional error correction scheme to try and reduce the bit error rate (BER) as much as possible.

II. Demodulation techniques and difficulties

A. Principles of demodulation

Figure 2 shows the basic principle of demodulation. Different steps are needed to demodulate an incoming optical signal:

1) *Convert the optical signal* into an electrical current with a photodetector. Demodulation is going to be done electronically and thus we need an electrical signal as input. This conversion is done through a photodetector;
2) *Amplify the electrical signal*. The optical signal received and thus the electrical signal produced is usually very weak as it has been through a transmission line and lost a lot of power. The photodetector also introduced loss. To correctly work on the signal, it needs to be amplified;
3) *Filter the signal*. High frequency components can be removed for example as they usually correspond to noise. This filter tries to reduce the various noises produced throughout the transmission chain to have the cleanest possible signal;
4) *Recover the clock rate*: The arriving signal is an analog signal but it needs to be sampled to determine if it represents a one or a zero. However, sampling needs to be done at the right time and at correct intervals. The recovery mechanism recovers the clock rate from the signal and tries to correctly synchronize the sampler to take samples in the middle of the bit slot for example;
5) *Sample the signal*. Here, samples are taken to determine whether the measured signal represents a zero or a one;
6) *Decide*. The receiver then decides whether the sampled signal is a zero or a one. The principle used is usually that of a comparator. A threshold value is determined and the comparator decides whether the signal value is over or under the threshold value. The determination of the threshold value can be tricky and comparators usually use the first few bits to determine a threshold value;
7) *Correct errors*. The last section of this paper will deal with error correction schemes which reduce the BER dramatically. This stage is optional but very common in today's links, especially long haul submarine links.

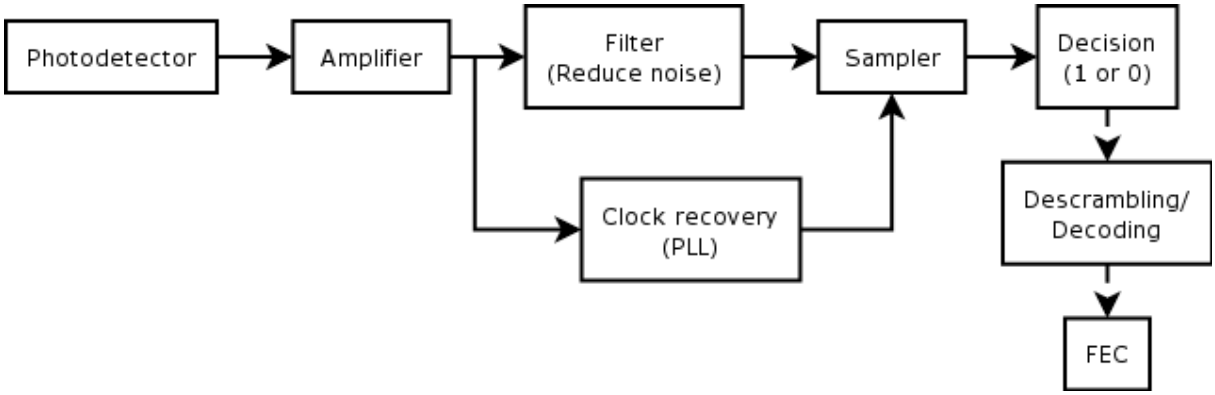


Fig. 2. Block diagram showing the elements of the receiver. The last two elements (de-scrambling and error correction) are optional though they are usually there, especially in long haul systems.

B. Description of a photodetector

Converting the optical signal to an electrical signal will not be discussed in much detail. We can however briefly describe a photodetector and characterize it in the following way:

- *Responsivity*: this parameter describes how the photocurrent coming out of the photodiode varies with the variation in intensity of the incoming optical signal;
- *Dark current*: this parameter describes how much current flows out from the photodiode in the absence of light. This can be considered as noise;
- *Noise equivalent power*: this parameter describes how much optical power is needed to produce the same current as the dark current. Variations of optical power by this amount will thus be undetectable because they will be like noise;
- *Detectivity*: this parameter describes the signal to noise ratio. It is normalized to bandwidth and area of the photodetector;
- *Response speed*: this parameter describes how fast the photocurrent can vary. It thus limits the slope of current change and thus limits, in a sense, the maximum bit rate;
- *Spectral response*: this parameter describes which wavelengths the detector responds to. This depends mainly on the bandgap of the material used.

Refer to [2] for more detail on optical detectors.

C. Obtaining a "clean" signal

The electrical signal obtained from the photodetector is usually not "clean" enough to extract viable information from. This section describes the factors to take into account in order to clean out the signal and obtain something that may be used to correctly interpret and demodulate the signal.

1) Noise: This section will briefly describe the noise present in signals and the different sources of noise. This is not an exhaustive description of noise but is useful in understanding the problems described in II-C.2.

a) Noise comes from different sources: Noise is not a simple phenomenon and comes from different sources. Three main sources of noise are [3]:

- *Thermal noise*: this noise comes from the random motions of the electrons. This depends on temperature. At zero Kelvins, there would be no thermal noise as all particles would be still. This has never been achieved;
- *Shot noise*: this noise comes from the fact that the photons do not come in a continuous fashion. They actually arrive at the detector as quanta. Therefore the electrons generated by the photodetector are also not generated in a continuous fashion even if the optical intensity arriving at the detector is constant. This is due to the quantum nature of the photons;
- *Amplifier noise*: amplifiers also add noise. This is one of the big trade-offs of an amplifier. To get more gain and thus more easily differentiate between a zero and a one, you need to have amplifiers. However, this not only amplifies previous noise present in the signal but it also adds more noise. This noise mainly comes from spontaneous emission.

2) Amplification and filtration: As previously stated, the signal received by the receiver is usually fairly weak and distorted; it thus needs to be amplified and filtered. The amplification's goal is of course to make a one easier to distinguish from a zero. However, the problem with amplification is that it also tends to amplify the noise added by different components of the transmission system. The filters come in play here to try to block out the noise present in the signal. Noise is usually at a high frequency and can thus easily be blocked out by low-pass filters. This of course causes problems as a rectangular bit (representing a one for example) has an infinite spectral content. Indeed Fourier theory states that any finite temporal signal has an infinite frequency content and vice-versa.
Describing amplification and filtration could make a whole paper by itself so we will not go into details about these processes. We will rather concentrate on a clean signal. There are still many characteristics in a clean signal that need to be considered and that are important to demodulation.

