SCRAMBLING

A *scrambler* (or *randomizer*) is an algorithm that converts a source bit-stream into a pseudo-random one of the same length in order to avoid long sequences of bits of the same value [1]. In telecommunications, this is important for two main reasons:

1. to ease and improve timing recovery on receiver side,
2. to better disperse signal power spectral density and reduce intermodulation interference.

Here the implementation of the two most common scrambling techniques (*additive* and *multiplicative*) in MATLAB (without employing any of its built-in functions) , C/C++ and Python languages is proposed. Both follow an N-cell *linear-feedback shift register* (LFSR) pattern, as depicted in the examples of **Fig.1** and **Fig.2**.

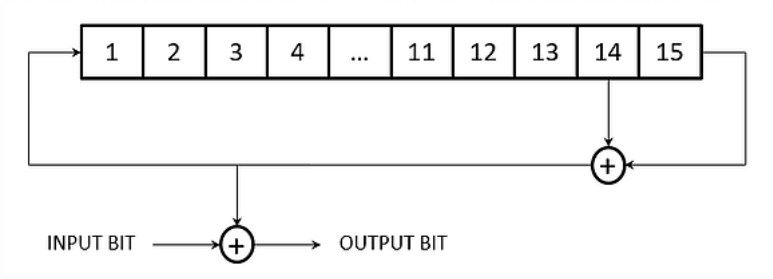

Fig.1 - Additive scrambler / descrambler scheme

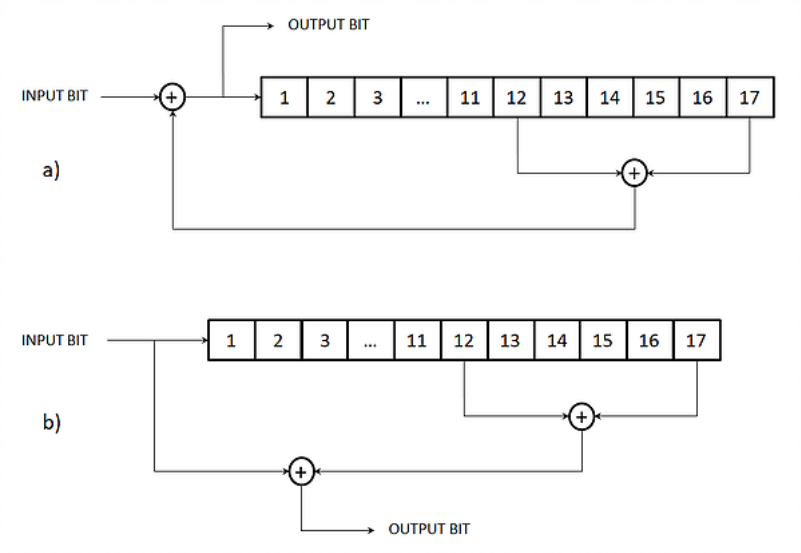

Fig.2 - Multiplicative scrambler (a) and descrambler (b) schemes

The additive (aka *synchronous*) approach foresees the same scheme for both scrambler (on Tx side) and descrambler (on Rx side) and needs for the content of the LFSR cells to be initialized to a known synchronization word each time a new incoming bit-stream begins. The correct operation of the scrambler / descrambler pair is reported in **Fig.3**, where the original info bits are recovered with no errors since both employ the same sync word and no bit corruptions occur between the scrambler output and the descrambler input. The main advantage of the additive scrambling technique is that it avoids error multiplication in case of wrong bits arise at the descrambler input, as can be seen from the MATLAB simulation of **Fig.4**, which makes use of a simple *Binary Symmetric Channel* (BSC) with an error probability of 1%. However, in case there is a mismatch between the Tx and Rx LFSR initial states a massive error propagation results, as shown in **Fig.5**.

```
PARAMETERS:
- Scrambling type : Additive
- Info stream length = 600 bits
- Number of LFSR cells = 15
- Tx/Rx LFSR initial state mismatch = 0 bits
- Probability of error over BSC = 0.000

RESULTS:
```

Fig.3 - Additive scrambler - #1 : Correct operation

```
PARAMETERS:
- Scrambling type : Additive
- Info stream length = 600 bits
- Number of LFSR cells = 15
- Tx/Rx LFSR initial state mismatch = 0 bits
- Probability of error over BSC = 0.010

RESULTS:
- Errors before descrambler = 3 out of 600 bits
- Errors after descrambler = 3 out of 600 bits
```

Fig.4 - Additive scrambler - #2 : Errors at descrambler input

```
PARAMETERS:
- Scrambling type : Additive
- Info stream length = 600 bits
- Number of LFSR cells = 15
- Tx/Rx LFSR initial state mismatch = 1 bits
- Probability of error over BSC = 0.000

RESULTS:
- Errors before descrambler = 0 out of 600 bits
- Errors after descrambler = 224 out of 600 bits
```

