CARLOSrx v3 reference manual

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

INFN Bologna

May 2003

CARLOSrx v3

Outline

CARLOSrx v2 - CARLOSrx v3

CARLOSrx v2	CARLOSrx v3
CARLOSrx board successfully tested in June 2002 at CERN with the DDL (v2 since it interfaces CARLOS v2)	CARLOSrx board to be used in August 2003 test beam (v3 since it interfaces CARLOS v3)
Xilinx XC3195A	Xilinx XC2V1000
5V I/Os	2.5 V I/Os
able to process the data stream coming from one CARLOS only	able to process the data stream coming from up to 2 CARLOS chips
simple data packing	data coming from CARLOS are grouped in 32-bit words depending on their type (header, footer, data, JTAG word, error flag word). Then they are stored in a 20kx32 bit long FIFO
no interface towards the trigger system implemented	interface towards the trigger system implemented
interface towards the SIU implemented for what concerns "Event data transmission mode"	same as v2
no JTAG implemented	standard JTAG IEEE 1149.1 implemented
no custom JTAG instruction	2 custom JTAG instructions: "Put CARLOS in JTAG mode" and "Put CARLOS in RUN mode"
no serial backlink (CARLOS v2 had no serial backlink)	management of the serial backlink towards CARLOS
no flow control implemented	flow control towards the SIU implemented

<u>Main features</u>

- XC2V1000 Xilinx Virtex2 FPGA (see Fig. 1);
- 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 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

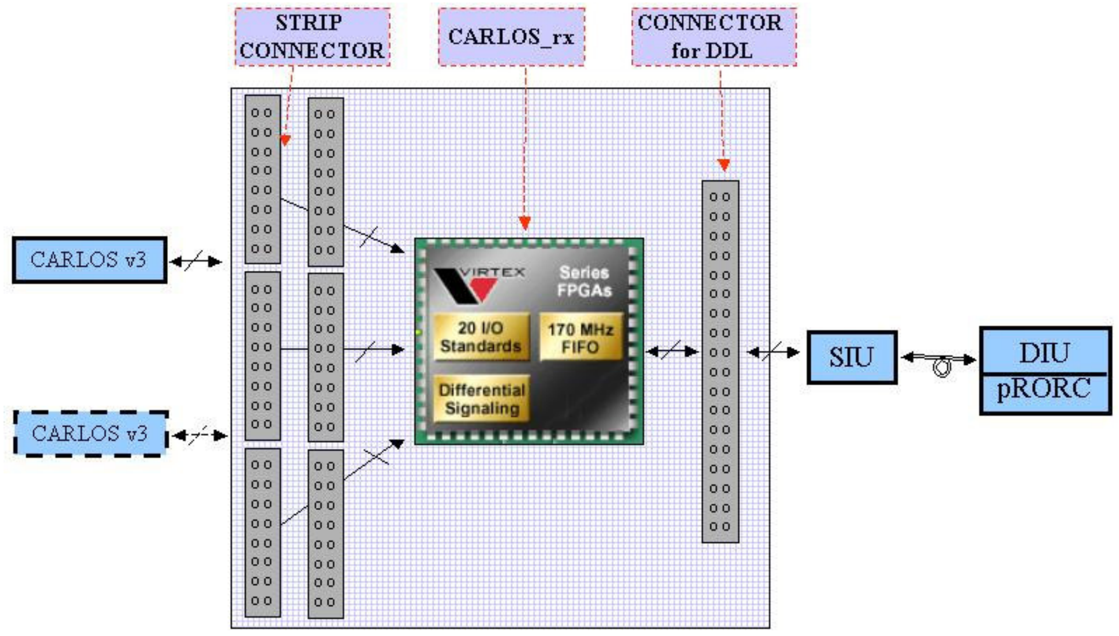


Fig. 1: CARLOSrx board schematic layout

<u>General description</u>

CARLOSrx v3 is a Xilinx Virtex2 FPGA-based device with the main purpose of concentrating data coming from up to two SDD detectors. Thanks to a large FIFO inside the FPGA, CARLOSrx (see Fig. 2) stores data coming from the front-end electronics and CARLOS through the optical links, packs data into 32-bit words and sends them towards the DDL system, after a transaction has been opened by the SIU. CARLOSrx also drives the JTAG ports for addressing and programming the front-end chips and CARLOS and the serial back-links used to send to CARLOS reset commands, trigger signals and control commands.
It also interfaces the CTP (Central Trigger Processor) with the busy signal and the TTCrx device from which it receives the L1accept signal and the trigger information.