D. Signal characteristics

In this section, we will suppose that the amplification and filtration of the signal have achieved their goal and we have obtained a clean signal that is interpretable. We will thus suppose that we have a signal that looks reasonably like the original transmitted signal. We will concentrate now on the

characteristics of the received signal and how it can influence the recovery of the binary data.

1) DC balance: As previously stated, a signal optimally needs to be DC balanced, that is, have as many zeros as ones to have a constant DC (average) value. This makes it easier to determine the power difference between a zero and a one. Different techniques exist to achieve DC balance. Most are based on a table lookup principle. A 'source word' (a specific number of bits) of length m is mapped to a unique word of length $m+p$. p corresponds to the overhead needed to achieve the DC balance. Of course, p has to be as small as possible in order to maximize the code's spatial efficiency. The code also needs to be efficient in the sense that it is easy to code a source word and easy to decode a coded word back to its source word.

The information presented in this section was taken mainly from [4] and [5].

a) Some useful definitions: To understand DC balance, we will start by defining terms useful in comprehending what is going on.

Some useful definitions:

Source word	A source word is a portion of the binary data to be transmitted (or that has been transmitted). It is characterized by its length. It is thus a sequence of zeros and ones.
Codeword	A codeword is an equivalent to a 'source word'. A codeword is usually longer than the source word it corresponds to. The extra length is due to the fact that the codeword tries to achieve a certain property (in our case, having as many zeros as ones)
Disparity	This refers to a word (source or code). It is the difference between the number of zeros in a word and the numbers of ones. A one is counted as '+1' and a zero is counted as '-1'. A word having exactly the same number of ones and zeros thus has *zero disparity*.
Zero-disparity code	Such a code transforms source words to codewords that all have zero disparity.
Low-disparity code	These codes are usually simpler than zero-disparity codes but do not guarantee zero-disparity codewords but only ones that have a low disparity.

We will suppose the following:

- The source words will have length m where m is even. It is easier to consider m even but the algorithms proposed below can usually be generalized to m odd;
- The overhead incurred by the coding will be p. The final codeword will thus be of length $m^{'} = m + p$;

We also define the following terms:

- The source word will be represented by

$$x = (x_1, x_2, x_3, ..., x_m) \tag{1}$$

where

$$x_i = \begin{cases} -1, & \text{if the binary digit is 0} \quad (2) \\ 1, & \text{otherwise} \quad (3) \end{cases}$$

- The disparity of the first k bits of a word x will be calculated by:

$$z_k(x) = \sum_{i=1}^{k} x_i \tag{4}$$

Therefore, the disparity of a source word x is:

$$z(x) = z_m(x) = \sum_{i=1}^{m} x_i \tag{5}$$

We can also easily define the number of balanced words of length m with m even is:

$$M(m) = \begin{pmatrix} m \\ \frac{m}{2} \end{pmatrix} \tag{6}$$

This is easily understandable as it represents the number of ways to place $\frac{m}{2}$ ones among m possible positions.

To use balanced words to transmit information with m bits of information, we clearly need to have more balanced words than the maximum number of possibilities coded by these m bits. Therefore, we need to have:

$$M(m+p) \geq 2^m \tag{7}$$

We can then use Stirling's equivalent to extract useful information about the minimum number of overhead bits p we will need to use to obtain a balanced code. Stirling's equivalent states that:

$$\ln\left(M\left(m+p\right)\right) = m+p-\frac{\ln\left(m+p\right)}{2}-\frac{\ln\left(\frac{\pi}{2}\right)}{2}-o\left(\frac{1}{m+p}\right) \tag{8}$$

b) Trivial DC balance method: A very trivial and spatially inefficient way to achieve DC balance is to transmit the source word w followed immediately by its complement $\overline{w}$. We will thus transmit the codeword $w^{'} = w\overline{w}$. We thus have $p = m$ and the overhead is thus enormous. This does however achieve DC balance over $2m = m+p$ bits. For m sufficiently small compared to the bit rate (so that the balance is achieved in a small interval of time), this will be sufficient to determine the threshold.

This method is of course very inefficient and is not used in real world systems. It is however a good example to understand what DC balance is and how to obtain it.

c) A parallel coding scheme: The principle of this coding is to split the source word in two parts. The complement of one of these parts will be taken. The position of the split will be calculated so that the resulting concatenation of the original half and the other complemented half has a zero disparity.

More specifically, let $w^{(k)}$ be the word w with the its first k bits complemented. Therefore, for example, if $w = 01011010$, we have $w^{(3)} = 10111010$.

It is clear that

$$z\left(w^{(k)}\right) = -z_k(w) + \sum_{i=k+1}^{m} x_i = z(w) - 2z_k(w) \tag{9}$$

If we want $w^{(k)}$ to be balanced, we must have:

$$z(w) = 2z_k(w) \tag{10}$$

We must find a k so that 10 is satisfied. We see, that:

$$z\left(w^{(0)}\right) = z(w) \tag{11}$$

$$z\left(w^{(m)}\right) = -z(w) \tag{12}$$

Given that m is even, $z(w)$ is also even. Every-time k changes by 1, $z\left(w^{(k)}\right)$ changes by 2 and thus also stays even (since it takes an even value for $k = 0$). Since when k goes from 0 to m, $z\left(w^{(k)}\right)$ has to go from a number to its opposite, and since $z\left(w^{(k)}\right)$ varies continuously over $\mathcal{N}/2\mathcal{N}$, according to the intermediate values theorem, we can say that $\exists k$ so that $z\left(w^{(k)}\right) = 0$. Note that this value is not necessarily unique.
We thus have a coded word that is balanced that can represent our source word. We however also need to transmit the value of k. We need to do this in a balanced fashion so that the total disparity of the transmitted word is not affected.
Therefore, for example, if we want to transmit 256 bits, we can assume, that $k < 256$. Therefore, we need to find 256 balanced words to code k. We must thus have 11 parity bits because:

$$M(10) = \begin{pmatrix} 10 \\ 5 \end{pmatrix} = 252 < 256$$

$$M(11) = \begin{pmatrix} 11 \\ 5 \end{pmatrix} = 462 > 256$$

We can thus choose any 256 balanced words (we will name them $u_0 ... u_{255}$ to encode the position of the cut and we will be thus sending the coded word $u_k w^{(k)}$. We thus have an overhead of 11 bits (4.2%).
The decoding process is also quite simple. The value of k is deduced by using a lookup table on the first 11 bits (in our case) and then the original source word can be deduced from the other 256 bits by again complementing the first k bits. It is indeed obvious that $w = \left(w^{(k)}\right)^{(k)}$. Decoding the word is thus not a problem.
This scheme is thus pretty effective as it has a relatively low overhead. However, it is not the best that can be achieved.