Fig.5 - Additive scrambler - #3 : Tx/Rx LFSR initial state mismatch

The multiplicative (aka *self-synchronizing*) approach does not require scrambler and descrambler to be initialized to a common and fixed synchronization word. In fact, after at most N processed bits (needed for self-synchronization) the descrambler is always able to correctly recover the original info stream (an example is reported in **Fig.8**, where the case of different Tx/Rx LFSR initial states is simulated). However, the disadvantage of the multiplicative scrambling is that even a single wrong bit at the descrambler input can cause a significant error propagation at its output, as shown in **Fig.7**. Again, in **Fig.6** the correct operation of the scrambler / descrambler pair is reported as well, where all the original info bits are recovered with no errors since both employ the same LFSR initial state and no bit corruptions occur between the scrambler output and the descrambler input. Therefore, under ideal conditions, the two techniques exhibit the same error-free performance. Moreover, it is important to notice that, unlike the additive approach, the multiplicative one is characterized by different schemes for scrambler and descrambler, as shown in **Fig.2**.

```
PARAMETERS:
- Scrambling type : Multiplicative
- Info stream length = 600 bits
- Number of LFSR cells = 17
- Tx/Rx LFSR initial state mismatch = 0 bits
- Probability of error over BSC = 0.000

RESULTS:
- Errors before descrambler = 0 out of 600 bits
- Errors after descrambler =
    0 out of 17 bits (before sync)
    0 out of 583 bits (after sync)
```

Fig.6 - Multiplicative scrambler - #1 : Correct operation

```
PARAMETERS:
- Scrambling type : Multiplicative
- Info stream length = 600 bits
- Number of LFSR cells = 17
- Tx/Rx LFSR initial state mismatch = 0 bits
- Probability of error over BSC = 0.010

RESULTS:
- Errors before descrambler = 8 out of 600 bits
- Errors after descrambler =
    1 out of 17 bits (before sync)
    23 out of 583 bits (after sync)
```

Fig.7 - Multiplicative scrambler - #2 : Errors at descrambler input

```
PARAMETERS:
- Scrambling type : Multiplicative
- Info stream length = 600 bits
- Number of LFSR cells = 17
- Tx/Rx LFSR initial state mismatch = 13 bits
```

```
RESULTS:
- Errors before descrambler = 0 out of 600 bits
- Errors after descrambler =
    11 out of 17 bits (before sync)
    0 out of 583 bits (after sync)
```

Fig.8 - Multiplicative scrambler - #3 : Tx/Rx LFSR initial state mismatch

All the three implementations (MATLAB, C/C++ and Python) give the possibility to customly set the connection vectors (i.e. the transfer function) and the initial states of the scambler / descrambler LFSRs. However, by default these (represented in polynomial form) are set as follows:

$$H_A(z) = 1 + z^{-14} + z^{-15}$$
$$H_M(z) = 1 + z^{-12} + z^{-17}$$

where additive and multiplicative polynomials are respectively in accordance with DVB and AX.25 standards.

Finally, with the C/C++ implementation it is also shown the scrambling effective capability of the two algorithms. In fact, for instance starting from an all-zero info byte-stream (INF) the results displayed in **Fig.9** and **Fig.10** show how both scramblers perform an almost random shuffle bringing about a satisfying alteration between ones and zeros (SCR), before being recovered by the descrambler (DES).

```
* Scrambling type : Additive
* Number of LFSR cells = 15
* Info stream length = 50 bytes

INF :   0  0  0  0  0  0  0  0  0  0
        0  0  0  0  0  0  0  0  0  0
        0  0  0  0  0  0  0  0  0  0
        0  0  0  0  0  0  0  0  0  0
        0  0  0  0  0  0  0  0  0  0

SCR :   3 F6  8 34 30 B8 A3 93 C9 68
       B7 73 B3 29 AA F5 FE 3C  4 88
       1B 30 5A A1 DF C4 C0 9A 83 5F
        B C2 38 8C 93 2B 6A FB 7E 1B
        4 5A 19 DC 54 C9 FA B4 1F B8

DES :   0  0  0  0  0  0  0  0  0  0
        0  0  0  0  0  0  0  0  0  0
        0  0  0  0  0  0  0  0  0  0
        0  0  0  0  0  0  0  0  0  0
        0  0  0  0  0  0  0  0  0  0
```

Fig.9 - Scrambling capability for the additive technique

```
* Scrambling type : Multiplicative
* Number of LFSR cells = 17
* Info stream length = 50 bytes

INF :   0  0  0  0  0  0  0  0  0  0
        0  0  0  0  0  0  0  0  0  0
        0  0  0  0  0  0  0  0  0  0
        0  0  0  0  0  0  0  0  0  0
        0  0  0  0  0  0  0  0  0  0

SCR :  95 53 1F 98 76 4B 5F 90 56 CD
       47 B2 D8 F4 E3 34 42 DE  C 8F
       CE BB  C ED 48 A2 2E 73 F0  6
       F8 6C FA F9 D2 E1 C7 6C 95 7F
       1D 4E 5A 42 89  9 D4 19 AB 96

DES :   0  0  0  0  0  0  0  0  0  0
        0  0  0  0  0  0  0  0  0  0
        0  0  0  0  0  0  0  0  0  0
        0  0  0  0  0  0  0  0  0  0
        0  0  0  0  0  0  0  0  0  0
```

Fig.10 - Scrambling capability for the multiplicative technique

Below there the links through which the aforementioned codes can be free downloaded. For further details and explanations about the practical implementation of the Reed-Solomon encoder / decoder pair take a look at the comments within these files.

- **Download MATLAB project** : *Scrambling.m*
- **Download C/C++ project** : *Scrambling.cpp*
- **Download Python project** : *Scrambling.py*

WARNING#1: The *Scrambling.m* and *Scrambling.py* projects have been developed in MATLAB R2017a and Python 2.7.12. Therefore, the correct operation of the script may not be assured with different software versions.

References
[1] Wikipedia - *Scrambler (link/a)*

Contacts

Mail
filippo.valmori@gmail.com
LinkedIn
https://www.linkedin.com/in/valmorif