

Using Bit Error Rate Testers to Test Drive Forward Error Correction Codes

by Tom Waschura, SyntheSys Research, Inc.

Forward Error Correction is a critical component in many modern digital communications applications, turning otherwise unusable communications links into real and practical systems. From DVDs to cell phones, satellite TV to disk drives, error correction technology is a mathematical marvel that effectively makes a silk purse from a sow's ear. Figure 1 shows a simplified diagram of a communications channel with FEC encoding and decoding.

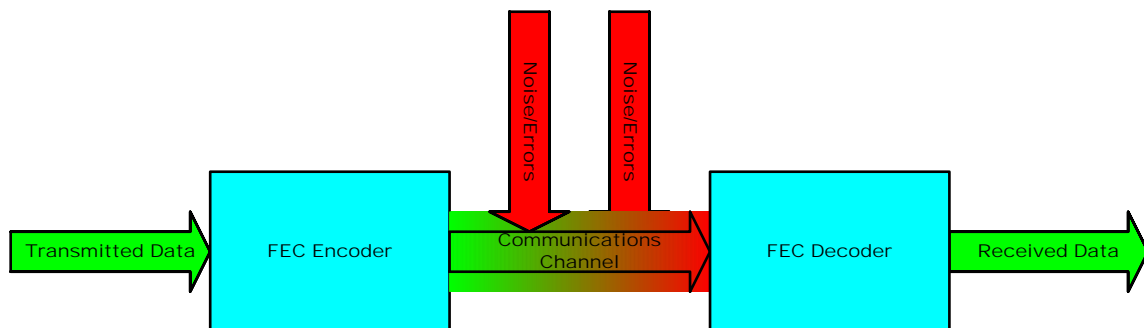


Figure 1 – Communications Channel with FEC

Approaches that correct for bit errors during digital communications vary from simple error-detection mechanisms, to non-real-time correction capability, to real-time on-the-fly error correction. Choices between different approaches depend on the system requirements and statistics of the anticipated errors. Faced with the need to correct occasional random single-bit errors, one might select an error correction strategy that might not be suitable for rare, short multiple-bit bursts of errors. Single long burst-error events might indicate a different approach that requires large amounts of buffering, but which introduces latencies that might be unacceptable in the system application. The trade-offs involved in specifying an efficient and effective error correction require the designer to have knowledge of actual or anticipated system performance, and application requirements.

Before designing an error correction process, it is important to fully understand the types of errors that are typical in the system. The best way to get this information is to collect error statistics during various typical-use cases. In the past, error statistics were limited to simple bit error rate averages that offer little insight when designing error-correction strategies. Bit error rate testers that capture the exact bit location of detected errors provide the precise statistics needed when making these choices. Examples of statistics that help in this determination include the following:

1. Separate measurements of bit and burst error rates
2. Probability distributions of different burst lengths
3. Populations of the number of data blocks by the number of errors they contain
4. The distribution of error-free intervals between errors

Used together with system requirements, these measurements provide the data necessary to make an intelligent design choice.

For example, Hamming Codes made popular in memory arrays are well suited for high-probability, random one-bit errors in short codewords. Maximum Likelihood Codes (of which Viterbi Trellis Detectors are a subset) are used to reduce one-bit errors caused by white noise. Fire codes, used in tape drive and floppy systems, offer efficient and fast correction of rare, single-burst errors that have a relatively short length (e.g., less than 7-15 bits). Product array Reed-Solomon codes used in everything from CD-ROMs to deep-

space communications provide effective correction for potentially long-burst errors at the expense of large buffers and processing latency.

The math behind error corrections codes is based on the concept of adding some information to the transmitted message such that the received message with no error becomes more unique than a received message with error. Generally, we think of a message with added FEC information as a codeword. Sometimes the added FEC information is just appended to the end of the message (e.g., CRC, parity, checksums). In other cases, the added information is convolved with the data to create an entirely new message (e.g., Viterbi, 8/10 codes).

