CARLOSrx v4 rel 1 for August 2004 SDD test beam reference manual

Samuele Antinori, Filippo Costa, Davide Falchieri,
Alessandro Gabrielli, Enzo Gandolfi,
Massimo Masetti, Samuele Zannoli

Department of Physics and INFN Bologna

June 2004

Outline

What's new in CARLOSrx v4 rel 1

CARLOSrx v4 rel 1 has changed from the previous versions both for what concerns the hardware, the firmware and the decoding software. CARLOSrx v4 rel 1 acts as an interface between the ASIC CARLOS v4 and the DDL in the context of the SDD readout chain. Here follows a list of the new features in CARLOS rx v4 rel 1.

Hardware upgrades:

- The number of available I/Os has been increased in order to easily interface the TTCrx board when necessary.
- The chip 74ALVC164245, with the purpose of driving the 50 Ω trigger signals, has been integrated in the board layout. In CARLOSrx v3 the chip had been manually connected on the board.
- 6 LEMOs have been added to the board for carrying the trigger signals, such as *trigger*, *busy*, *tdc* and so on.
- The chip CDCVF2310 has been added in order to drive different clock nets without signal distortion. Two clock nets are used for driving the FPGA XC2V1000 and the ASIC CARLOS v4.

Firmware upgrades:

- CARLOSrx receives the JTAG information from the SIU and encapsulates it over the serial backlink towards CARLOS v4.
- CARLOSrx manages the CARLOS v4 back-pressure by sending the commands "Stop acquisition" and "Restart acquisition" when CARLOSrx internal FIFOs are almost full. CARLOSrx also handles the back-pressure in order to avoid the generation of double events (one of which dummy) from AMBRA when working in multi-buffer mode. Using the back-pressure allows to use the DDL even when the flow control is active.
- Two CARLOSrx firmware versions have been developed with a different behavior of the busy signal: single buffer (long busy) and multi buffer (short busy).
- New input CARLOSrx pins have been configured in order to be able to accept the testpulse and prepulse signals.
- A new input CARLOSrx pin has been configured in order to be able to send the L1reject command for testing how events are aborted.

Software upgrades:

- A new version of the C++ decoding software *carlosrx* has been developed. It features an easy to use GUI interface and it also allows to view reconstructed data by using ROOT functions.
- A graphical SW tool for the generation of the JTAG programming file has been developed.

Main features

- XC2V1000 Xilinx Virtex2 FPGA;
- 40 MHz working frequency;
- 1.8 V core power supply; 2.5 V I/O pads power supply;
- standard IEEE 1149.1 JTAG implemented;
- interface towards CARLOS v4 implemented;
- interface towards the SIU implemented (with flow control);
- interface towards the trigger system implemented;
- it can be directly interfaced either to CARLOS or to the optical link

Known limitations

- Even if the board allows to acquire data coming from two CARLOS chips (that is from two detectors) the current version of the firmware of CARLOSrx allows to acquire data coming from 1 CARLOS (= 1 detector only).
- The CARLOSrx board can be connected to CARLOS in either of 2 ways:
    - direct connection;
    - with the optical link (GOL + optical fiber + TLK1501)

  If used in the second configuration, the clock and the serial back-link running from CARLOSrx to CARLOS have to be carried with wires (not with optical fibers as expected in ALICE). In fact no optical transceiver for these two signals has been foreseen on the CARLOSrx board.
- The current version of the firmware of CARLOSrx allows to interface a 1-level trigger with a simple trigger – busy scheme. This version is not able to interface the 3-levels trigger system expected for ALICE and it does not interface the TTCrx device.

General description

CARLOSrx v4 rel 1 is a Xilinx Virtex2 FPGA-based device with the main purpose of concentrating data coming from one SDD detector on one LDC.
CARLOSrx packs data coming from the front-end electronics and CARLOS through the optical links into 32-bit words, stores them in a large data FIFO and then sends them towards the DDL system, after a transaction has been opened by the SIU. The use of a FIFO allows not to lose any data even when the DDL asserts the flow control: in fact CARLOSrx asserts the back-pressure towards CARLOS and CARLOS will stop AMBRA, thus freezing the data acquisition process until the DDL is ready to accept data again.
CARLOSrx also drives the serial backlink port towards CARLOS for sending JTAG information to PASCAL, AMBRA, CARLOS and GOL and for sending to CARLOS reset commands, trigger signals and control commands.
It also interfaces the trigger system by receiving the *trigger* signal and asserting the *busy* signal. It also asserts the *tdc* signal that will be used by the TDC chip in order to determine the latency occurred from the moment the trigger arrives to the moment it is actually sent to AMBRA.
Fig. 1 shows a schematic representation of the chain implemented in our Lab. The chain has also been successfully tested using an optical link between CARLOS and CARLOSrx, including the GOL chip, 50m of optical fiber and the deserializer TLK1501.
This firmware implemented on the FPGA contains 5 major logic blocks:

1. *data packing*: CARLOSrx receives the 16-bit data words coming from CARLOS, groups them depending on their type, packs them into 32-bit words and stores them into a FIFO, before they are sent towards the SIU.
2. *SIU interface*: this block manages the protocol interface towards the SIU. It is able to recognize the commands sent from the SIU and then to send packed data towards the SIU.
3. *trigger interface*: this block directly interfaces the trigger system by receiving the trigger input and asserting the busy signal. When CARLOS is in RUN mode the busy signal value reflects the related value received from the CARLOS error flag words.
4. *JTAG interface to SIU*: this block receives the JTAG signals from the SIU and encapsulates them on the serial back-link towards CARLOS. It also implements 2 JTAG instructions: "Put CARLOS in JTAG mode" and "Put CARLOS in RUN mode". When one of these instructions is detected, a signal is sent to the serial backlink block in order to send to CARLOS the corresponding command.
5. *serial backlink*: this block drives the *serial backlink* signal from CARLOSrx to CARLOS. It is used to send JTAG commands, reset commands, trigger signals, prepulse and testpulse signals, the commands "Enter JTAG mode" and "Enter RUN mode".

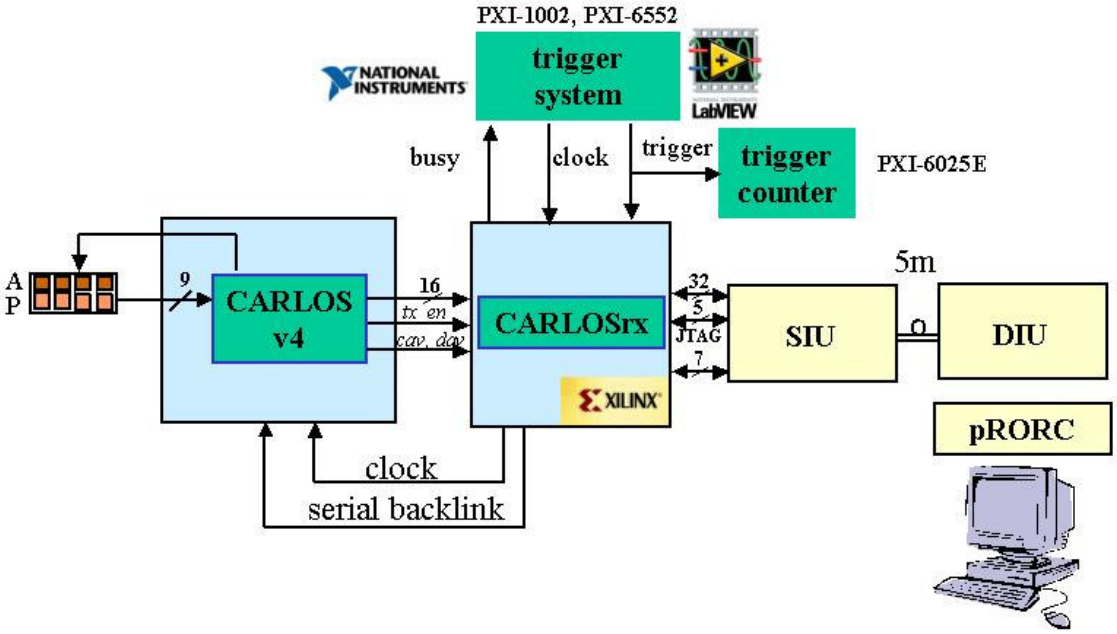


Fig. 1: Schematic representation of the SDD readout chain implemented in Lab.

