Reed Solomon Primer

Reed Solomon (RS) is referred to as a forward error correcting block code in that the data input (Telemetry Transfer Frame shown below) is taken as a block of 8-bit bytes (octets for the old school guys) and corrected on the receiver end.

A typical telemetry frame consists of the frame header, user data and a trailer. We don’t care about the data frame, it’s just cargo. Reed Solomon takes the frame as is and tacks on an ASM (attached synchronization marker) or just sync word. It appears to use the same ASM I used in Turbo. The RS encoder then calculates a group of check symbols that can identify errors in the RS Codeword (think CRCs or checksums). This packet is then serialized and transmitted.

A picture containing timeline

Description automatically generated

The decoder takes the transmitted data serial stream and detects the ASM (just like Turbo) then packs the bits following into bytes thus regenerating the RS Codeword/Check symbols and any errors it picks up. The check bits are then used to determine which bytes of the Codeword are in error and corrects them. The corrected Codeword is then reserialized and exits the receiver. The check symbols are tossed having done their job.

RS codes can only detect so many errors before it gets confused and gives up. If we can spread a burst of errors over several codewords, we might be able to correct more bytes. Interleaving (shuffling) the data over several codewords is the option here. The data comes into an array in left to right, then next row down format just like reading this paper. So now you have an array of 255 x 5 bytes (assuming an interleave ratio of 5). This array is then read top to bottom then next column thus shuffling the deck, so every byte is now 5 bytes away from his original neighbors. These 5 codewords form a codeblock that is decoded by the receiver to correct the errors and stuffed back into another array this time top to bottom then next column and read out left to right and next row. This puts each byte transmitted back into their original position. Interleaving ratios, according to the CCSDS green book, can vary from 1 to 5, one being the default (no interleaving).

Diagram

Description automatically generated

If the RS encoding doesn’t provide adequate margin for errors, then the option of inserting a convolutional (Viterbi) encoder into the serial stream between the RS encoder and the modulator, as shown above. This convolutional code is then received with its errors and Viterbi decoded as a first pass to correct errors. The Viterbi output is then RS decoded. Fortunately for me, the Viterbi process adds then removes any evidence that the transmitted data was manipulated. The Concatenated coding (RS then Viterbi) further improves BER beyond what either process can do on its own.

The diagrams came straight from the CCSDS Green book.

# Further options/customer questions,

CCSDS recommends a 255-byte Codeword of data and check bytes. (The green book refers to the data as symbols rather than bytes because you can have 3 to 12 bits per symbol, but everyone uses 8 bits and bytes are easier to type.)

Options for 223 or 239 data bytes allows 32 or 16 check bytes respectively. Divide the number of check bytes by two gives the number of errored data bytes that can be corrected regardless of the number of wrong bits in that byte. (Actually, it recreates the byte and replaces the transmitted data). These codes are typically denoted as RS(255,223) or RS(255,239) codes for the length of the Codeword and number of data bytes in the Codeword.

“Shortened” codewords, known as “S”, allow for less than the 223 data words to be transmitted. The encoder assumes the missing fill bytes are all zero when calculating the check bytes.

Xilinx allows for Puncture patterns in its core, but I see no mention of Puncturing in the Reed Solomon section of the Green book, only in the Viterbi. So, I’m assuming we don’t need to support puncturing.

The core expects dual-basis data but can be turned off.

QPSK with independent data channels. Do we need RS for each channel?

Do we want statistics on the errors, counts, fails