Making a wrong decision of error correction type can mean considerable effort in the complexity of the system since the complexity of an FEC decoder follows the type of strategy chosen. The complexity defines the inherent latency, processing requirements, probabilities of mis-detection/correction and error propagation modes. For example, floppy disk drives can use firmware to correct single sector small-burst errors with only simple hardware CRC error detectors. When a CRC error is detected, the reading process slows down and the software would take over to attempt a small correction using the result of the CRC computation. This can work effectively because errors are rare and the system does not have a real-time requirement. A digital video tape player cannot pause playback to fix an error. In this case, the playback unit must have real-time error correction capability. These decisions must be made based on real-life error statistics.

Identification and recording of the exact bit locations of detected errors in a channel allows proposed error correction strategies to be easily emulated in bit error rate testers. The simplest case is to consider a Reed-Solomon (RS) type block code. RS block codes are the basis behind some of the most popular FEC systems, including satellite broadcast, underwater fiber optics, digital tape recording, and deep-space communications. In these codes, a message of length k symbols has $2T$ symbols of overhead appended to it to make a total message of length n ($n=k+2T$). This code, sometimes referred to as an $RS(n,k)$ code, has the potential to correct T symbols in error no matter where in the message the incorrect symbols occur.

For example, in MPEG2 data used for DVB satellite broadcast, a 30-90 Mbit/sec $RS(204,188)$ code is used, allowing for up to eight byte symbols of correction. Every block of 204 bytes received by the detector is decoded in real time. As long as there are fewer than eight byte errors, the decoder will remove all errors and provide perfect video. If there are more than eight byte errors, the error detector will not be able to remove the errors and image problems will occur.

Analysis functions integrated into bit error rate testers can sort and count the exact bit locations of detected errors according to the user-defined error correction parameters to see how many errors land in individual codewords. For example, for DVB MPEG2 data, errors can be accumulated on 204-byte boundaries. Any time the number of symbol errors inside the block of 204 bytes is less than or equal to eight, the errors can be removed from further error counting/analysis as they would have been removed by an error corrector. This type of analysis means that the corrected error rate can be computed by counting errors only when they occur at a rate greater than eight byte errors per 204-byte packet. Figure 2 shows this type of analysis on one popular BER tester.

CURRENT INTERVAL	Total Accumulation	Processing
Current Interval	Before FEC	After FEC
Error Count	3,227	547
Error Rate	2.15E-06	3.96E-07
Bit Count	1,500,000,576	1,382,353,472
Data Rate	1,500.00 Mbit/s	1,382.35 Mbit/s
	Failures	Corrections
Inner-Code Status (blocks)	17	773
Outer-Code Status (blocks)	185	0
Erasure Status (tables)	0	0

Figure 2 – Before and After FEC Analysis in BER Tester

Symbol size is the first parameter that must be defined before doing this type of analysis inside a bit error rate tester. Symbol sizes of 8-10 bits are often used. For the rest of the analysis, individual bit errors are

ignored in favor of symbol errors. A symbol is considered to be in error if it has one or more bit errors inside the symbol. Symbol error statistics are readily computed when the exact locations of all bit errors in a data stream are known.

Individual codewords in Reed-Solomon block codes can only correct a relatively small number of symbol errors. Increasing this number causes a large increase in the amount of code overhead as well as a large increase in the processing power and time needed to make real-time corrections. When errors tend to occur in small or large bursts, an alternative to increasing the T value of a RS block code exists that will improve the correction capability at the expense of more latency. This can be accomplished by interleaving the data into a memory buffer.

The effect of interleaving is to attempt to split apart burst errors so that their individual symbol errors fall into multiple, different codewords. If a 14-symbol burst error confronts a single RS(204,188) code, it will cause the code to fail; however, if every other byte is split apart and handed to two separate RS(204,188) codes, then the entire error can be corrected using the same T=8 correction logic. The penalty here is that the receiver must wait to get two complete 204 byte codewords before it can start working on the correction. In some systems, this latency is never noticed (e.g., streaming applications such as digital video playback and deep-space satellite receivers). However, in other transactional systems, this severely limits the code's usability (e.g., networking packets).