Interface to CARLOS

CARLOSrx receives data coming from CARLOS v4. These are the interface signals:

- *output_data* (16 bits): this is the 16-bit bus containing the data coming from CARLOS v4;
- *tx_en*: it is a strobe signal, active high. When active the *output_data* bus contains a valid value, whichever its type (header, footer, data from ch1, data from ch0, error flag word, JTAG word).
- *cav*: it is a strobe signal, active high. When active, the *output_data* bus contains a valid control word, that is either a JTAG word or an error flag word.
- *serial backlink*: it is a serial link from CARLOSrx to CARLOS. It is used to send 8-bit commands to CARLOS, such as reset signals, trigger signals, JTAG information and commands for putting CARLOS in JTAG mode or in RUN mode.

The *serial backlink* block on CARLOSrx is used to drive the *serial backlink* signal. After the reset is activated, the block sends a number of IDLE codes for link synchronization, then it sends the commands for resetting all the front-end chips (PASCAL, AMBRA, CARLOS and GOL). Then it continues sending IDLE codes until it has to send one of the following commands (from highest priority to lowest: if two commands have to be sent at the same time, the highest priority is sent first)):

- enter JTAG mode;
- enter RUN mode;

- trigger signal;
- JTAG information;
- L1reject;
- L2reject;
- prepulse25;
- testpulse;
- prepulse50, 75, 100, 125, 150, 175, 200
- stop acquisition;
- restart acquisition.

Since these commands are to be sent on a 8-bit serial link, there is a variable jitter of a few clock cycles from the moment a signal is received to the moment it is actually sent to CARLOS over the serial back-link.
This is the list of actions occurring when a signal is about to be transferred over the serial back-link (refer to Fig. 2):

- a 3-bit counter, *ibit*, continuously runs from 0 to 7: a complete 8-bit command is sent in output over the serial back-link in 8 clock cycles, from *ibit* = 0 to *ibit* = 7. In each of these 8-period slots a command is sent in output.
- on the rising edge of the clock *trigger* is sampled and its value passed to *trigger_sync1*;
- on the next rising edge of the clock *trigger_sync1* is sampled and its value passed to *trigger_sync*;
- on the next rising edge of the clock *trigger_sync1* is sampled and its value passed to *trigger_to_serial* if busy = 0;
- on the next rising edge of the clock *trigger_to_serial* is sampled and its value passed to *trigger_long*;
- the internal state machine driving the serial back-link output continuously checks for the *trigger_long* signal to be high. When it does, it changes state from DECISION_STATE to L0_STATE.
- When state is L0_STATE, the 8-bit register *vector1* is updated with the trigger command value AC.
- Then when *ibit* = 7, the 8-bit register *vector* is updated with *vector1* value and then the command is transmitted.
- The *tdc* signal is activated for one clock cycle when a trigger command is being sent over the serial back-link and *ibit* = 2.

In the example shown in Fig. 2 the latency from the trigger arrival to the moment in which the related commands starts to be sent over the serial back-link is about 12 clock cycles.
Figure 3 shows that this latency can be smaller (8 clock cycles) depending on the relative timing between the trigger arrival time and the *ibit* counter value. The latency might also be larger if the trigger occurs while an other command is being transmitted over the serial link.

A proposal of upgrade of the CARLOSrx firmware suggests to store the latency of the trigger signal (as well as the testpulse and prepulse) in a FIFO and send them towards the DDL on the DDL header.

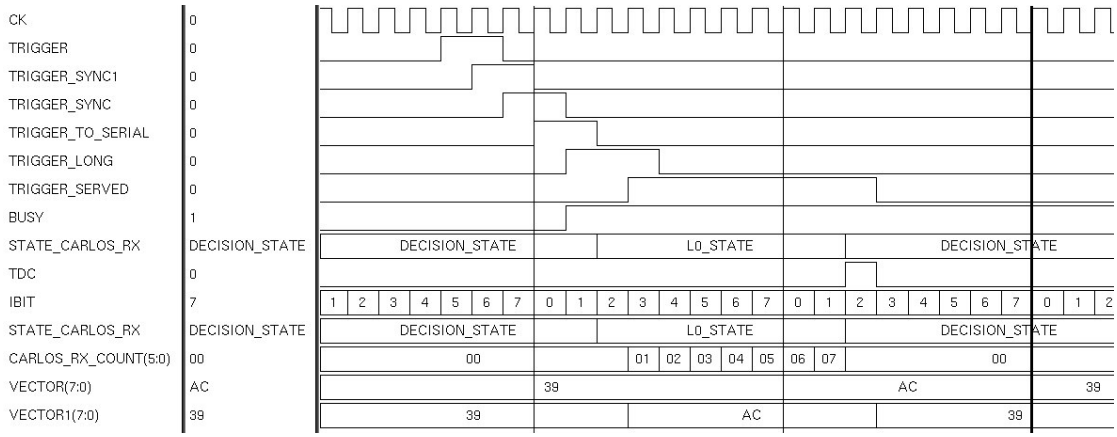


Fig. **2**: In each 8-cycle timing slot (between vertical bars) a command is sent over the serial back-link. In this case the *trigger – tdc* latency is 12 clock cycles.

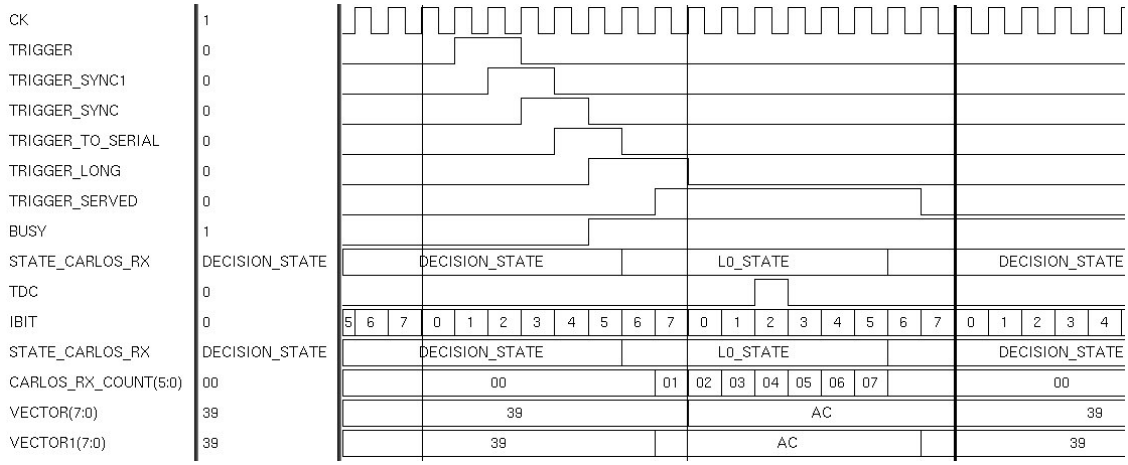


Fig. **3**: In this case the *trigger tdc* latency is 8 clock cycles.