d) Towards more complex coding: The idea is that now, instead of balancing the source word by complementing part of it and then coding with a balanced word the position of the cut in the source word, we can code the position in an unbalanced word with a disparity that will be exactly compensated by the disparity in the coded source word.
Therefore, we will also transmit $u_k w^{(k)}$ but this time $z\left(u_k\right) = -z\left(w^{(k)}\right)$. This allows more flexibility.
We will not get into the details of this scheme although the reader can go look at [4] for more on this scheme. With this new scheme, we can reduce the overhead to 8 bits for 256 data bits. This is thus a significant gain but is not the point of this article.

e) Other methods: The methods described above are not the only ones available to achieve DC balance. They are however very efficient techniques. They do require some overhead though. Other techniques which do not necessarily achieve perfect DC balance consist in scrambling the signal. The signal to transmit is mixed (for example with a XOR operation) with another signal chosen to minimize long sequences of zeros or ones. This incurs no overhead and reduces long sequences of zeros or ones but does not guarantee DC balance. It is however a good solution in some cases.

2) Other considerations: DC balance thus allows the receiver to easily determine the threshold value. It also reduces long sequences of zeros or ones and thus allows better clock synchronization.
Of course, it could also be interesting to discuss other aspects of the signal such as its spectral content. For example, the effect of line coding on the spectral content of a signal is discussed in [5]. This however, is out of the scope of this paper. It is nonetheless important to recognize that these considerations exist and are sometimes crucial in designing a line code. It is not only necessary to design an efficient code with little overhead, it is also imperative to consider other factors that will influence the real behavior of the line coding. A code that adds too much spectral content for example will not be a good code as it will take more bandwidth to transmit and increases dispersion and other phenomena that occur during transmission through an optical link.

E. Synchronization

The last element that we must consider when discussing a receiver is synchronization. As previously stated, the received signal is continuous. The receiver, however, needs to extract bits (zeros and ones). Therefore, it needs to sample the signal. Of course, to obtain the best possible signal, it is better to sample the signal in the middle of the bit slots (at least for NRZ format). The receiver thus needs to determine two parameters to correctly sample in the middle of the bit slot:

- Where does a bit slot start;
- How long is a bit slot?

With these two parameters, the receiver will be able to determine when to sample the signal and get the best possible values. This is where the importance of transitions comes into play. The only way the receiver is going to determine where a bit slot starts is if it sees a change of value (that is between a one and a zero). If there is no transition, there is no way, apart from separately transmitting a synchronizing clock, that the receiver will be able to determine what the bit length is. A circuit called as Phased Lock Loop can recover the bit rate. This loop makes use of a VCO (Voltage Controlled Oscillator) and a phase comparator. The VCO will produce a signal oscillating at a frequency proportional to the input voltage to the VCO (well, actually, there is also a constant term). The phase comparator will produce a voltage corresponding to the difference in phase of two signals, in this case, the input signal and the signal outputted from the VCO. This voltage will control the VCO. If the input frequency is within the range of the VCO (because the VCO only operates on a certain range), the signal produced by the VCO will be perfectly synchronized with the input signal. It will even follow the little variations in frequency of the input signal.
Therefore, we can determine the frequency of the input signal

and thus know when to take a sample. A more complete, yet simple, explanation of a PLL can be found in [6].

F. Conclusion

This section briefly introduced the different stages that occur during demodulation. All the stages described above must be done before any error correction is applied. Error correction is an optional step that can further help reduce the BER but it must be able to work on an electrical signal of zeros and ones. The previous steps thus convert the received signal from optical to electrical and decode/de-scramble it. Encoding was presented in this section. It is important to differentiate encoding and error correction. Encoding is merely a way to make a signal more transmission friendly. It is *not* in any way a mechanism for correcting errors. It helps the receiver interpret the signal better and in that sense you could say that it helps diminish the BER but it is totally different from error correction.

III. Error Correction

Error correction, the core of the subject of this paper. This section will start by briefly describing what error correction is and why it is used. The Reed Solomon codes will then be described in detail. Theses codes are commonly used in today's optical links. Finally, we will look at hardware constraints for error correction chips. This is an important aspect to look at as error correcting modules need to be present in every receiver. If they are too complex and/or expensive, they will not be worth it, even if they increase performance a little bit. A compromise between cost and performance must yet again be found. It is thus important to have a global view of the problem.

A. What is error correction?

Error correction seems like a very straightforward term: it is a technique to correct errors. Basically yes, it is. However, saying that an error correction code corrects errors is not enough to describe it and give a correct idea about such a code. You must also ask "how effective is this code?", or "how much more data do I have to transmit to correct errors?", or "how much power does this code take?". This section will introduce the terms necessary to understand error correction codes. Some historical points will also be given to show how error correction has progressed throughout the years.

1) Birth and history of error correction: Error correction techniques rely heavily on abstract algebra and concepts put forth by Ernest Galois (*1811-1832*) among other things. Mr. Galois was unfortunately killed at a very young age in a duel. Very good in math but hot headed and not a good fighter. It is most unfortunate that this promising mathematician was killed so early.

These concepts were thus fairly old and had been around a long time in mathematics when Claude Shannon wrote and published "A Mathematical Theory of Communication" in 1948 where he introduced the concept that information could be transmitted as a sequence of symbols, each symbol occupying a finite amount of time. He introduced the word 'bit' and even talked about correcting errors. Shannon is also known for his theorem which specifies sampling limits (the Shannon theorem). Transmitting data as symbols with a finite temporal occupancy was extremely new at the time. Analog signals were the only way that data was transmitted. Shannon introduced the basis for the digital era and his work is still studied today.

The theory of error correcting codes was researched from that day till the 1980s. The emphasis then shifted to how to practically make these codes work. Today, the theory of codes is largely understood although research still continues and better ways to implement known algorithms are being researched. We will present the theory of error correcting codes as well as practical aspects in implementing them.

