WWW.ESISTREAM.COM

ESIstream

The **E**fficient **S**erial **I**nterface

Protocol Specification

Version 2.0

ESIstream - 2016

ESIstream

The **E**fficient **S**erial **I**nterface

ESIstream is an Open high efficiency serial interface protocol based on 14b/16b encoding.

Its main benefits are low overhead and ease of hardware implementation.

TABLE OF CONTENT

FIGURE TABLE

Issue	Date	Comments
1.0	November 2014	Creation
2.0	October 2016	Precision specified around the boundaries of the protocol

1. SCOPE

1.1 Objective

This specification describes the protocol of an efficient 14b/16b serial interface.
It has been designed for 12 and 14 bits resolution data converters but can be used in any system using serial interface.
This interface provides an efficiency of 87.5% based on a 14b/16b encoding using an LFSR scrambling unit.
It also uses a disparity bit to ensure DC balance transmission and a toggling bit to enable synchronization monitoring.

1.2 Terms and definition

CDR: Clock and Data Recovery
ESIstream: Efficient Serial Interface
LFSR: Linear Feedback Shift Register
PLL: Phase Locked Loop
PRBS: Pseudo Random Binary Sequence
RX: Receiver
TX: Transmitter

2. ESISTREAM PROTOCOL

2.1 Overview

The serial protocol described in this specification can be used in multiple configurations. It is based on a transmission line from the transmitter to the receiver working at high speed. It also needs a SYNC signal from the receiver to the transmitter to synchronize them. Only one occurrence of the SYNC signal is necessary between transmitter and receiver even if multiple serial links are implemented (see example below).

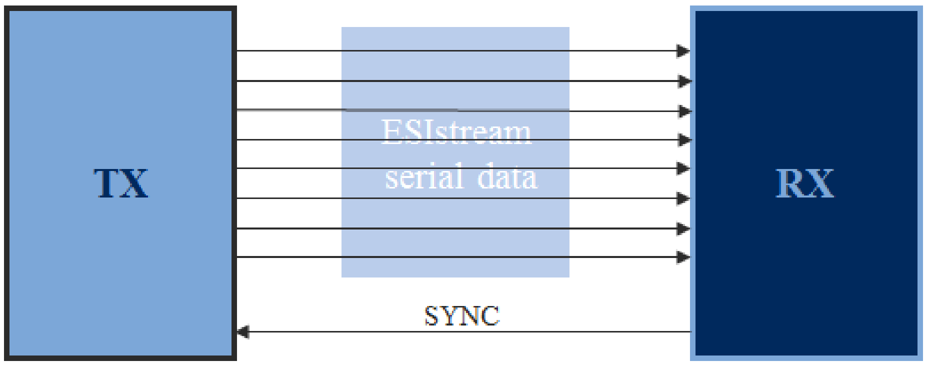

Figure 1: Interface between TX and RX using multiple serial lanes

14 bits of useful data are transmitted, LSB first, between transmitter and receiver encoded in 16 bits.

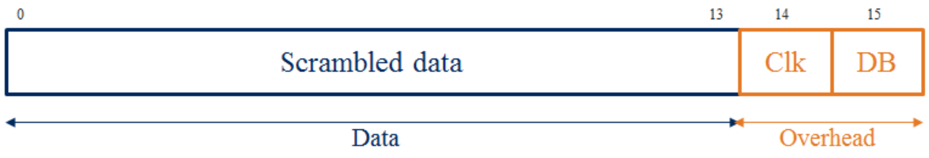

Figure 2: Frame

These 14 bits are first scrambled, and then the overhead is added.
The clk bit is a single bit that is toggling and can be used to monitor the synchronization of the interface. The disparity processing is then done on these 15 bits to obtain the 16 bits frame transmitted.

These different stages of encoding are realized to manage the limitations of a serial interface. First, an AC coupled interface between transmitter and receiver implies that the transmission be DC balance, otherwise the AC coupling capacitor will drift and the received data will be corrupted. Secondly the CDR in the reception stage usually contains a PLL. This means that there must be transitions in the transmission otherwise this PLL will lose its lock.
The scrambling, clk bit and disparity processing ensure a deterministic transmission with a DC balance between +/- 15 and a max run length of the transmission of 32.
Note: Current CDR can work with max run length of 80 and above.

2.2 Scrambling

Applying scrambling ensures a statistical DC balanced transmission. It also statistically ensures that there are transitions in the transmission.

