SRS Data Format

Content

Contents

1. Overview

SRS outputs data using UDP protocol. For each event, each SRS node will output a number of UDP frames containing data and an *Event Trailer* frame containing only a code word (0xFAFAFAFA). The format does not foresee a fixed number of frames per event, but in practice, current applications constrain the number of frames per event to a fixed number.

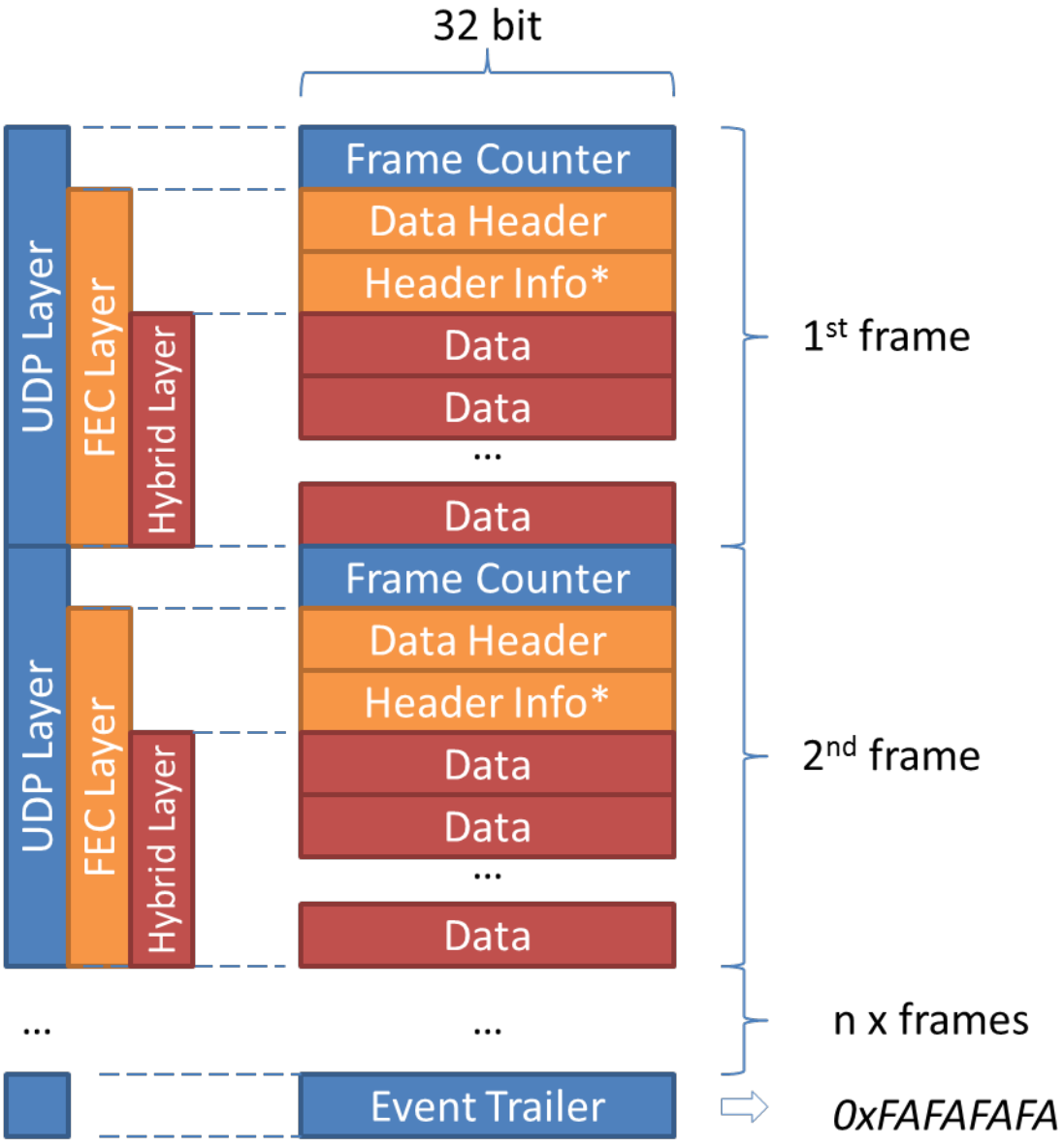


Figure 1. SRS general DAQ data format

The general SRS data format defines 3 layers:

- **UDP Layer:** UDP data payload delivered as raw data by DAQ software like DATE. Contains a either a *Frame Counter* word and the FEC Layer data, or the *0xFAFAFAFA* event trailer word.

- **FEC Layer:** contained in the UDP layer, holds the DAQ data and header information used to identify data source, data packing, etc., and other information useful in the DAQ process.
- **Hybrid Layer:** DAQ data originating from the front-end hybrids. Depending of the data source, this layer may or may not have a particular structure.

1.1.Frame Counter

The Frame Counter is a 32-bit word primarily used to tag each UDP frame corresponding to one event. DATE uses this field to check the integrity of the event. There are 3 different syntaxes of this field corresponding to 3 different ways DATE can use it for integrity validation:

	Byte 3	Byte 2	Byte 1	Byte 0
Single-FEC mode (*default)*	0x00	0x00	0x00	F#
Multiple-FEC mode	TS			F#
Test Mode	GF#			

- **F#** = frame counter value (1 byte) which starts from 0 for a new event and increases for every frame.
- **TS** = 3-byte timestamp tag attributed to each event.
- **GF#** = 4-byte global frame counter which starts from 0 at the beginning of the run and and counts continuously (does not reset for each event).

The frame counter behavior is controlled by SRS registers. By default the system uses the first syntax. Second variant is used with multiple FEC cards, where participation of each FEC to the readout event needs to be validated using the timestamp value. Syntax number 3 is a test mode, never used for taking data.

1.2.Data Header

The Data Header is a mandatory field that essentially identifies the data source. The general format and the current list of values are sown in the table below.

	Byte 3	Byte 2	Byte 1	Byte 0
General format	H0	H1	H2	C#
ADC mode	“A”	“D”	“C”	C#
APV with zero-suppression	“A”	“P”	“Z”	C#
Arizona C-Card	“A”	“Z”	“C”	0x00

- **Hx** = header identification byte; for the current APV configuration HHH = 0x414443 ("ADC" in ASCII)
- **C#** = DAQ channel number
    - in case of APV hybrids (ADC or APZ modes), even numbers represent masters, odd numbers represent slaves.

1.3.Header Info Field

The Header Info field is reserved for future use to allow the transport of event related data or information about the data structure in the Hybrid Layer. The minimum length of this field is one word (32-bit).

	Byte 3	Byte 2	Byte 1	Byte 0
General format	I0	I1	I2	I3

- Ix = reserved for optional info

2. ADC acquisition mode

2.1. Overview

The ADC acquisition mode uses the ADC C-Card equipped with two 8-channel 12-bit ADCs, for a total of 16 ADC channels. The SRS unit (FEC + ADC C-Card) acquires a fixed-length window of digitized data following each trigger pulse (provided that the trigger does not come too early wrt the previous trigger). Data is acquired in parallel from each enabled input channels. The acquisition window is synchronized to a trigger signal which can either be generated internally or taken from outside, using the NIM input of the FEC card.

Relevant SC registers for this mode of operation are listed in the table below. For more information see the *SRS Slow Control Manual*.