2) Important terms and definitions: Firstly, it is very important to note that error correction can only be applied on digital signals. In our case, we will only consider binary signals. However, the Reed-Solomon codes presented later in this paper do not make use of only two symbols. These codes group together groups of bits to form new symbols. Error correction is thus possible on a signal that used a finite number of symbols. We could for example apply error correction on signals that had three basic symbols like -1, 0 and 1. In binary, the two symbols used are zero and one. Error correction is however not possible with analog signals for example. Therefore, before any error correction scheme can be applied, the data source must be converted to binary format if it is not already in that form.

a) Channel encoding: Once the data is translated into a binary word d, we are going to encode this word with our error correction algorithm and form a *codeword* c of length greater than d. The extra *symbols* in c will contain redundant information. Note that a 'symbol' is not necessarily a zero or a one. A symbol may be a sequence of bits for example. We will see an example of a code that uses sequences of bits as symbols. However, you do need to have a finite number of symbols.

b) Modulation and demodulation: After having been encoded, the codeword c is then used to modulate an optical signal that is transmitted over fiber optic. This signal will then be demodulated on the receiving end and produce a received word r. This word probably contains errors due to transmission noise. This is the word that will be used to decode the channel.

c) Channel decoding: r is then used as input to the channel decoder which performs the opposite function than the channel encoder. The decoder will try to output a word $\hat{d}$ that is equal to the original information word d. It will thus try to detect if there are any errors and correct them if it can. If it cannot correct the errors, it will at least report that there is an error and that the data should be sent again.

d) Types of codes: Error correcting codes are typically classified in two categories:

Block codes	These codes separate the input into k-tuples and process each of them sequentially and separately. A n-tuple is produced to encode each separate k-tuple. These codes make use of al-

gebraic properties and are more easy to decode;

Convolutional codes These codes use a shift register and thus do not make use of algebraic properties but rather decoding a convolutional code is probabilistic. This makes these codes more difficult to implement even though there performance may be greater. See V for more detail. These codes were also introduced later (in 1955 by Elias)

The codes we will describe, the Reed-Solomon codes, are block codes and are the most used in today's optical links.

B. Description of the Reed-Solomon codes

The most commonly used codes in today's optical links are the Reed-Solomon codes. They are a special type of BCH (Bose, Ray-Chaudhuri and Hocquenghem) codes which are random-error correcting block codes. BCH codes where introduced in 1960. These codes are very interesting because they can correct an arbitrary number of symbol errors in an easy to implement way. They are thus very interesting to build cheap decoding chips. The section will introduce some of the mathematical principles needed to understand these codes though will not go into details. More information is available basic undergraduate algebra classes on monoids, groups, rings, and fields and polynomial manipulation.
Reed Solomon codes are a subset of BCH codes which are a subset of cyclic codes which are a subset of linear block codes. In the following sections, we will start by briefly presenting linear block codes and then cyclic codes. This is necessary to correctly understand BCH codes and thus Reed Solomon codes.

IV. Reed-Solomon codes

A. Basic concepts of linear block codes

Linear block codes are also referred to as 'group codes' or 'parity-check codes'. As its name implies, a linear block code is a block code. A block code takes k symbols and transforms them into n symbols. This is commonly noted (n, k). We can define a linear block code as follows:

> A block code made up entirely of 2^k code words of block length n is called a linear (n, k) code if and only if its 2^k code words form a k-dimensional subspace of the vector space of all the n-tuples over the binary field $GF(2)$ [7, 35]

In other words, the code space has to be stable for addition and stable when multiplying with an element of the field.

1) Going from a data word to a code word: The above definition might seem abstract and difficult to understand. The following example should dissipate any doubts. Suppose we have a linear binary block code (n, k) and a data word of length k. We wish to form the code word c of length n. We can always write this in the form:

$$\mathbf{d} = (d_0, d_1, d_2, ..., d_{k-1}) \quad (13)$$

$$\mathbf{c} = (c_0, c_1, c_2, ..., c_{n-1}) = \mathbf{d} \cdot \mathbf{G} \quad (14)$$

$$where\ \mathbf{G} = \begin{bmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,n-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,n-1} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{bmatrix} \quad (15)$$

We of course want to send the data in the code word so we will suppose the following format for the code words (this is called the systematic format):

$$\mathbf{c} = (\gamma_0, \gamma_1, ..., \gamma_{n-k-1}, d_0, ..., d_{k-1}) \quad (16)$$

$$where\ \gamma_j = \sum_{i=0}^{k-1} p_{i,j} d_i \quad p_{i,j} = 0\, or\, 1 \quad (17)$$

The γ_j are parity-check bits that calculate redundancy information on the data bits. They are a linear combination of the data bits thus the name 'linear block code'. We thus have the following $\mathbf{G}$ matrix:

$$\mathbf{G} = \left[\mathbf{P}_{k\times(n-k)} | \mathbf{I}_k\right] = \quad (18)$$

$$\begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,n-k-1} & 1 & 0 & 0 & ... & 0 \\ p_{1,0} & p_{1,1} & \cdots & p_{1,n-k-1} & 0 & 1 & 0 & ... & 0 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n-k-1} & 0 & 0 & 0 & ... & 1 \end{bmatrix} \quad (19)$$

The matrix presented in 19 thus allows the conversion from a data word to a code word (with equation 14).

2) Parity-check matrix and Hamming distance: Now that we know how to code a data word, two very important factors when considering a code are important to look at.

a) The parity-check matrix: This matrix is built so that $\mathbf{G} \cdot \mathbf{H}^T = 0$. The $\mathbf{H}$ matrix generates the orthogonal space to all the code words. We can easily show that:

$$\mathbf{H} = \left[\mathbf{I}_{n-k} | \mathbf{P}^T_{(n-k)\times k}\right] \quad (20)$$

This matrix is very useful because we see that all valid codeword must respect the condition $\mathbf{c} \cdot \mathbf{H}^T = 0$. This makes it very easy to check if a code word is valid or not.

b) The Hamming distance: Another important factor is the Hamming distance between two code words. It represents the number of positions where two code words differ. The minimum such distance is denoted d_{min}. This distance is very important because to have a code that corrects t random errors we must imperatively have:

$$d_{min} \geq 2t + 1 \quad (21)$$

This is easily understandable. Indeed, if we take two codewords u and v, we must make sure that a modified version of u with t errors and a modified version of v with t errors as well cannot be identical. If that were the case, we would be incapable of correcting the errors because we would not know whether to decode the received word as u or v. The minimum

distance d_{min} is thus a very important parameter and it must be as big as possible. However, this distance is bounded. Different bounds exist for d_{min}. The easiest to understand is maximum bound given in 1.