N.B. The main difference between this timing and the same one at the August 2003 SDD test beam concerns the two signals trigger_sync1 and trigger_sync that have been introduced in August 2004 SDD test beam only. This implies a fixed 2 clock cycles delay added to the *trigger – tdc* timing.

The number of clock cycles from the moment the *tdc* signal is asserted and the moment CARLOS asserts its trigger outputs towards AMBRA is fixed and equal to 8 (see Fig. 3.1).

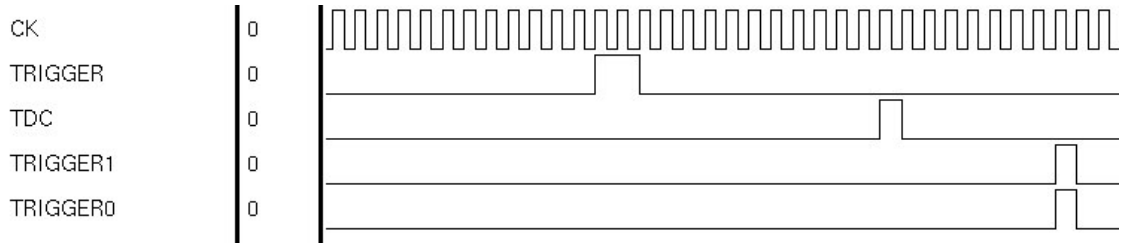


Fig. **3.1**: Legend: *trigger* is CARLOSrx input trigger, *trigger1* and *trigger0* are CARLOS trigger outputs towards AMBRA. The latency between *tdc* and *trigger1* (*0*) is fixed and equal to 8 clock cycles.

Interface to the SIU

A list follows of the signals involved in the CARLOSrx- SIU interface (see Fig. 4):

- *fidir*: it is an input to CARLOSrx. It asserts the direction of the data flow between CARLOSrx and the SIU: when 0 the direction is from the SIU to CARLOSrx, when 1 the direction is from CARLOSrx to the SIU.
- *fiben_n*: it is an input to CARLOSrx, active low. It enables the communication on the bidirectional bus between CARLOSrx and the SIU. When 0 the communication is enabled, when 1 the communication is disabled.
- *filf_n*: it is an input to CARLOSrx, active low, "lf" stands for link full. When the SIU is no longer able to accept data coming from CARLOSrx it asserts this signal. When this happens CARLOSrx sends an other valid data word, then stops transmitting waiting for the *filf_n* signal to switch back to 1. This is the signal used by the SIU to implement the back-pressure on the data flow running from the front-end to the data acquisition system.
- *foclk*: it is a free running clock generated on CARLOSrx and driving the CARLOSrx-SIU interface. It is a 20 MHz clock generated by dividing the system clock frequency by two. Interface signals coming from the SIU change state on the falling edge of *foclk*.
- *fbten_n*: it is a bidirectional signal, active low, it can be driven by CARLOSrx or by the SIU, "ten" stands for transfer enable. When CARLOSrx is assigned to drive the bidirectional buses (when *fidir* is 1 and *fiben_n* is 0) *fbten_n* value is asserted from CARLOS: it turns to its active state when CARLOSrx is transmitting valid data to the SIU, otherwise it is inactive. When the SIU is assigned to drive the bidirectional buses (when *fidir* is 0 and *fiben_n* is 0) *fbten_n* value is asserted from the SIU: it turns to its active state when the SIU is transmitting valid commands to CARLOSrx, otherwise it is inactive.
- *fbctrl_n*: it is a bidirectional signal, active low, it can be driven by CARLOSrx or by the SIU, "ctrl" stands for control. When CARLOSrx is assigned to drive the bidirectional buses (when *fidir* is 1 and *fiben_n* is 0) *fbctrl_n* value is asserted from CARLOSrx: it turns to its active state when CARLOSrx is transmitting a Front End Status Word to the SIU, otherwise, when in the inactive state, CARLOS is sending normal data to the SIU. When the SIU is assigned to drive bidirectional buses (when *fidir* is 0 and *fiben_n* is 0) *fbctrl_n* value is asserted from the SIU: it turns to its active state when sending command words to CARLOSrx, to its inactive state when sending data words. The second option has not been implemented on CARLOSrx since we decided that CARLOSrx needs only commands and not data from the SIU.

- *fobsy_n*: it is an input signal to the SIU, active low, "bsy" stands for busy. CARLOS should put this signal active when not able to accept data coming from the SIU. Since CARLOSrx has not to receive data from the SIU, this signal has been fixed at 1, meaning that CARLOSrx will never be in a busy state. In fact it always has to accept command words coming from the SIU.
- *fbd*: it is a bidirectional 32-bit bus on which data or command words are exchanged between CARLOSrx and the SIU.

This is the way the communication protocol works:
The SIU acts as the master and CARLOSrx acts as the slave, that is the SIU sends commands to CARLOSrx and CARLOSrx sends data and front end status words to the SIU. At first the link CARLOSrx - SIU has to be initialized and the SIU acts as the master of the bidirectional buses. So CARLOSrx waits for the bidirectional buses to be driven from the SIU (*fidir* is 0 and *fiben_n* is 0) and waits for a valid (*fbten_n* = 0) command (*fbctrl_n* = 0) named Ready to Receive (RDYRX). This command is always used in order for a new event transaction to begin. The RDYRX command contains a transaction identifier (bits 11 to 8) and the string "00010100" as the less significant bits. As the command is accepted and recognized CARLOSrx waits for the *fidir* signal to change value in order to take possession of the bidirectional buses, then, if the *filf_n* is not active, it is able to send valid data on the *fbd* bus if it has any.
Each data packet begins with the DDL header. At the end of a data packet CARLOSrx puts in output the Front End Status Word, a word that confirms that no errors occurred and that the whole event has been successfully transferred to the SIU. The Front End Status Word contains the Transaction Id code received upon the opening of the transaction (bits 11 to 8) and the 8-bit FESTW code "01100100". After this happens CARLOSrx begins waiting for some action of the SIU to be taken: it means that the SIU can decide to take back its control on the bidirectional buses and close the data link towards the data acquisition system, or the SIU can leave the bidirectional buses control to CARLOSrx for an other data event to be sent. So far CARLOSrx begins waiting 16 *foclk* periods: if nothing happens CARLOSrx is able to begin sending data again without the need to receive some other commands from the SIU; if the SIU takes back the possession of the bidirectional buses CARLOSrx closes the link towards the SIU and keeps waiting for an other RDYRX command asserted from the SIU itself.

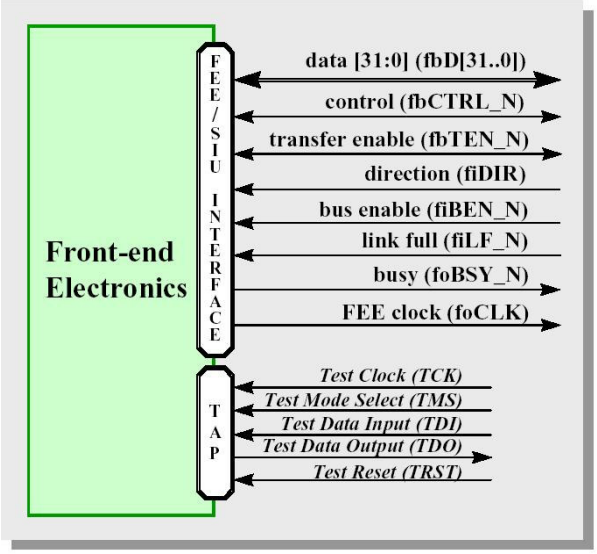


Fig. 4: CARLOSrx – SIU interface

Interface to the trigger system

CARLOSrx interfaces the trigger system with the 2 following signals:

- *trigger*: it is the trigger signal received from the trigger system, active high. It is completely asynchronous with respect to the clock and its width is 80 ns.
- *busy*: it is the busy signal from CARLOSrx to the trigger system, active high.