Register Name	SC Port	Addr	Description	Default value
BCLK_MODE	APVAPP	0x00	Bit 2 = external(1)/internal(0) trigger Bit 3 = external trigger polarity (NIM logic)	0x07
BCLK_FREQ	APVAPP	0x02	External trigger: post-trigger deadtime Internal trigger: trigger period	40000
BCLK_ROSYNC	APVAPP	0x05	Start of the acquisition window wrt the trigger pulse	300
EVBLD_CHENABLE	APVAPP	0x08	Channel enable register. Each bit corresponds to one ADC channel	0xFFFF
EVBLD_DATALENGTH	APVAPP	0x09	Length of the data-capture window (ADC samples)	3000
EVBLD_MODE	APVAPP	0x0A	Must remain 0	0
RO_ENABLE	APVAPP	0x0F	Readout Enable register (bit 0). Triggers are accepted for acquisition when this bit is 1. Upon boot this is set to 0. Must be set to 1 to start the acquisition.	0

The acquired event will have one UDP frame for each channel enabled in the EVBLD_CHENABLE register. The **C#** field in the Data Header indicates the number of the readout channel. On each HDMI plug there are 2 analog inputs defined as *Master* and *Slave*. Masters will have even channel numbers and Slaves odd channel numbers.

2.2.ADC data format

A. FEC Layer

- **Data Header.** The ADC data format is identified in *the Data Header* field by the ASCII characters "ADC", followed by the ADC channel number (*unsigned byte*):

	Byte 3	Byte 2	Byte 1	Byte 0
General format	H0	H1	H2	C#
ADC mode	“A”	“D”	“C”	C#

- **Header Info:** one 32-bit word (reserved - see chapter 1.3 Header Info Field)

B. Hybrid Layer

In ADC mode acquisition data follows immediately after the header, occupying the entire *Hybrid Layer.* Data is organized as 16bit words with the least-significant byte first (only 12 bits of the 16 are used; the most-significant 4 bits are set to 0):

- Data word 0: D0L|D0M|D1L|D1M
- Data word 1: D2L|D2M|D3L|D3M
- Data word 2: …

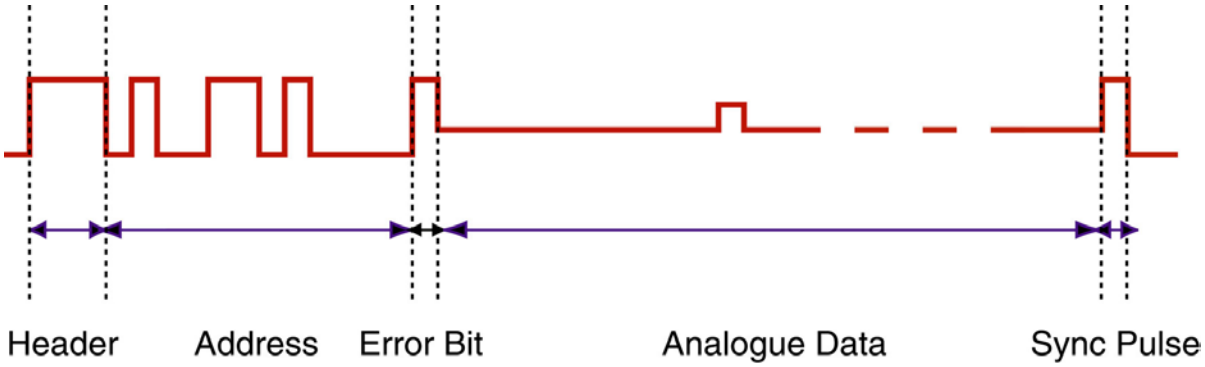
Data can be reconstructed using the following formula