Theorem 1 (Maximum bound for d_{min}): For a (n, k) linear block code, the minimum Hamming distance is upper bound as follows: $d_{min} \leq n - k + 1$

Proof: Let $\mathbf{c}$ be a codeword with a minimum weight m. We thus have m non-zero components and since $\mathbf{c} \cdot \mathbf{H} = 0$, we have $\forall j \in [1; n-k]\ \ h_{j,\sigma(1)} + h_{\sigma(2)} + ... + h_{\sigma(m)}$ where $\sigma : j \rightarrow \sigma(j)$ gives the position $\sigma(j)$ of the j non null component of $\mathbf{c}$. Therefore, we have a linear combination of columns that is null. The m columns are thus not independent. You cannot have more than $m - 1$ independent columns.
Since d_{min} is the minimum distance between two words, it can be seen as the minimum sum of two codewords. Indeed, a modulo two sum of two words will give another word in the code (property of codes) and will only be one when both original words are not equal. Therefore, we have $m = d_{min}$. Given that the rank of the $\mathbf{H}$ matrix is at the most $n - k$ as it only has that many lines, we have $n - k \geq d_{min} - 1$ and therefore we show what we wanted. ■

3) Decoding a received word: When the receiver receives a word r that is the result of the transmission of a code word c it will receive the word $r = c + e$ where e is the error due to noise and other line problems. The goal is to find the errors and correct them. To do so, one must first identify the location of the errors. Then if the code is a binary code, it is easy to correct the error as each symbol can take only two values (zero or one). Therefore if a bit has an error and it is a zero, the real value is one.

a) The syndrome: We will only consider binary codes here. The syndrome is calculated by

$$\mathbf{s} = \mathbf{r} \cdot \mathbf{H}^T = \mathbf{e} \cdot \mathbf{H}^T \tag{22}$$

The second part of equation 22 comes from the fact that the code word $\mathbf{c}$ is orthogonal to $\mathbf{H}^T$. The syndrome will thus solely depend on the error bits.

b) Interpreting the syndrome: The syndrome will give $n - k$ linear equations in e_0 through e_{n-1}. These equations will not give a single solution (there are $n-k$ equations but n variables) and we will therefore choose the solution that has the least weight (the smallest number of errors). The following example, taken from [7, 53-54] illustrates this point.
Suppose we have the parity-check matrix for a (6, 3) linear-block code:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \tag{23}$$

We receive the word $\mathbf{r} = (1, 1, 0, 1, 1, 0)$ and we can thus calculate $\mathbf{s} = \mathbf{r} \cdot \mathbf{H}^T = (1, 0, 1)$.
We thus obtain the following equations:

$$e_0 + e_3 + e_4 = 1 \tag{24}$$
$$e_1 + e_4 + e_5 = 0 \tag{25}$$
$$e_2 + e_3 + e_5 = 1 \tag{26}$$

We thus have 8 possibilities (because we need to set arbitrarily three bit to zero or one and then determine the other three bits from the above equations). We will find $\mathbf{e} = (0, 0, 0, 1, 0, 0)$ and thus obtain the codeword $\mathbf{c} = \mathbf{r} + \mathbf{e} = (1, 1, 0, 0, 1, 0)$.

B. Extension to cyclic codes

In the above section, we presented linear block codes. This section briefly focuses the concepts developed above to a particular set of codes called cyclic codes. These are a subcategory of linear block codes and include some new properties that make them easier to decode. These codes also form the basis of BCH codes so it is primordial to understand these codes to be able to understand Reed Solomon codes.

1) Added property: A cyclic code is just like a linear block code except that it has the following extra property:

> A (n, k) linear code C over $GF(q)$ is called a cyclic code if whenever a code word $\mathbf{c} = (c_0, c_1, ..., c_{n-1})$ is in C then $\mathbf{c}^{(1)} = (c_{n-1}, c_0, c_1, ..., c_{n-3}, c_{n-2})$ is also in C. [7, 89]

Recursively, you can of course extend this to any shift of the original codeword.

2) Polynomial representation and polynomial generator: In IV-A, we described the codes as matrices. Although this is possible to do this in this case, it is also convenient to represent a cyclic code through the use of a generator polynomial.
We can first start by representing the codeword as:

$$c = c_0 + c_1 x + c_2 x^2 + ... + c_{n-1} x^{n-1} \tag{27}$$

We also have a generator polynomial for a (n, k) code that satisfies the following conditions:

1) $g(x)$ is of degree $n - k$;
2) $g(x)$ divides $x^n - 1$ in the field considered ($GF(q)$);
3) $g(x)$ divides at least one code word.

The last property given is generalized to the fact that all code words must be multiples of g. Indeed it is easily shown that:

$$\forall i \in N\ x^i c(x) = q(x)(x^n - 1) + c^{(i)}(x) \tag{28}$$

If $x^n - 1 = g(x)h(x)$, it is obvious that if $c(x)$ is a multiple of $g(x)$ so is the cyclically shifted $c^{(i)}$ codeword.
To produce a codeword with a data word, of length k, we would just have to multiply the polynomial data word $d(x)$ with the generator polynomial $g(x)$. This is very easy to implement in a circuit and thus these codes are extremely liked.

a) Systematic code form: Unfortunately, simply multiplying $d(x)$ by $g(x)$ does not define a codeword in systematic form. For example, if we have the generator polynomial $1 + x + x^3$ for a $(7, 4)$ code (we can check that $x^7 - 1 = x^7 + 1 = (1 + x)(1 + x + x^3)(1 + x^2 + x^3)$. One must remember that if we work modulo 2, $-1 = 1$), if we have a data word $d_0 + d_1 x + d_2 x^2 + d_3 x^3$, we will produce the code word $c(x) = d(x)g(x) = d_0 + (d_0 + d_1)x + (d_2 + d_1)x^2 + (d_0 + d_2 + d_3)x^3 + d_3 x^4 + d_2 x^5 + d_3 x^6$ which is not in systematic form.
We want the code to be like $\mathbf{c} = (\gamma_0, \gamma_1, ..., \gamma_{n-k-1}, d_0, d_1, ..., d_{k-1})$. If we also

express the parity bits in the form of a polynomial $\gamma(x) = \gamma_0 + \gamma_1 x + ... + \gamma_{n-k-1}x^{n-k-1}$, we see that we can always express $\gamma(x)$ as the remainder in the Euclidian division of a polynomial of degree n by one of degree $n-k$. This is the fundamental property of the Euclidian division for polynomials. If the dividend has a minimum degree of $n-k$, we can then add the remainder and the dividend to form a polynomial of degree n containing all the information without any overlap of both original polynomials.
This is thus exactly what we do, we choose the dividend to be $x^{n-k}d(x)$ and we choose the divisor to be $g(x)$ in order for the sum of the remainder $\gamma(x)$ and the dividend to be a codeword. We will thus have:

$$c(x) = \gamma(x) + x^{n-k}d(x) = q(x)g(x) \quad (29)$$
$$where \quad x^{n-k}d(x) = q(x)g(x) + \gamma(x) \quad (30)$$

b) Link with the code matrix: The generator polynomial is directly linked to the code matrix. Indeed, the code matrix is obtained by encoding the basis vectors of the data word space. Therefore, for example, for a $(7,4)$ code with $1+x+x^3$ generator polynomial, we would code the four data words $(1,0,0,0);(0,1,0,0);(0,0,1,0);(0,0,0,1)$ with equation 29 and obtain the four rows of $\mathbf{G}$ as:

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (31)$$

3) Encoding and decoding: Why go through all this hassle to form new codes? The answer lies in the fact that they are extremely easy to implement in hardware. Encoding and decoding can make use of the same circuit and that circuit is very easy to build. It basically contains only adders, multipliers and registers.

a) Encoding: Encoding basically consists in calculating $\gamma(x)$. The procedure is shown in Fig. 3(a). The circuit might thus take some space but is simple to build and the components are very inexpensive. A particular case is also shown in Fig. 3(b).

b) Decoding: It is easy to show that the syndrome is just the remainder of the Euclidian division of $r(x)$ by $g(x)$. The circuit used can thus be the same as the one used to calculate $\gamma(x)$. It is thus easy to decode these codes. The syndrome is very easy to calculate and the error correction is also quite simple. Fig. 5 shows the syndrome calculation based from $r(x)$. A complete decoder for $g(x) = 1 + x + x^3$ is also shown in Fig. 4.

C. Reed-Solomon codes

After all these preliminary explanations, we can finally get to Reed-Solomon codes.
Reed-Solomon codes do not operate on one bit symbols by rather on m bit symbols. Therefore, it operates on $GF(2^m)$. A (n,t) RS code thus has:

- $n = 2^m - 1$ symbols in a codeword;
- $k = n - 2t$ information symbols;
- $n - k = 2t$ check symbols.

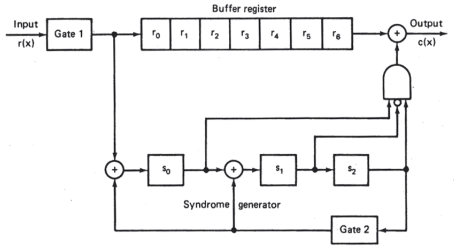


Fig. 4. Decoder for the $(7,4)$ cyclic code with generator polynomial $g(x) = 1 + x + x^3$ [7, 109]

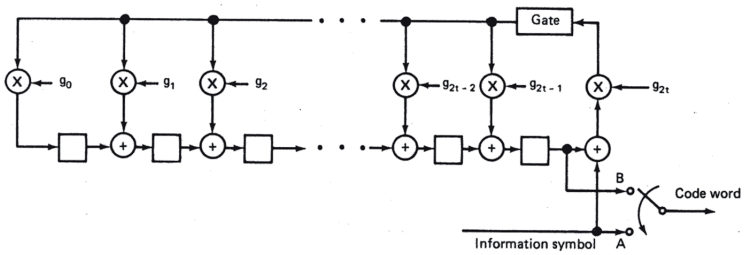


Fig. 6. A (n,t) Reed-Solomon encoder over $GF(2^m)$. This is very like an encoder for a cyclic code. The only difference is the multiplicative coefficients due to the fact that we are no longer in a binary world. [7, 223]

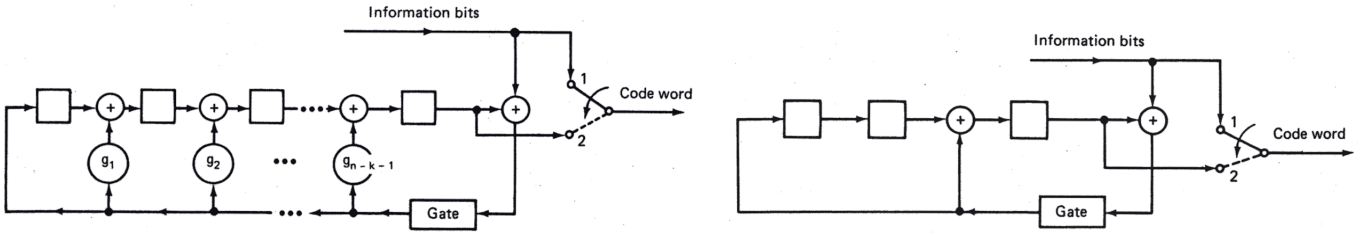
A (n,t) RS code is also expressed as a (n,k) code. You can usually tell from the value of the second number if it is t or k. t is usually much smaller than k. A (n,t) RS code is also a (n,k) cyclic code. All the properties for cyclic codes can thus be applied to RS codes.
The following subsections will not describe in detail the ways to encode and decode Reed-Solomon codes as a whole book could be written just on that subject. The reader is invited to look at the references for a more detailed analysis of this subject. The following sections will merely describe some particular points of the Reed-Solomon codes.

1) Generator polynomials for Reed-Solomon codes: The generator polynomial for a (n,t) RS code over $GF(2^m)$ is given by:

$$g(x) = (x+\alpha)(x+\alpha^2)...(x+\alpha^{2t}) \quad (32)$$

where α is a primitive element of $GF(2^m)$. This is an extra constraint with respect to cyclic codes. This will impose constraints on the coefficients of the code matrix $\mathbf{G}$. Of course, since Reed-Solomon codes are cyclic, encoding a word is fairly straightforward as shown in 6.

2) Decoding Reed-Solomon codewords: What is very interesting in Reed-Solomon codes is that since it works on m bit symbols, an 'error' is considered symbol wise. Therefore, a 3 error correcting Reed-Solomon code can correct from 3 bits (if all errors occur in different symbols) to as many as $3m$ bits in error. This is extremely useful to correct what is known as 'burst errors'. Burst errors can easily be understood: if there is an external phenomenon at time 't' that affects the transmission of a portion of information, there is a great likelihood that all the bits that many bits that are close by will get corrupted. Burst correction is thus very important and RS is a very effective way to do it. This section will briefly present the basic decoding technique for Reed-Solomon codes.