The scrambling technique used in ESIstream is an additive scrambling to avoid error propagation in case of a single bit error. It is based on Fibonacci architecture (see Annex A for more information) using the following polynomial: $X^{17}+X^{3}+1$. It has a run length of $2^{17} - 1$.
Instead of using a shift of one bit per operation, it uses shifts of 14 bits per operation to adapt to the size of the data being scrambled. See Annex B for the equations to use.

The PRBS is applied to the data as follow (LSB to LSB and MSB to MSB):
DATA_SCRAMBLED[13:0] = DATA_USEFUL[13:0] XOR LFSR[13:0]

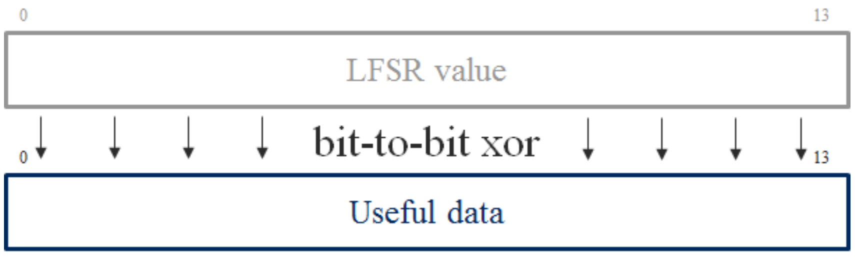

Figure 3: LFSR operation

In case of a multi-lane interface, in order to reduce correlation between lanes, each lane may have different initial values for the scrambling units.

This particular LFSR was chosen because it reduces the number of necessary gates to be implemented.

2.3 Encoding

After scrambling, the 14 bits of data are encoded into a 16 bits frame. One of the added bits is the clk bit; it toggles at every frame.

The last bit encoding these 15 bits is the disparity bit. Its objective is to ensure deterministically the advantages brought statistically by the scrambling process.

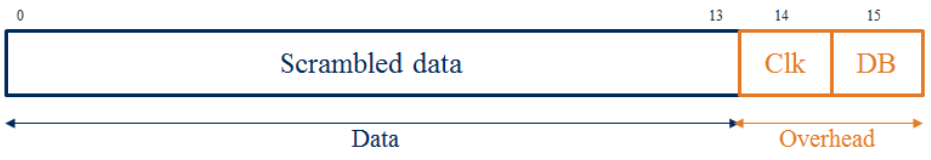

Figure 4: Frame format.

Even with scrambling, large running disparity can still occur with very low probability and could produce excessive eye shifts. These eye shifts could be balanced by a more complicated equalization stage in the receiver if the running disparity was still limited. However, a PRBS does not bind the running disparity deterministically, thus the data could be corrupted on the reception end and it could eventually cause the PLL in the CDR to lose its lock. To prevent this, the disparity bit is implemented.

The running disparity of the transmission is constantly monitored by the transmitter.
For each frame, its disparity is calculated, 2 cases can occur on the running disparity:

a. The running disparity of the transmission **does not** increase above +/- 16 (+16 and -16 included). In this case, the disparity bit is set to ‘0’ and the 15 bits of data (scrambled data + clk bit) are transmitted as such.
b. The running disparity of the transmission **does** increase above +/-16 (+16 and -16 excluded). In this case, the 15 bits of data (scrambled data + clk bit) are inverted and the disparity bit is set to ‘1’.

The running disparity is updated with the disparity of the frame. See Annex C for an example.

This disparity bit ensures that the longest possible series of ’1’ or ‘0’ transmitted is of 48 bits (the clk bit reduces the effective value to 32). It also ensures that the running disparity does not exceed +/- 16 (included) which satisfies the DC balance condition.

In normal operating mode, the receiver will check the disparity bit first. If it is high then it will invert the received data and descramble them. Otherwise it will only descramble them. From this point the data are available for processing.

2.4 Synchronization

The link must be synchronized to align the frames between the transmitter and the receiver and to synchronize the reception scrambler with the transmission scrambler.

The synchronization is controlled through the SYNC signal sent by the receiver to the transmitter.

The synchronization works in 2 steps.