For what concerns the busy signal, two CARLOSrx firmware versions have been developed and tested:

- *single buffer version*: the busy signal is asserted after receiving the trigger signal and put to 0 again when the whole event has been transmitted to the DDL. In this configuration only one of the 4 AMBRA buffers is used at a time. When using anode length = 256, the busy on mean time is 2 ms.
- *multi buffer version*: the busy signal is asserted after receiving L0 and put to 0 again when the corresponding busy signal from CARLOS turns to 0 again. In this way CARLOSrx busy is a copy of the busy signal received from CARLOS in the error flag words.
  In this configuration all the 4 AMBRA buffers can be used. Using standard configuration for PASCAL (analog memory full frequency and ADC half frequency) and anode length = 256, the busy on time is 500 μs when some buffer is free and 1.6 ms when all AMBRA buffers are full.

JTAG instruction set

CARLOSrx receives the JTAG port from the SIU and encapsulates the JTAG information on the serial back-link towards CARLOS as it is. The JTAG *tck* frequency has to be at most 5 MHz (suggested value = 5 MHz). Higher *tck* frequencies will lead to errors in JTAG programming and reading back register values.
Beside that, CARLOSrx internal JTAG unit monitors the input JTAG port looking for the JTAG instructions reported in Table 1.

JTAG instruction	JTAG IR value	Length of scan register involved
Put CARLOS in JTAG mode	10001	5
Put CARLOS in RUN mode	10010	5
Load anode length	10011	5
Load trigger delay	10000	5

Table 1: List of CARLOSrx JTAG instructions

- After decoding the instruction "Put CARLOS in JTAG mode", the command "Enter JTAG mode" is sent to CARLOS through the serial backlink.
- After decoding the instruction "Put CARLOS in RUN mode", the command "Enter RUN mode" is sent to CARLOS through the serial backlink.
- After decoding the instruction "Load anode length", the value of a 8-bit register is read from the JTAG port and put in output as ninth word of each event sent towards the LDC as LSBs. All the other bits of the ninth word are set to 1. For example, if the anode length value received is 0xC7, the value of the ninth word sent in output within each event is: FFFFFFC7. This information may prove useful when decoding data of unknown anode length (decoding information mixed together with data).
- After decoding the instruction "Load trigger delay", the value of a 8-bit register is read from the JTAG port and stored internally. The value of this register is the number of clock cycles of latency with which the backpressure is applied after receiving a trigger, with the purpose of avoiding faulty double events from AMBRA. This value is directly related to the SOP delay register value on AMBRA with the following relationship:

$$trigger\ delay\ value = SOP\ delay + 49$$

Back-pressure

CARLOSrx v4 rel 1 makes use of the back-pressure as soon as it needs to, that is as soon as its internal FIFO is going to get full. CARLOSrx internal FIFO contains a 20k 32-bit word dual-clock RAM, provided as a core by Xilinx ISE software. Since the allowed number of words is a power of 2, the FIFO is obtained by putting in series 2 FIFOs:

- *small fifo*: one 4k 32-bit words FIFO;
- *large fifo*: one 16k 32-bit words FIFO.

As soon as a word enters a FIFO, it is first written in the *small fifo*. Then an automatic process scans the *small fifo* and, as soon as the *small fifo* is no longer empty, it pops the *small fifo* and pushes the word in the *large fifo*. Then the *large fifo* is popped when the SIU is ready to accept data (the SIU has opened a transaction and the flow control has not been activated).
The back-pressure is activated when the *small fifo* has only 1k 32-bit free locations available (3k on 4k are used cells), see Fig. 5. Then the back-pressure is de-activated when the *small fifo* has 2k 32-bit free locations (half of the small fifo is available).
Back-pressure is needed when the input data rate is larger than the output data rate, that is DDL input data rate. Backpressure is very important especially:

- when transmitting large events (anode length = 256);
- when no compression is operated on CARLOS;
- when the trigger rate is high (= when AMBRA works in multi-buffer mode)

So far the back-pressure can be activated very often during an acquisition run when these conditions apply. Of course it is sufficient to use CARLOS as a compressor chip in order to avoid using the back-pressure and so to decrease data processing and transmission delay.
The back-pressure mechanism is also activated after a programmable number of clock periods from the moment a trigger is received and immediately de-activated. This number is given by the value of the JTAG trigger delay register. The purpose is to be sure that AMBRA does not send some events twice, as it happens to do when working in multi-buffer mode and with trigger rates near to the maximum one (611 Hz). The activation of the back-pressure with a fixed latency after the arrival of each trigger allows to avoid the coincidence in time of two different signals internal to AMBRA managing its internal buffers.

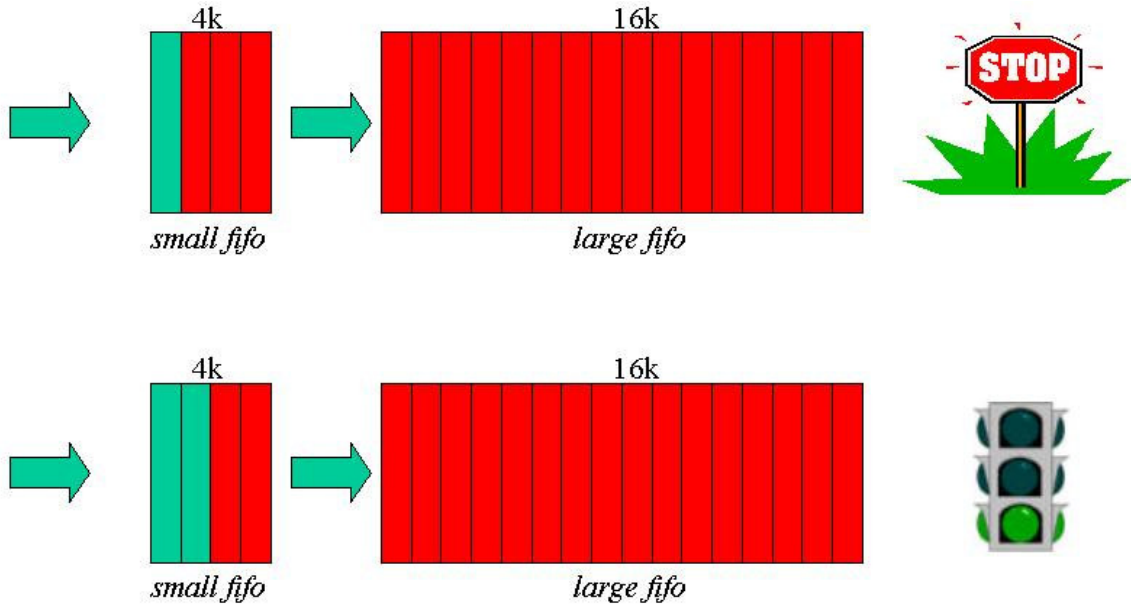


Fig. 5: Back-pressure is activated when the small fifo is almost full and de-activated when the small fifo is half full.