(a) General Case (b) For $g(x) = 1 + x^2 + x^3$

Fig. 3. Encoder for a (n, k) cyclic code. The squares represent registers. They thus introduce the shifting necessary to calculating the partiy bits. We can see that the first parity bit will be ready when k information bits have been fed to the circuit. At that time, the switch will start outputting the parity bits. This will thus correctly form the codeword. [7, 101]

(a) $r(x)$ from the left (b) $r(x)$ from the right

Fig. 5. Syndrome calculation from the received word $r(x)$. The circuit used is the same as for encoding (almost). We see that $r(x)$ can be entered into the circuit in two different places. [7, 106]

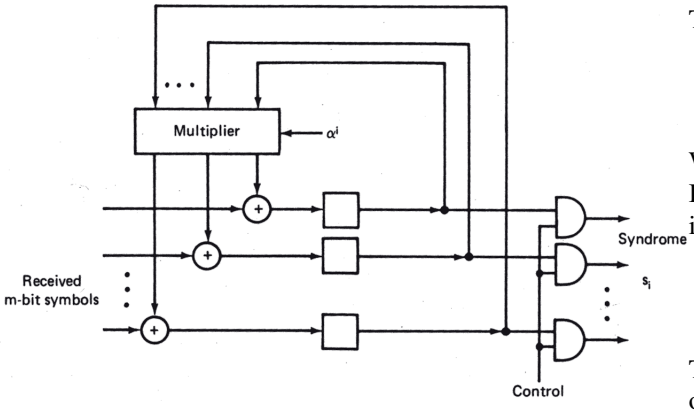


Fig. 7. Syndrome calculation for a RS code over $GF(2^m)$ [7, 232]

a) Syndrome calculation: Again we need to calculate the syndrome of the received word $r(x)$ to solve for the location of the potential errors. A diagram to calculate the syndrome is shown in Fig. 7. Since we have a linear block code, we can reason with the matrix $\mathbf{H}$ (it is easier to understand this way). Since codewords are in the null space of $\mathbf{H}$ (by definition) and since $\forall i \in [1; n-k]\ \ c(\alpha^i) = 0$ (because c is a multiple of g) we can express $\mathbf{H}$ as:

$$\mathbf{H} = \begin{bmatrix} \alpha^0 & \alpha^1 & ... & \alpha^{n-1} \\ \left(\alpha^2\right)^0 & \left(\alpha^2\right)^1 & ... & \left(\alpha^2\right)^{n-1} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ \left(\alpha^{n-k}\right)^0 & \left(\alpha^{n-k}\right)^1 & ... & \left(\alpha^{n-k}\right)^{n-1} \end{bmatrix} \quad (33)$$

Therefore, we immediately see that:

$$\forall l \in [1; n-k]\ \ s_l = \sum_{i=1}^{n} \left(\alpha^i\right)^l \quad (34)$$

We can introduce a new notation to simplify this expression. If we consider we have t errors at position j_i where $1 \leq i \leq t$ in the code, then we know that:

$$\forall l \in [1; n-k]\ \ s_l = \sum_{i=1}^{t} \left(\alpha^{j_i}\right)^l \quad (35)$$

This is true since all the other coefficients of e are null. We obviously have $s(\alpha^i) = e(\alpha^i)$. We thus want to calculate s and solve for the j_i. This will give us the position of the errors. To do so, we will introduce a new polynomial and use Newton's identities to link the coefficient of the polynomial to its roots. We introduce the *error locator polynomial*:

$$\sigma(x) = \sigma_0 + \sigma_1 x + ... + \sigma_t x^t \quad (36)$$
$$\sigma(x) = (1 + \alpha^{j_1} x)(1 + \alpha^{j_2} x)...(1 + \alpha^{j_t} x) \quad (37)$$

Newton's identities will yield the following result:

$$s_1 + \sigma_1 = 0 \quad (38)$$
$$s_2 + \sigma_1 s_1 + 2\sigma_2 = 0 \quad (39)$$
$$... \quad (40)$$
$$s_t + \sigma_1 s_{t-1} + \sigma_2 s_{t-2} + ... + \sigma_{t-1} s_1 + t\sigma_t = 0 \quad (41)$$

The exact calculations can be found in [7, 187-188].
Once we have found the error locator polynomial, we just need to find the roots. The roots of this polynomial will give us the position of the errors. The Berlekamp algorithm provides an iterative method for finding the error locator polynomial as well as its roots.

b) The error value: With binary codes, locating an error was the same as correcting an error as there were only two possible values for each symbol. For Reed-Solomon codes, this is no longer the case, we must thus determine the value of each error. To do so, we introduce yet another polynomial called the *error-evaluator polynomial* defined as:

$$\Omega(x) = s(x)\sigma(x) \tag{42}$$

This will lead to the following result:

$$\forall i \in [1;t] \;\; e_{j_i} = \frac{\Omega\left(\alpha^{-j_i}\right)}{\prod_{l=1,l\neq i}^{t}\left(1+\alpha^{j_l-j_i}\right)} \tag{43}$$

V. Convolutional codes

It would be presumptuous to want to describe convolutional codes in this brief paper. However it is important to point out that they exist and are very effective codes. These codes use as input not only the k bits considered but also the previous outputted data. You thus cannot represent these codes simply with algebra as before. These codes are better represented by trees. This thus makes it complicated to decode them as you would think at first that you need to look at the whole tree to find the decoded sequence. However, algorithms exist that try to decode these codes without looking at the entire tree. These are mostly probabilistic decoding techniques that try to associate the coded word to a decoded word with the highest probability of success. The books given as reference all have sections on convolutional codes.

VI. Hardware implementation of error correcting codes

When considering an error correcting code, it is not only important to make sure that it is efficient mathematically, i.e. that it will reduce the BER significantly, but it is also capital to ensure that the code can be implemented in hardware. Convolutional codes, for example, are more complicated to implement than Reed-Solomon codes. Although they may be more efficient than the Reed-Solomon codes and induce less overhead, they are more expensive to implement and require more power. This is a problem when you know that you must have an error correcting chip on every receiver. Since receivers are often in the open and in uncontrolled spaces, they have to be robust, cheap and use little power.