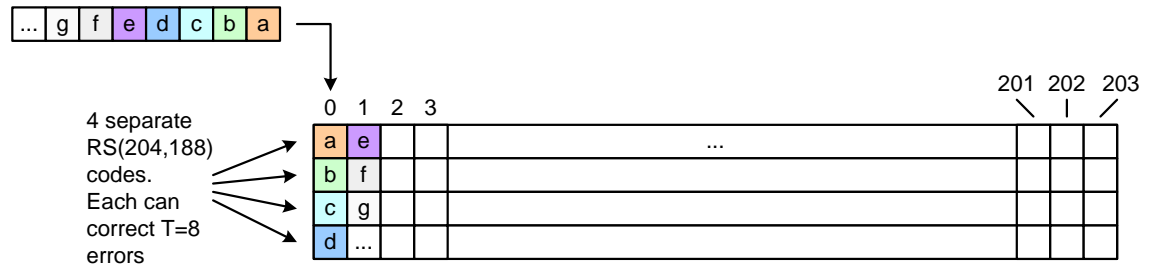


Figure 3 -- Four-Row Interleave with One-Dimensional RS(204,188) Code

Interleaving is a simple sorting function that can also be easily emulated during bit error analysis. Interleaves are typically formed by specifying the number of codewords that are simultaneously filled. For example, an interleave of four RS(204,188) eight-bit symbol codes would have a table of four rows with 204 bytes in each row. This table would represent a total of 6,528 bits. Figure 3 illustrates this type of interleave scenario. When a bit error is detected, the location information is used to determine where in this table the error would have occurred. Once all 6,528 bits of data have been received, the table is examined row-by-row to see if any row has more than eight symbol errors present. Errors in all rows that have eight or fewer symbol errors are removed from the error table before counting. Removing these errors from the table virtually implements the error correction that would occur in this case and the error rate of errors that remain will reveal the post-corrected error performance.

Other examples of this two-dimensional interleave with a one-dimensional correcting code are the ITU standard G.709 and G.975 codes used for fiber optic communications. For example, G.709 calls out 8-bit symbols with at T=8, RS (256,239) code interleaved across 16 rows. G.975 calls out a similar code with only 4 rows of interleave.

Multiple dimensions of block codes can also be used to enable a relatively simple RS block code (e.g., one with a small T value) to correct very large burst errors. This further increases the amount of latency between data reception and decoded data output, since now the full table must be present before corrections can start and two levels of correction must be performed. This is the approach taken in applications such as digital video recorders, where latency is no issue and large bursts are common. In this architecture, once a table is filled with codewords, each individual row is first corrected. After all rows are corrected, individual columns are corrected as well. As long as less than T rows fail, all errors in these remaining T rows will be corrected by the column-corrector. This technique is a good trade-off for correcting both random and burst-error components.

In cases where random errors are not a problem, but the ultimate in burst-length correction is needed, another technique can be employed. In RS coding, 2T symbols are appended to message to accomplish only T corrections. This is because one symbol is needed to find the error and another symbol is needed to correct the error. In cases where the error locations are known, it is not necessary to use a symbol for this

determination. All symbols can be used for error correction, resulting in twice the correction efficiency. For example, when using a two-dimensional product array code, we could use the inner code decoder to find which rows have any errors in them. As long as the number of all rows with any errors in them is less than $2T$, we can mark these rows as the locations of the errors and then use the outer code decoder to blindly make a correction at each of these rows. Depending on how we filled the interleave table, then, this mechanism can double the burst length correction ability of the code. Often this is referred to as using an inner code failure as an erasure for the outer code

All of these block code-based architectures are easily analyzed using a bit error rate tester. Interleave table dimensions and filling/drainage algorithms may be configured to adapt for any of these approaches. Emulating FEC algorithms with a bit error rate tester offers the advantage of using real error data from a digital channel for analysis rather than relying on a hypothetical model for error statistics. Channels may suffer from phenomena that are not included in the model and this can dramatically affect the overall corrector effectiveness.