CARLOSrx v4 rel 1 pin function

Terminal name	Type	Description
carlos4_output(15-0)	I	Input data bus coming from CARLOS v4
tx_en	I	Input signal coming from CARLOS v4: when 1 it means that CARLOS is sending a valid word
cav	I	Input signal coming from CARLOS v4: when 1 it means that CARLOS is sending a valid error flag word or JTAG word
dav	I	Input signal coming from CARLOS v4: when 1 it means that CARLOS is sending a valid data
serial backlink	O	Output signal towards CARLOS v4: it carries JTAG information and commands (reset, trigger, backpressure, …)
reset_carlos	O	Output signal resetting CARLOS v4 (active low reset)
gol_ready	O	Output signal towards CARLOS v4 (when the GOL is not used): its value is fixed to 1
ck	I	Input clock
reset_n	I	Active low reset
fidir	I	From the SIU: it decides the bidirectional port direction
fiben_n	I	From the SIU: it decides if the bus is enabled or not (active low)
filf_n	I	From the SIU: it decides if the link is full or not (active low)
foclk	O	To the SIU: 20 MHz free running clock
fbten_n	INOUT	To and from the SIU: when active (low) the data *fbd* is valid
fbctrl_n	INOUT	To and from the SIU: when active (low) *fbd* contains the FESTW
fobsy_n	O	To the SIU: when active (low) CARLOSrx is not ready to accept data from the SIU
fbd	INOUT	32-bit data bus between CARLOSrx and SIU
tdi_from_siu	I	JTAG *tdi* from SIU
tck_from_siu	I	JTAG *tck* from SIU
tms_from_siu	I	JTAG *tms* from SIU
trst_from_siu	I	JTAG *trst* from SIU
tdo_to_siu	O	JTAG *tdo* to SIU
trigger	I	Input trigger, active high, 80 ns wide
busy	O	Output busy
L1reject	I	Input L1reject, active high, 80 ns wide
testpulse	I	Input L1reject, active high, 80 ns wide
tdc	O	Output signal, active for one clock cycle when the trigger command is being sent over the serial backlink

CARLOSrx v4 rel 1 operation

This section contains an explanation of the sequence of actions needed to program and run CARLOSrx operationally.

Using CARLOSrx

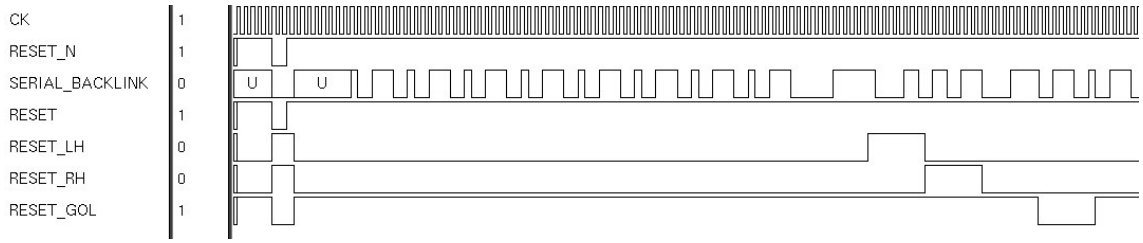
CARLOSrx utilization should include the following sequence of actions:

1) power supply to the front-end electronics, to CARLOS, CARLOSrx and the DDL is turned on. CARLOSrx receives a signal reset (active low) either from an external RC network or from the outside on the *reset_n* pin. After being reset, CARLOSrx begins sending reset commands to the front end chips on the left hybrid, the front end chips on the right hybrid and CARLOS, one after the other using the serial backlink. Busy = 1. See Timing 1.
2) Using the JTAG port the SIU sends to CARLOSrx the command "Put CARLOS in JTAG mode". As a consequence CARLOSrx sends to CARLOS the command "Enter JTAG mode" using the serial backlink. Busy = 1.
3) Using the JTAG port (*tck* = 5 MHz) the SIU sends to CARLOSrx commands and data for addressing, programming and reading back the selected device (the chosen PASCAL, AMBRA or CARLOS). CARLOSrx encapsulates this information on the serial backlink towards CARLOS. After each chip has been programmed, the JTAG connection is closed by asserting the *trst* signal. After this step is over all the selected front-end chips have been programmed. At this point all the JTAG information read from the front-end chips and from CARLOS is stored in the internal FIFO of CARLOSrx. Busy = 1. See Timing 3.
4) Using the JTAG port the SIU sends to CARLOSrx the command "Put CARLOS in RUN mode". As a consequence CARLOSrx sends to CARLOS the command "Enter RUN mode" using the serial backlink. After this step, CARLOSrx begins waiting for the error flag words coming from CARLOS one every 64 clock cycles containing the busy value. So far when CARLOS is in RUN mode, the busy value asserted towards the trigger system is the one received by CARLOS. Should CARLOS be brought back in JTAG mode, then the busy signal would be fixed to 1 again by CARLOSrx, meaning that no trigger signal can be accepted.
5) CARLOSrx begins waiting until the SIU opens a transaction by sending the RDYRX (Ready to Receive) command to CARLOSrx on the 32-bit bidirectional bus *fbd*. Then CARLOSrx takes possession of the bidirectional bus until the transaction is closed. Busy = 1. See Timing 2.

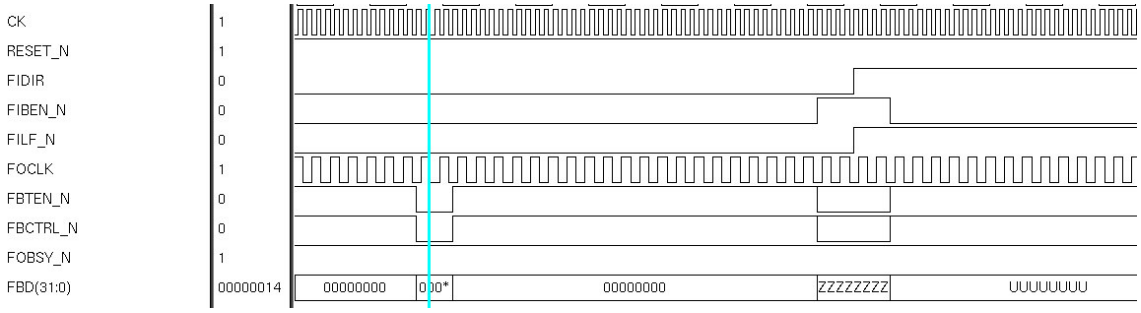
6) After the transaction is opened, CARLOSrx sends a DDL header (8 32-bit words whose meaning is explained later in this paragraph), then it begins emptying its internal FIFO by sending the JTAG words towards the SIU.
7) After CARLOSrx receives a trigger, it sends the related command to CARLOS using the serial backlink and it asserts the busy output value. Then CARLOSrx begins waiting for the data packets coming from CARLOS.
8) Valid words coming from CARLOS are grouped into 32-bit words depending on their meaning:
    - header words;
    - footer words;
    - data from channel 1 (right hybrid);
    - data from channel 0 (left hybrid);
    - error flag words;
    - JTAG words.