<u>During the SDD August 2003 beam test</u>, CARLOSrx will be used in the configuration reported in Fig. 3. These are the main simplifications that have been introduced in order to ease the data acquisition process:

- CARLOSrx acquires data coming from 1 detector (= 1 CARLOS only);
- CARLOSrx is directly connected to CARLOS (the optical link is omitted);
- CARLOSrx interfaces to a trigger system with the busy signal (it does not interface to the TTCrx device).

In order to comply with this data acquisition chain setup, a specific VHDL design has been implemented on the Virtex FPGA. All the information (text and simulation waves) contained in this datasheet from this point on are related to the current version of the design implemented on the FPGA and to the chain reported in Fig. 3.
This design contains 5 logic blocks (see Fig. 4):

1. *<u>data packing</u>*: 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. *<u>SIU interface</u>*: 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. *<u>trigger interface</u>*: 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 is received from the CARLOS error flag words.
4. *<u>JTAG interface to SIU</u>*: this block receives the JTAG signals from the SIU and directly forwards them 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 right command.
5. *<u>serial backlink</u>*: this block drives the *serial backlink* signal from CARLOSrx to CARLOS. It is used to send reset commands, trigger signals and the commands "Enter JTAG mode" and "Enter RUN mode".

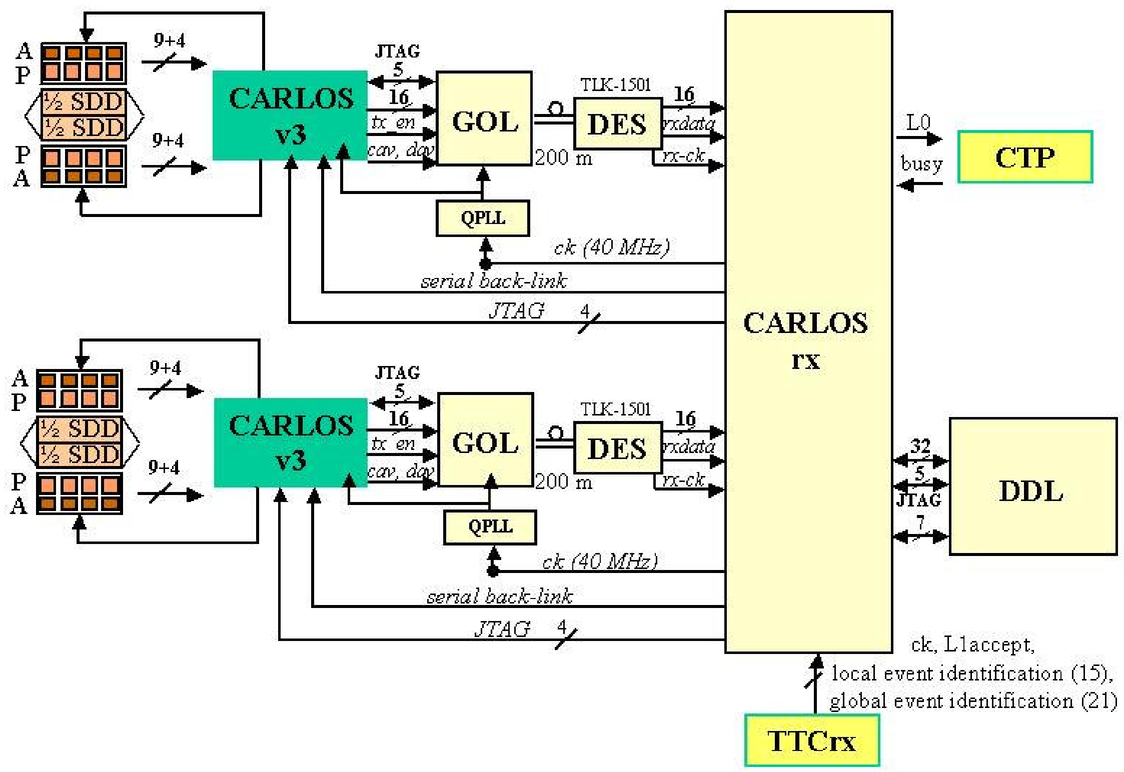


Fig. 2: Data acquisition chain for CARLOSrx

JTAG instruction set

CARLOSrx receives a JTAG bus from the SIU and directly forwards it towards CARLOS as it is. 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

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.