$$Dx = DxL + 256 \times DxM$$

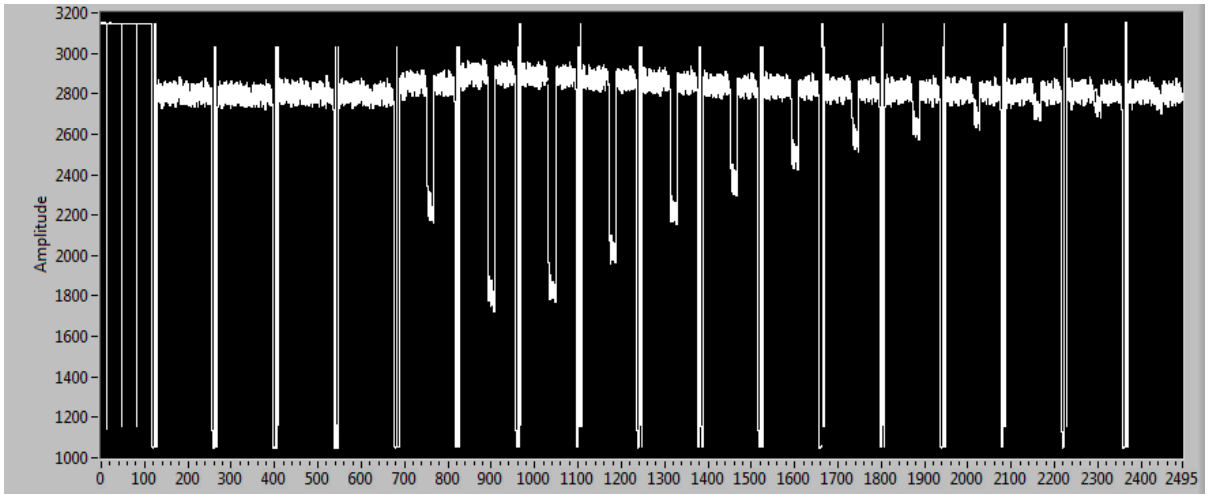
The byte-length of the data field is equal to (EVBLD_DATALENGTH x 2) bytes which correspond to (EVBLD_DATALENGTH) samples (16 bit).

2.3.APV readout using ADC mode (raw APV data)

The ADC mode can be described as an "oscilloscope like" operation mode. The on-board ADC surveys the output of the APV, while the firmware acquires from the ADC a time window which is EVBLD_DATALENGTH samples long and synchronous with the trigger (starting BCLK_ROSYNC x 25ns after the trigger). Unfortunately the APV does not synchronize its data stream on the trigger signal, but on an internal loop that starts at the APV reset and has a period of 35 clock cycles (25ns). The result is that the useful data "jitters" inside the SRS payload by up to 35 samples, thus the offline software need to "hunt" for the beginning of the data, which is generally done by looking for the header formation provided by the APV data protocol (see the APV manual, page 10, chapter 4.3 and page 18, chapter 8).



The APV multiplexes digital and analog data on to the same line; this line arrives inverted at the ADC so you'll find as follows (thresholds are approximate, see picture attached):



Condition	**APV data type**
sample < 1200	digital "1"
sample > 3000	digital "0"
1200 < sample < 3000	analog data

3. APZ data format (APV zero-suppression firmware)

3.1.Data types and structures definition

```
typedef unsigned char BYTE;        // 8-bit word
typedef unsigned int WORD32;       // 32-bit word
typedef unsigned short int WORD16; // 16-bit word
typedef signed short int INT16;    // 16-bit signed int


struct APV_HEADER
{
  BYTE    APV_ID;     // APV Identifier number on the FEC card (0 to 15)
  BYTE    N_CHANNELS; // the number of channels which will be following
                         the header
  BYTE    N_SAMPLES;  // the number of samples per channel
  BYTE    ZS_ERROR;   // Error code from the Zero Suppression Block,
                         meaning have to be defined
  WORD16  FLAGS;      // bit 0 : '0' – Classic zero suppression, '1' –
                         Zero suppression with peak finding
                      // bits 1 to 15 are still reserved for future use
  WORD32  RESERVED;   // 32 bits reserved for future use
};


struct CHAN_INFO
{
  BYTE    RESERVED;            // 8-bits reserved for future use
  BYTE    CHAN_ID;             // Channel identifier,
                               // APV physical channels are 0 to 127,
                               // 128 could be used for the common mode
                                 average
                               // 129 for error codes from the APV
                                 (pipeline address(8 bits) & error bit)
  INT16  CHANDATA[N_SAMPLES];  // 16 bit words, actual data will be 13-
                                 bits wide
};
```