   The MSBs are used for identifying the words when the data packet shall be decoded. The first data words (after the DDL header) belonging to the first event are the JTAG words stored in the CARLOS rx FIFO during the JTAG mode, then the event data words follow.
   After coding, 32-bit words are stored into a dual clock FIFO containing 20K 32-bit words. The FIFO allows to implement the flow control between CARLOSrx and the SIU. The choice of a dual clock FIFO is due to the fact that data are written into the FIFO with a 40 MHz clock, while they are read with an internally-generated 20 MHz clock (*foclk*). In fact 32-bit data are sent to the SIU with a 20 MHz clock (= 640 Mbit/s) since the total bandwidth of the DDL is 800 Mbit/s. See Timing 4.
   Each data packet begins with the DDL header (8 32-bit words) containing information on the orbit number and on errors occurred during the transmission.
9) After receiving a complete event from CARLOS (the 3 footer words have been received) the data packet to the SIU is closed with 3 32-bit footer words and with a FESTW (Front End Status Word). In the following 16 *foclk* cycles the SIU might send the EOBTR (End Of Block Transfer) command. Otherwise CARLOSrx begins sending data again. See Timing 5.
10) If any errors occurred during the transmission of an event, after the event has been completely transmitted a dummy event follows with the error bits asserted in the DDL header and a FESTW (see Timing 6).
11) CARLOSrx implements the flow control. This means that each time the SIU board can no longer accept input data from CARLOSrx and it asserts the *filf_n* signal, CARLOSrx stops sending data, while it continues to receive data from CARLOS. After the SIU board is ready to accept data again, the *filf_n* signal is de-asserted and CARLOSrx begins to send data to the SIU again. See Timing 7 and 8.

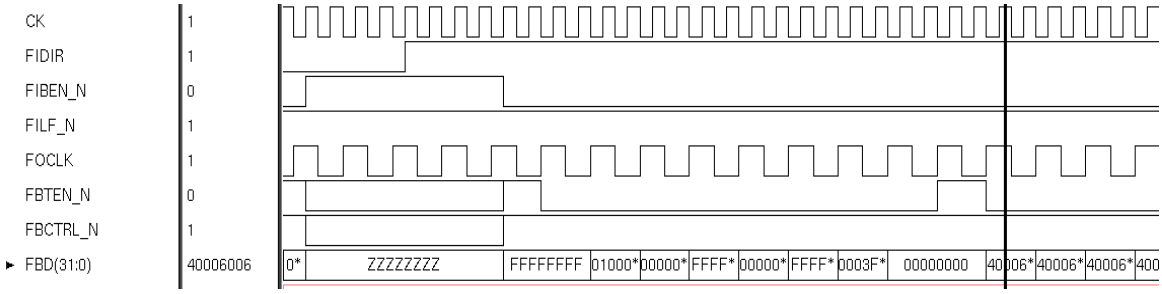
12) If an error occurs in the AMBRA – CARLOS communication (for instance if CARLOS does not receive the *data_end* signal in the expected time slot) and if StopIfError is 1, after closing the data packet, CARLOS asserts both the *data_stop* signals, thus stopping the acquisition of further events from AMBRA. In this case using the JTAG port the SIU has to send the command "Put CARLOS in JTAG mode" and, if necessary, reprogram some of the front-end chips. Then the SIU will send the command "Put CARLOS in RUN mode", so that CARLOS will de-assert the data-stop signals and the data acquisition will begin again.



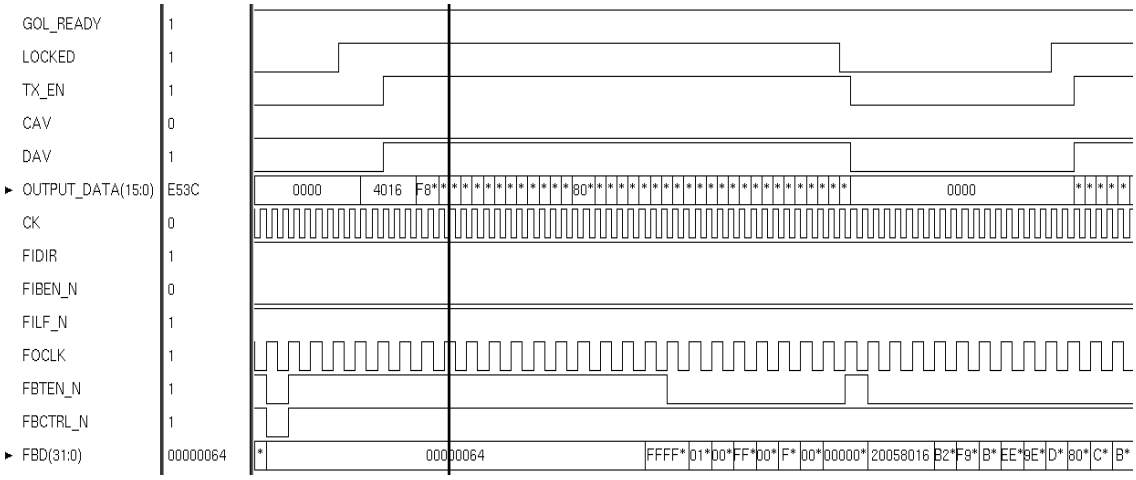
Timing 1: CARLOSrx receives the reset signal (*reset_n*). Then it sends IDLE commands on the *serial backlink*, followed by the reset commands for the front-end chips.



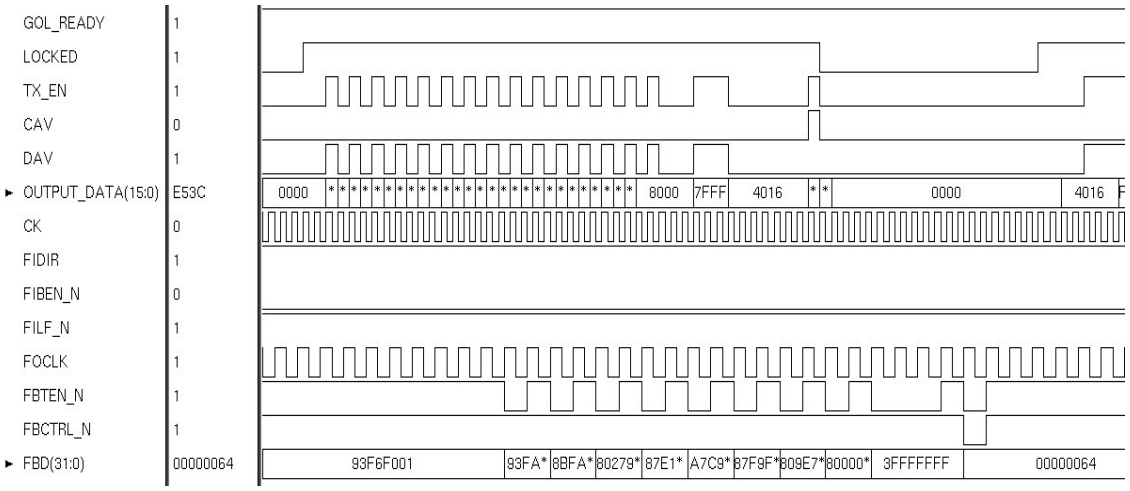
Timing 2: CARLOSrx receives the RDYRX command on the *fbd* bus, then it waits until the SIU changes the *fidir* value. Then CARLOSrx takes possession of the bidirectional bus.



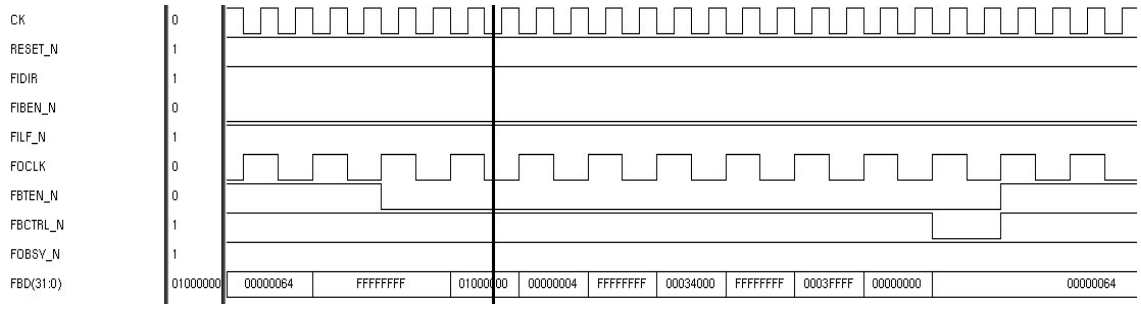
Timing 3: After the opening of a transaction, CARLOSrx sends in output the 8 32-bit words of the DDL header, followed by the JTAG words stored in the FIFO.