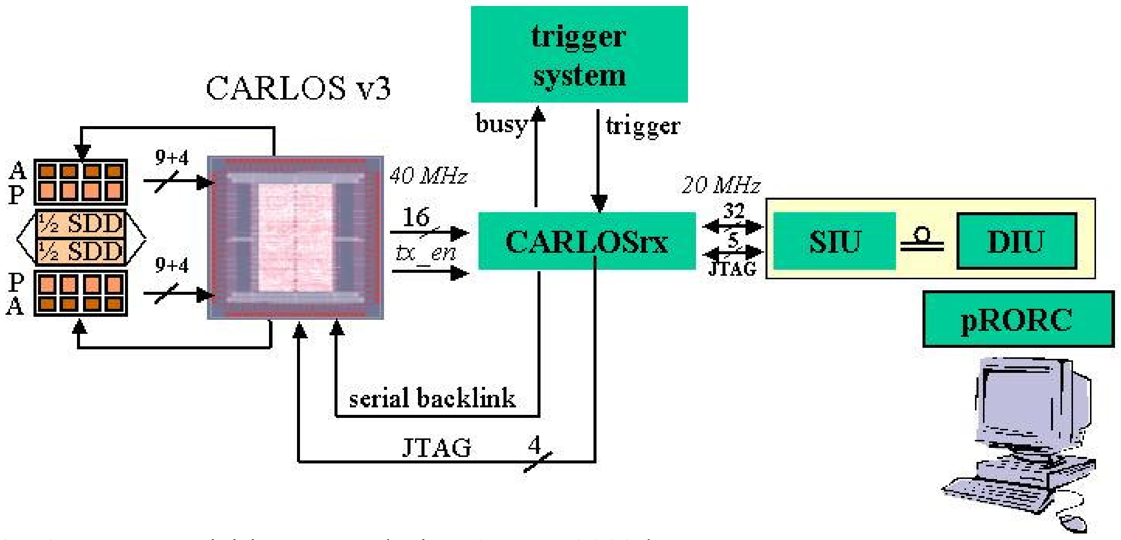


Fig. 3: Data acquisition setup during August 2003 beam test

CARLOSrx v3 operation

This section contains an explanation of the sequence of actions needed to program and run CARLOSrx operationally. Besides that a description of CARLOSrx main features is reported.

Using CARLOSrx

CARLOSrx utilization should include the following sequence of actions:

1) power supply 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 chip 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 forwards the JTAG port received from the SIU to CARLOS as it is. After

the last front end 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. <u>All the JTAG information read from the front-end chips and from CARLOS is stored in the internal FIFO of CARLOSrx</u>. 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 CARLOSrx receives a trigger, it sends the related command to CARLOS using the serial backlink. Then CARLOS begins waiting for the data packets coming from CARLOS.
7) Valid words coming from CARLOS are grouped into 32-bit words depending on their meaning:
   - header words;
   - footer words;
   - data from channel 1;
   - data from channel 0;
   - error flag words;
   - JTAG words.

   The MSBs are used for univoquely identifying the words when the data packet shall be decoded. <u>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</u>.

   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.

8) 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.
9) 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).
10) 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.
11) 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), 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.