Next, we'll work through an example. We will start with an uncorrected data channel that has a general background error rate of 2.68×10^{-6} average bit error rate, where errors come from both burst and non-burst components. We know the bursts are random and correlated because we can view the error distributions using various error location analysis techniques. In Figure 3 we can see an error map of a portion of this digital channel. The error map builds a two-dimensional view of the bit error information by breaking the data into segments and laying each segment next to each other in an image. The locations of detected bit errors are highlighted in the view. Separately, errors from bit or burst phenomenon can be colored differently to give better insight into the cause. In Figure 4 we can see that both bit and burst errors are present and that some errors are highly correlated to the segment length chosen (the horizontal "band"). In this case, the segment length was chosen because it was the natural data packet size for the system.

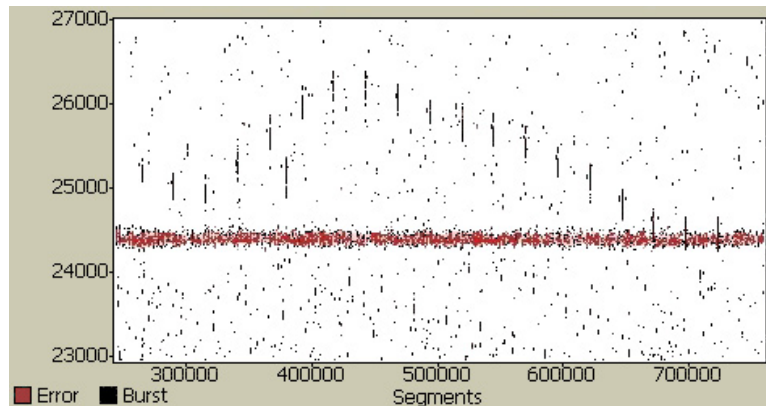


Figure 4 – Uncorrected Error Map, BER = 2.68×10^{-6}

Before we go much further, we should address another point about burst errors. A probability distribution of burst lengths alone is not enough to choose the required correction strength of an FEC code. Often burst errors are found in the presence of other background errors. Additionally, burst errors can be highly correlated to themselves, where one burst will predict the future presence of another burst. In these cases, a single FEC codeword may be presented with more than just a single burst error. Block error statistics, probabilities of error free intervals and error auto-correlations can be used to get better insight into the rest of the error story. Ultimately, though, actual FEC emulation based on bit error position is the most precise way to study FEC effectiveness before building hardware.

To design an error corrector for this data we'll first use a simple eight-bit symbol, one-dimensional block code, RS(204,196), which is a $T=4$ corrector. This code will remove symbol errors when their rate is less than four byte errors per block of 204 bytes. To perform this analysis inside a bit error rate tester equipped with FEC emulation, the user must enable this type of error corrector and set up the parameters:

Symbol Size: 8-bits
 Architecture: One-dimensional
 n : 204
 k : 196
 T : 4

The result of this analysis shows that this simple error correcting code will improve the error rate to 8.55×10^{-7} . To implement this code, a modest latency will be added to the data availability, as the 204 data bytes have to be buffered and then processed for correction (typically, this means a second pass through the data). Additionally, there is an added overhead of 4% required to achieve this improvement. Figure 5 shows the error map of the post-corrected data. Here, as anticipated, we see that the small errors are virtually removed and that larger burst errors continue to persist.