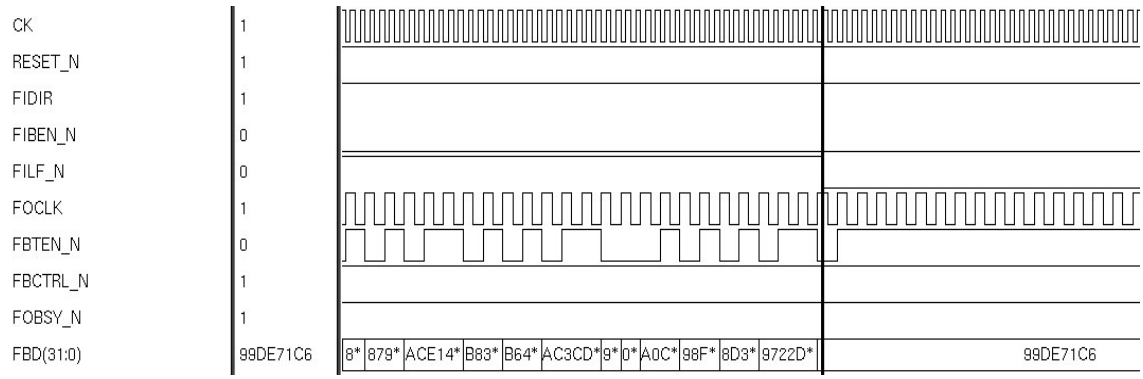
Timing 4: CARLOSrx packs into 32-bit long words the data packet coming from CARLOS. Each data packet from CARLOSrx to the SIU begins with the DDL header (8 32-bits words) followed by 3 CARLOS header words.



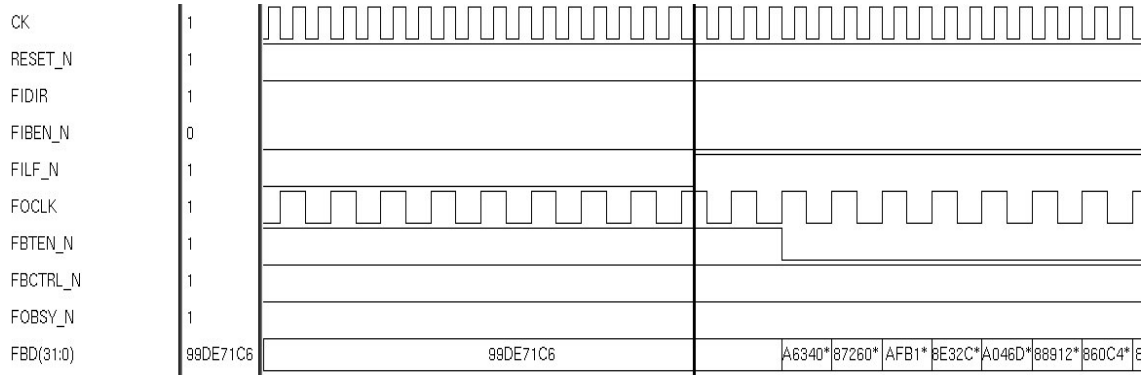
Timing 5: CARLOSrx closes a data packet with 3 footer words, then it sends the FESTW to the SIU.



Timing 6: Dummy event after a faulty event: the orbit number is the same as the previous event and the status and error bits are asserted.



Timing 7: Flow control: the SIU asserts the *filf_n* signal. CARLOSrx stops sending valid data on *fbd* (*fbten_n* = 1). Data coming from CARLOS are written into the internal FIFO.



Timing 8: Flow control: when *filf_n* is switched back to 1, then CARLOSrx begins emptying the internal FIFO.

Data processing

CARLOSrx (see Fig. 6) reads into an internal 16-bit register the data bus coming from CARLOS when it contains a valid value (*tx_en* = 1). Then, depending on the type of data word, it groups together words of the same type into 32 bit words. Error flag words coming from CARLOS are stored only during the timing frame of a data event: error flag words arriving outside these frames are discarded. In the same way error flag words are discarded when CARLOSrx is asserting the back-pressure towards CARLOS. It also changes the MSBs of these words in order to be able to recognize them after the packing process. The 32-bit words resulting from the packing process have the following format:

bit 31	bit 30	bit 29	bit 28	bit 27 – 0	word type
0	0	1	0	header[13-0], header[13-0]	header
0	0	1	1	footer[13-0], footer[13-0]	footer
0	1	0 & JTAG word [14-0]			JTAG word
0	0	0	0 & errflag [13-0]		error flag word
1	0	II output_data[14-0] & I output_data[14-0]			data from ch0
1	1	II output_data[14-0] & I output_data[14-0]			data from ch1

Table 2: 32-bit format from CARLOSrx to the SIU

In the 32-bit word, the first data received from CARLOS is packed as LSBs, while each second data is packed as MSBs.
When a 32-bit word is complete, it is written into a 20K 32-bit words long FIFO (see Fig. 6). Then data are popped from the FIFO synchronously with *foclk*, that is half the system frequency (20 MHz).
It may also happen that one of the data channels from CARLOS v4 sends an odd number of data, so that the related 32-bit register on CARLOS remains incomplete. In this case its value is put in output with the II *output_data[14-0]* containing all zeros before the footer words are sent in output.

Data transmission protocol

After the transaction has been opened, CARLOSrx begins sending data towards the SIU in the same order as the words received from CARLOS. While in RUN mode CARLOSrx sends 32-bit data containing real data words and error flag words. Each data packet begins with 3 header words (see Table 2) and ends with 3 footer words.
While data from CARLOS are generated in an alternated way, data flowing from CARLOSrx to the SIU have a bit (bit 30) in order to clearly identify between channels.
Every event begins with the DDL header (8 32-bit words) and ends with the FESTW (Front End Status Word) (see Fig. 7). The DDL header is required by the DATE v4.6

SW. It usually contains information regarding block length, L1 message, event ID, orbit number, participating sub-detectors, status and error bits, mini-event ID, ROI (Region of Interest). Since CARLOSrx does not receive most of this information, most of these fields are not used and, for this reason, they are filled with 0 or 1 as required by DATE v4.6.
The only meaningful information fields are:

- **format version**: DATE v4 requires this field to be 00000001;
- **orbit number**: this is an incremental number associated to every event;
- **status & error bits**: this field is usually filled with 0s when transmitting error-free events. If any error is detected during the transmission of an event, it is followed by a dummy event (see Fig. 8) containing the same orbit number as the previous event and the following bits asserted in the status & error bits field:
    - bit 5: ask DATE to stop the run;
    - bit 4: no central trigger present;
    - bit 2: data parity error.