A. Considerations when designing an error correcting chip

This section briefly describes what should be considered when implementing an algorithm on chip:

- *Design complexity*: How complex will it be to design the chip? When designing a chip, this is a fix cost that must be spread out among all chips built. The higher the design cost, the more chips will be needed to pay for the design;
- *Build complexity and price*: How complex is building this chip? This will of course depend on the design and ultimately influence the final price of the chip;
- *Size of the chip*: How much space does this chip take? A receiver can need to be relatively small. For example, in the wireless world, for cell phones, the decoding chip needs to be fairly small as phones have become smaller and smaller;
- *Power used by the chip*: How much power will the chip need? This will also be directly linked to how much heat it will dissipate. This is an important factor to consider as again, some receivers have limited power available like in cell phones. This is less true for optical networks but the heat dissipated is an important concern. Indeed, these chips will be close to heat sensitive equipment. If they produce too much heat, extra circuitry to cool them and the surrounding components will be needed;
- *Frequency of operation*: This is of course a very important parameter. How fast can the chip operate? It will need to operate at the line rate or a little slower since the signal scrambling or encoding might induce some overhead that will not be seen by the error correcting chip. This rate will however be faster than the user rate as the error correcting code will have introduced some overhead;
- *Scalability and possibility to evolve*: This is an important factor to consider. What will happen if you decide to go from 10 Gbps to 40 Gbps. Will you have to change everything? Also what happens if you now want to correct more errors? Will the chip be able to be reprogrammed or will you have to get a new one?

All the above considerations lead to the ultimate consideration in systems: *cost versus gain*. Error correction allows you to lower your BER. This is good as it allows you to go further and still maintain an acceptable BER. But at what cost? You will have to drive your system faster to maintain a certain line rate (because of the overhead incurred by error correction) and you will need additional circuitry to support error correction.

B. Example encoder for RS $(15, 11)$

Fig. 8 shows the schematics for an encoder. The decoder will also make use only of simple elements like registers and logical gates. Convolutional codes make use of multiplexers and more complicated circuitry and are thus more expensive to implement.
The decoder for the RS code will however be more complicated than the decoder.

VII. Conclusion

Transmission over an optical system, or any other media for that matter, introduces noise, distortion and other artifacts that make recognizing the original transmitted signal difficult. There are different approaches to try to make up for these natural problems. One of them is trying to correct the problems individually and either anticipating them and correcting in advance (for example, you can transmit the signal distorted and hope that the actual line distortion will restore the original signal) or correcting them near the end of the line (for example with dispersion compensating fiber). The other approach, and the one taken by FEC, is to treat all problems as one and say that they introduce errors in the transmission. The goal is then not to try to correct each problem but to recover the correct signal from the erroneous signal.
Of course, to produce the best results, both these approaches

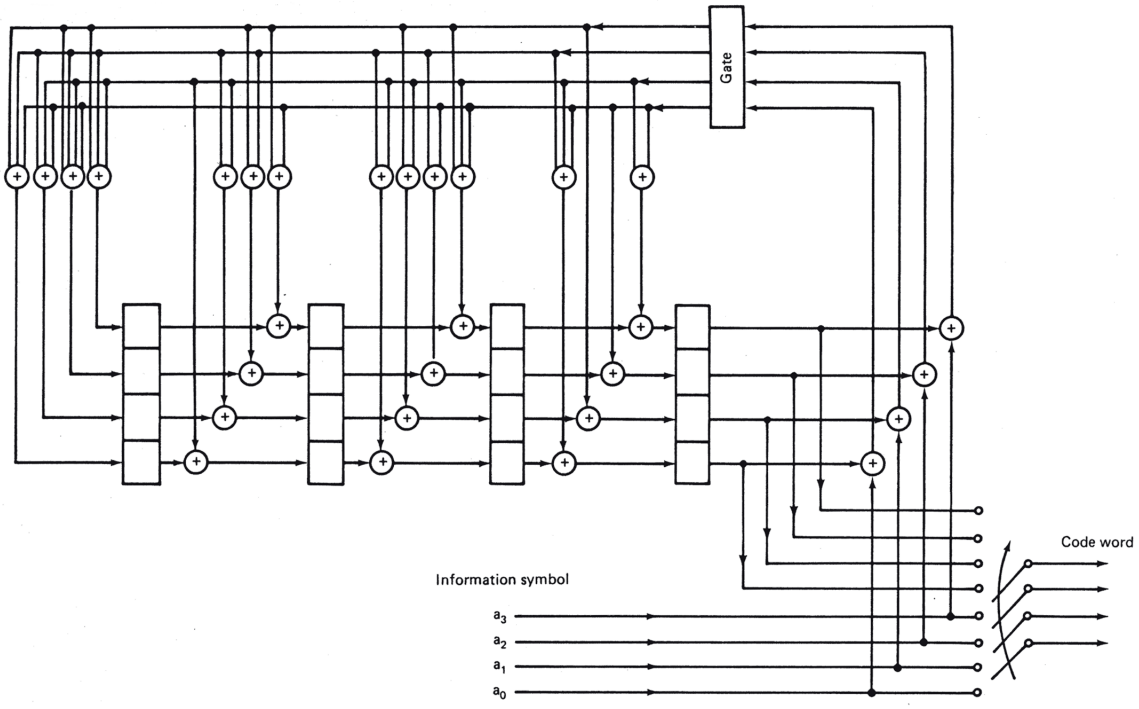


Fig. 8. Double error correcting RS $(15, 11)$ encoder [7, 225]

need to be pursued concurrently. This is what is done in today's modern optical systems. This is a great example of where physics and electronics come together to produce a better result. Physical phenomena can be corrected but electronics can also help with error correction.

It is however important to keep in mind also the physical constraints of a network. Price, space, power are all factors which must be taken into account when implementing an error correction system. Having a theoretical model and an ultra efficient algorithm is not worth anything if it cannot be implemented reasonably.

REFERENCES

[1] [Online]. Available: http://www.networkcables.com/m.htm
[2] R. L. Freeman, *Reference Manual for Telecomunications Engineering*, 2nd ed. John Wiley and sons, 1994.
[3] R. Ramaswami and K. N. Sivarajan, *Optical Networks, A Practical Perspective*, 2nd ed., ser. The Morgan Kaufmann Series in Networking, D. Clark, Ed. Morgan Kaufmann Publishers, 2002.
[4] D. E. Knuth, "Efficient balanced codes," *IEEE Trans. Inform. Theory*, vol. 32, pp. 51–52, Jan. 1986.
[5] H. D. L. Hollmann and K. A. S. Immink, "Performance of efficient balanced codes," *IEEE Trans. Inform. Theory*, vol. 37, pp. 913–918, May 1991.
[6] [Online]. Available: http://en.wikipedia.org/wiki/Phase-locked_loop
[7] M. Y. Rhee, *Error-correcting coding theory*, ser. McGraw-Hill communications, T. Shreve and R. T. Margolies, Eds. McGraw-Hill Publishing Company, 1989.