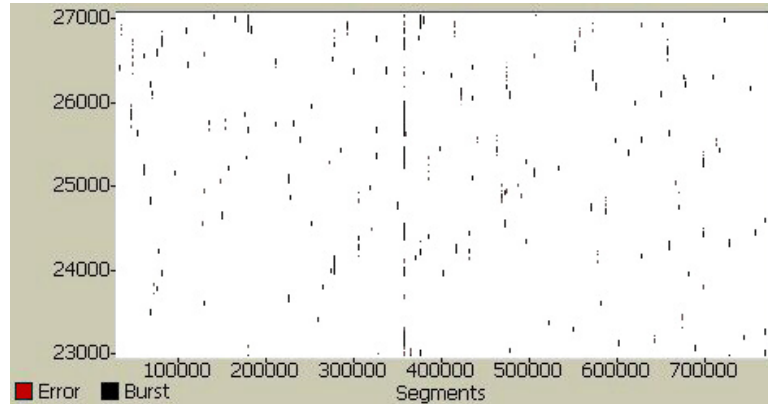


Figure 5 – Error data after one dimensional RS(204,196) corrector

To improve the burst error correction capability, we will next add a simple five-row interleave. Adding this interleave will increase the buffering requirement and latency of data availability, so we will keep this interleave depth to a minimum. In this example, we will change our FEC parameters to:

Symbol Size: 8-bits
 Architecture: Two-dimensional
 inner outer
 n: 204 5
 k: 196 5
 T: 4 0

With this code, we would expect to correct single isolated bursts with lengths up to 20 bytes. The T=0 setting in the second code indicates that there is no correction in the outer code for this analysis. This means that we will interleave the data in two dimensions, but will correct in one dimension. Using these settings, the corrected error rate drops to 2.64×10^{-8} . Figure 6 shows the error map for this interleaved error correction. Note that this enhancement still uses the same basic T=4 decoder technology, but precedes it with an interleave to more evenly distribute errors among a small group of FEC codewords.

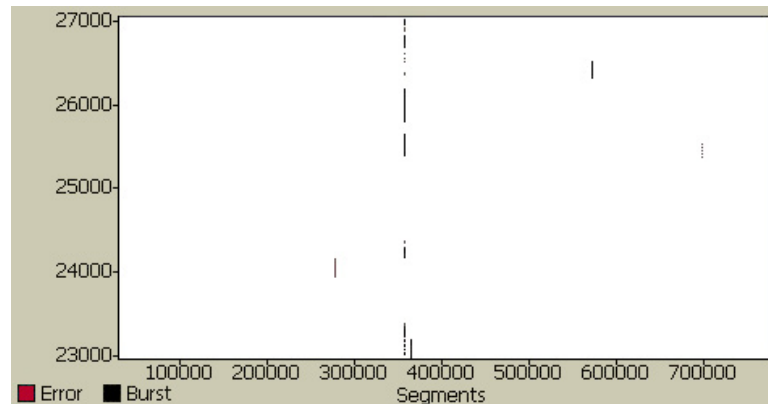


Figure 6 – Error Data after One-Dimensional RS(204,196) Corrector and Five-Row Interleave

It is clear that we could continue this with deeper interleaves and stronger correction strengths to achieve very significant error rate improvements, at the cost of complex correction decoders and long latencies. Using error location statistics in bit error rate testers, system developers can easily explore error correction

strategies to make smart choices about processing requirements, overheads, and latencies to give them usable digital channels for tomorrow's innovative products.

There are a number of simplifications that are taken both in this paper and in the analysis that purist mathematicians will not appreciate. For example, we simplify the RS correction ability to be based only on the location and quantity of errors within a codeword. In truth, there is also a probability that a decoder will mis-detect an error or that it will mis-correct an error. Fortunately, these probabilities are orders of magnitude smaller than the usual case.



Tom Waschura graduated from Stanford University, the Massachusetts Institute of Technology, and Hiram College. He focuses on developing physical layer test technology in areas of bit error rate, eye diagrams and jitter. He has a number of inventions that are used in products available from SyntheSys Research and Agilent Technologies.