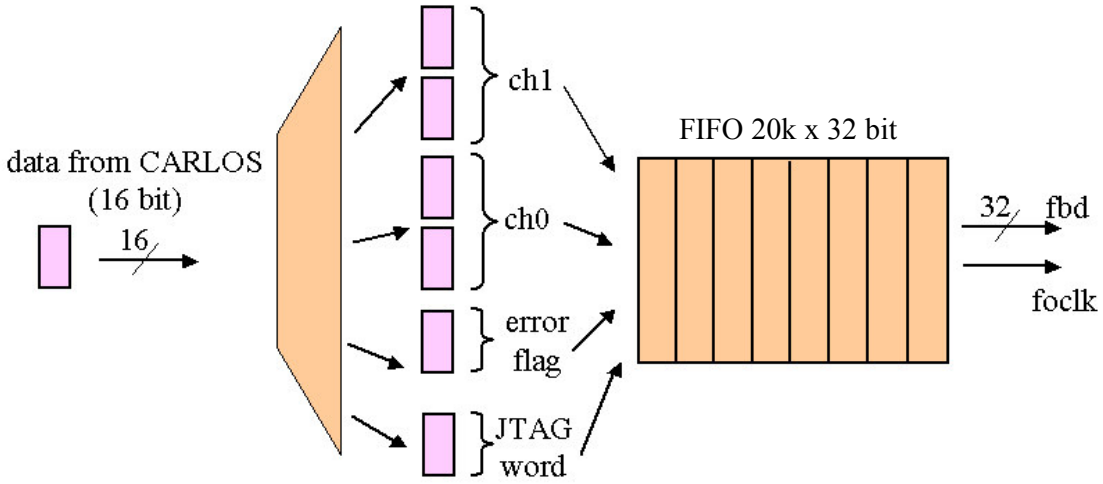


Fig. 6: Data packing and storing on CARLOSrx

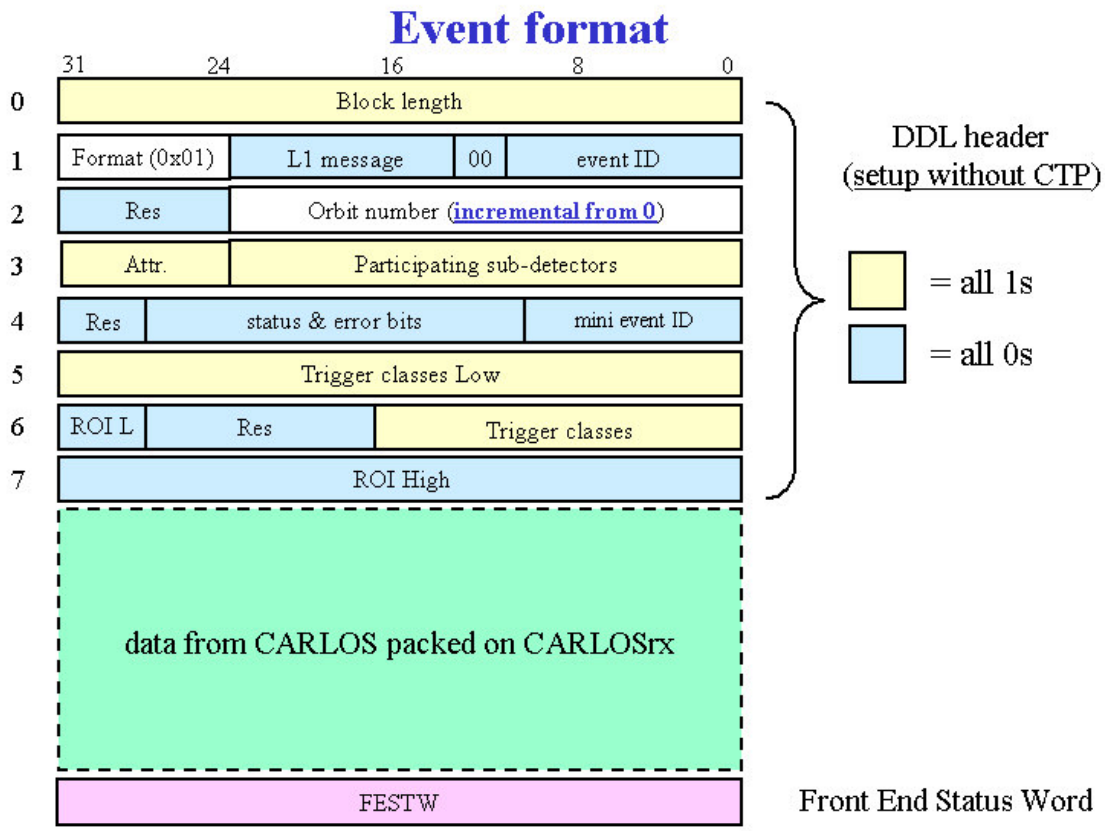


Fig. 7: CARLOSrx event data format

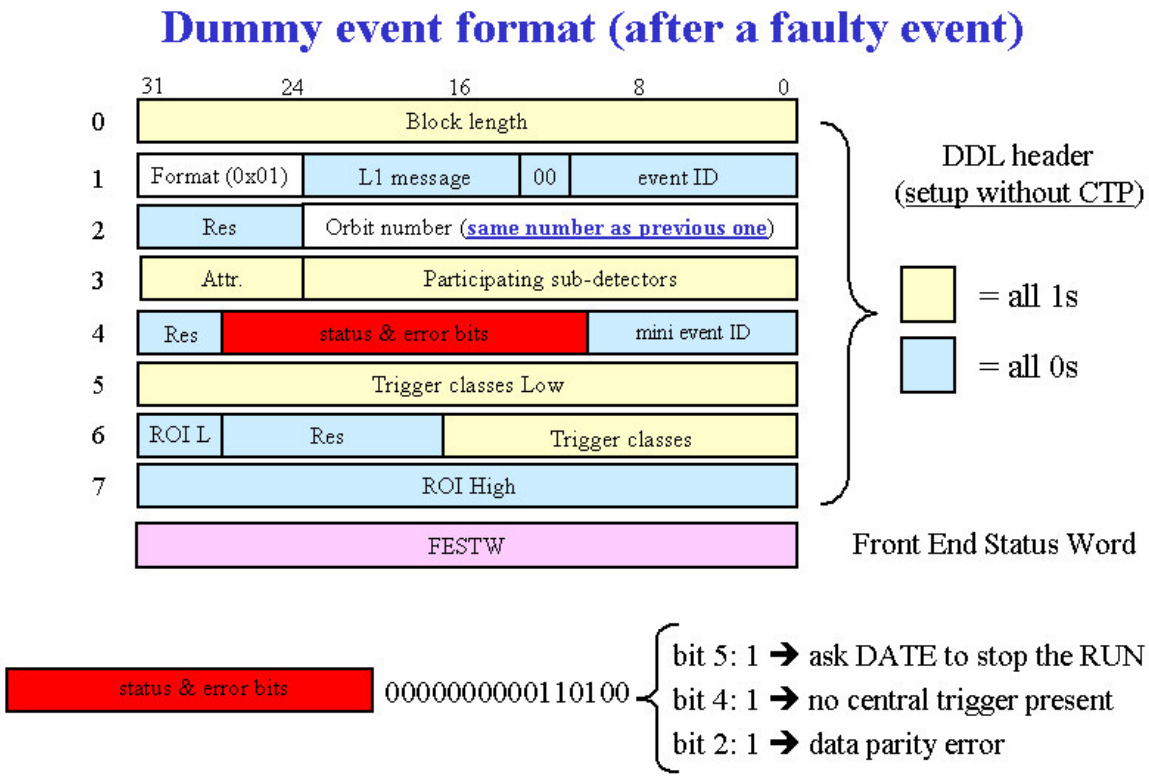


Fig. 8: CARLOSrx dummy event format

Software tool

A C++ software tool has been written in order to ease CARLOSrx debugging stage. The SW decodes data coming out from CARLOSrx and automatically compares the decoded data with the actual CARLOS outputs.