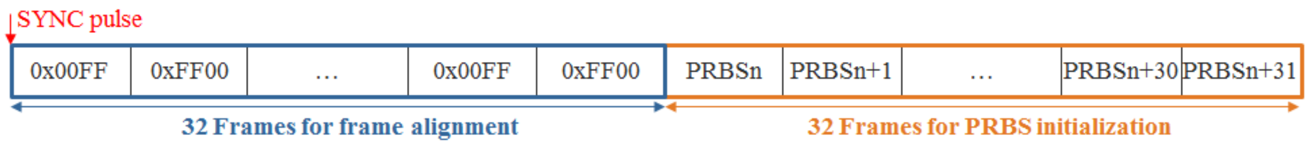

Figure 5: Synchronization sequence

The first step starts when the receiver sends a pulse of SYNC. The length of this pulse should be at least as long as one frame period.
When it is received by the transmitter, it will send an alignment pattern which is 32 frames alternating between 0xFF00 and 0x00FF. The sequence bypasses the scrambling and disparity processing (the sequence is DC balanced). This alignment pattern is used by the receiver to align its data on the transmitter output data.
After these 32 frames, the transmitter starts sending 32 additional frames containing the scrambling PRBS alone. These frames contain 14 bits of the PRBS plus the clk bit and the disparity bit. They go through the disparity processing, as the PRBS value will start to impact the running disparity of the transmission.

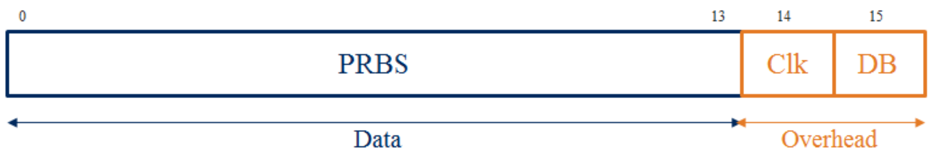

Figure 6: PRBS frames sent for PRBS initialization

The receiver will detect the transition from the alignment pattern to the PRBS alone. The PRBS is reset by the transmitter upon receiving the SYNC to avoid the first frame of the PRBS initialization being either 0x00FF or 0xFF00; this to ensure that passive detection is precise to the frame.
It will determine the initial value it has to start its PRBS with after receiving 2 frames of the PRBS. These 2 frames contain 28 bits of the PRBS sequence; the receiver needs 17 bits to determine its initial value. After that, the synchronization of the link is complete.
The synchronization can be monitored by the receiver using the clk bit. In case the receiver does not detect that the clk bit is toggling properly, then it can state that the link is not synchronous or has lost its synchronization and restart the synchronization process.

3. ANNEX A: PRBS ARCHITECTURE

Fibonacci architecture definition:

A LFSR is used as a generator for a pseudo-random binary sequence to scramble the serial link data.

It is defined by a polynomial: $G(X) = g_m X^m + g_{m-1} X^{m-1} + g_{m-2} X^{m-2} + \ldots + g_2 X^2 + g_1 X + g_0$ and an initial value.

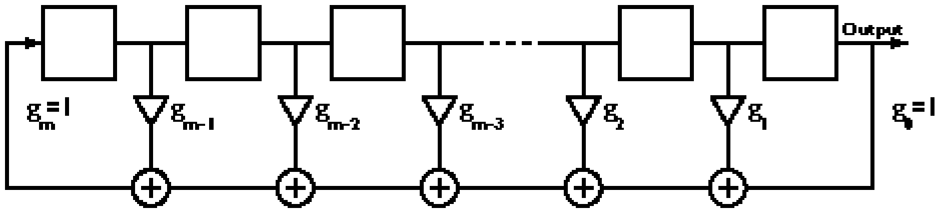

Figure 1: Fibonacci implementation of LFSR

Fibonacci architecture example:

For a 4 bits LFSR of the polynomial X^4+X^1+1:

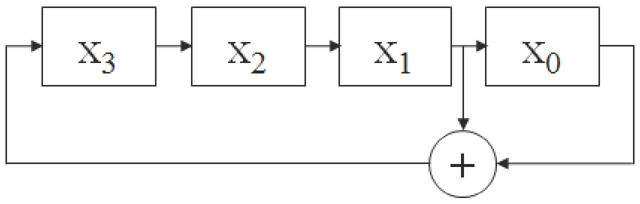

Figure 2: LFSR Fibonacci serial architecture

Fibonacci parallelized architecture:

For a 4 bits LFSR of the polynomial X^4+X^1+1, with a 4 steps parallel architecture:

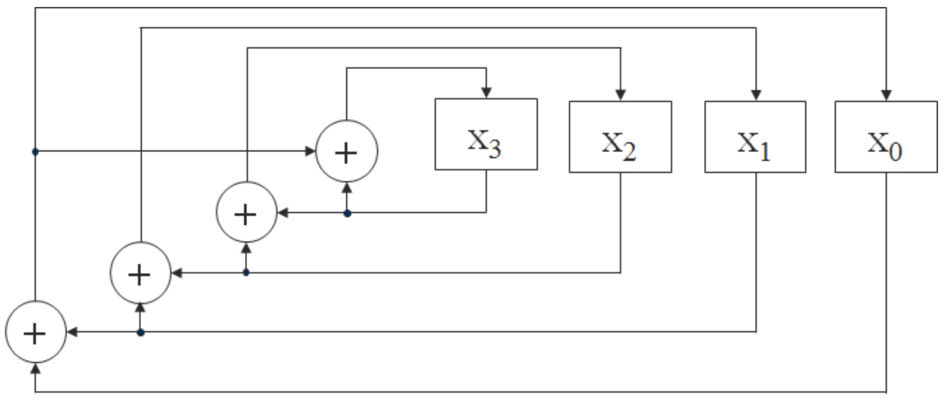

Figure 3: LFSR Fibonacci 4 step parallel architecture

This architecture does in one clock period the same operation that the serial architecture does in 4 clock periods. It keeps the same length.

4. ANNEX B: LFSR EQUATIONS

The LFSR polynomial used is $X^{17}+X^{3}+1$. The LFSR is based on a Fibonacci architecture working with steps of 14 bits shifts. The following equations characterize this LFSR:

$$
\begin{array}{l}
LFSR_{n+1}(0) = LFSR_n(14) \\
LFSR_{n+1}(1) = LFSR_n(15) \\
LFSR_{n+1}(2) = LFSR_n(16) \\
LFSR_{n+1}(3) = LFSR_n(0) \text{ xor } LFSR_n(3) \\
LFSR_{n+1}(4) = LFSR_n(1) \text{ xor } LFSR_n(4) \\
LFSR_{n+1}(5) = LFSR_n(2) \text{ xor } LFSR_n(5) \\
LFSR_{n+1}(6) = LFSR_n(3) \text{ xor } LFSR_n(6) \\
LFSR_{n+1}(7) = LFSR_n(4) \text{ xor } LFSR_n(7) \\
LFSR_{n+1}(8) = LFSR_n(5) \text{ xor } LFSR_n(8) \\
LFSR_{n+1}(9) = LFSR_n(6) \text{ xor } LFSR_n(9) \\
LFSR_{n+1}(10) = LFSR_n(7) \text{ xor } LFSR_n(10) \\
LFSR_{n+1}(11) = LFSR_n(8) \text{ xor } LFSR_n(11) \\
LFSR_{n+1}(12) = LFSR_n(9) \text{ xor } LFSR_n(12) \\
LFSR_{n+1}(13) = LFSR_n(10) \text{ xor } LFSR_n(13) \\
LFSR_{n+1}(14) = LFSR_n(11) \text{ xor } LFSR_n(14) \\
LFSR_{n+1}(15) = LFSR_n(12) \text{ xor } LFSR_n(15) \\
LFSR_{n+1}(16) = LFSR_n(13) \text{ xor } LFSR_n(16)
\end{array}
$$

5. ANNEX C: EXAMPLE OF DISPARITY BIT IMPLEMENTATION

During a transmission, the running disparity (RD) is constantly monitored. At some instant t_i, the running disparity of the transmission is -5. It means that from the beginning of the transmission, there were 5 more ‘0’ sent than ‘1’.
The next 15 bits of data (14 bits scrambled + clk bit) to be sent are below:

$RD(t_i) = -5$

t_i	0	1	0	0	0	0	1	0	0	1	1	0	0	0	0	$D_w = -7$

The disparity of the word (D_w) is -7. The updated running disparity of the transmission is:

$RD(t_{i+1}) = RD(t_i) + D_w + DB = -5 – 7 – 1 = -13$

DB = +1 if the disparity bit is ‘1’
DB = -1 if the disparity bit is ‘0’

The updated running disparity is still between -16 and +16, the data are transmitted as such and the disparity bit is ‘0’:

t_i	0	1	0	0	0	0	1	0	0	1	1	0	0	0	0	0	←DB

The next 15 bits of data to be sent are:

t_{i+1}	0	0	0	0	0	1	1	1	0	1	0	0	0	0	1	$D_w = -5$

The updated running disparity of the transmission is:

$RD(t_{i+2}) = RD(t_{i+1}) + D_w + DB = -13 – 5 – 1 = -19$

The running disparity exceeds -16, the data are inverted and the disparity bit is set to ‘1’.

t_{i+1}	1	1	1	1	1	0	0	0	1	0	1	1	1	1	0	1	←DB

The running disparity becomes:

$RD(t_{i+2}) = RD(t_{i+1}) + D_w + DB = -14 + 5 + 1 = -8$

The next data are then processed using the same algorithm.