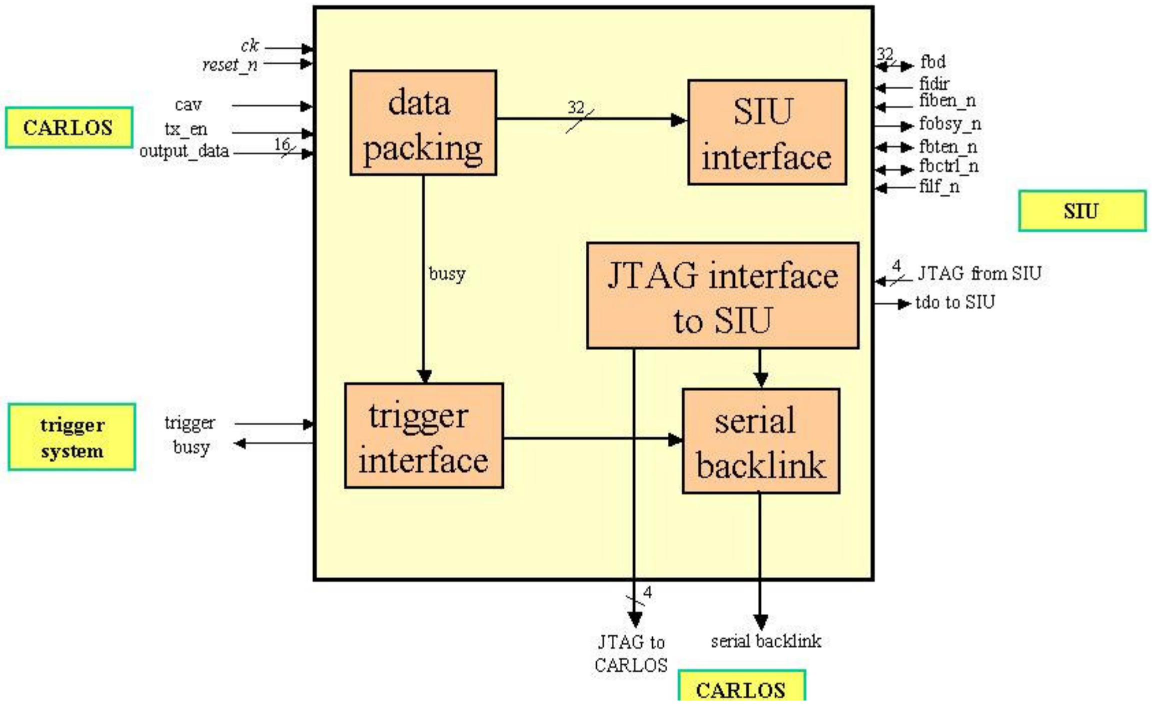
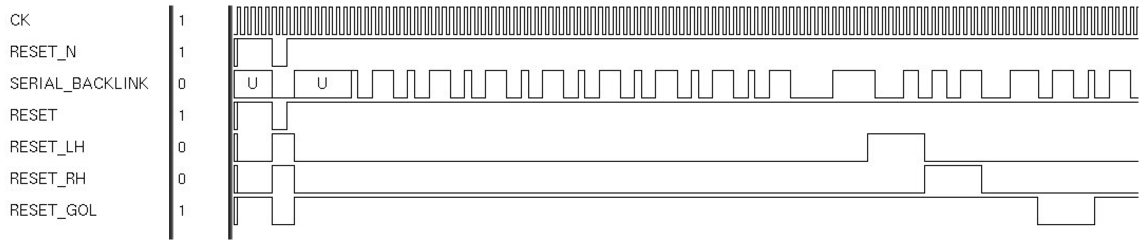
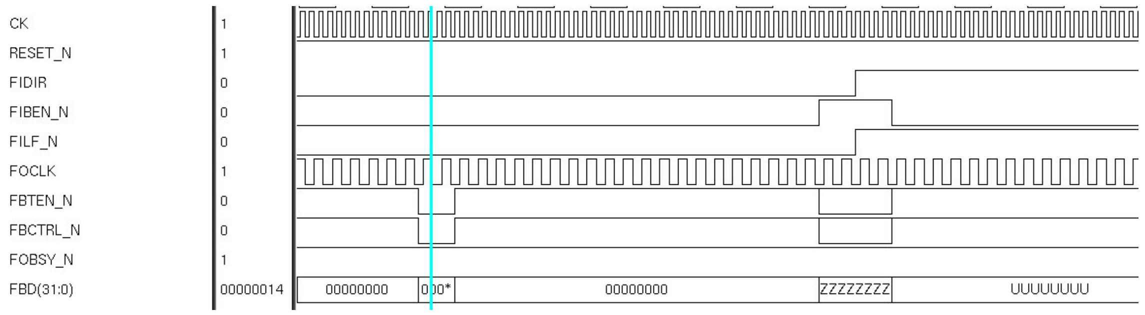


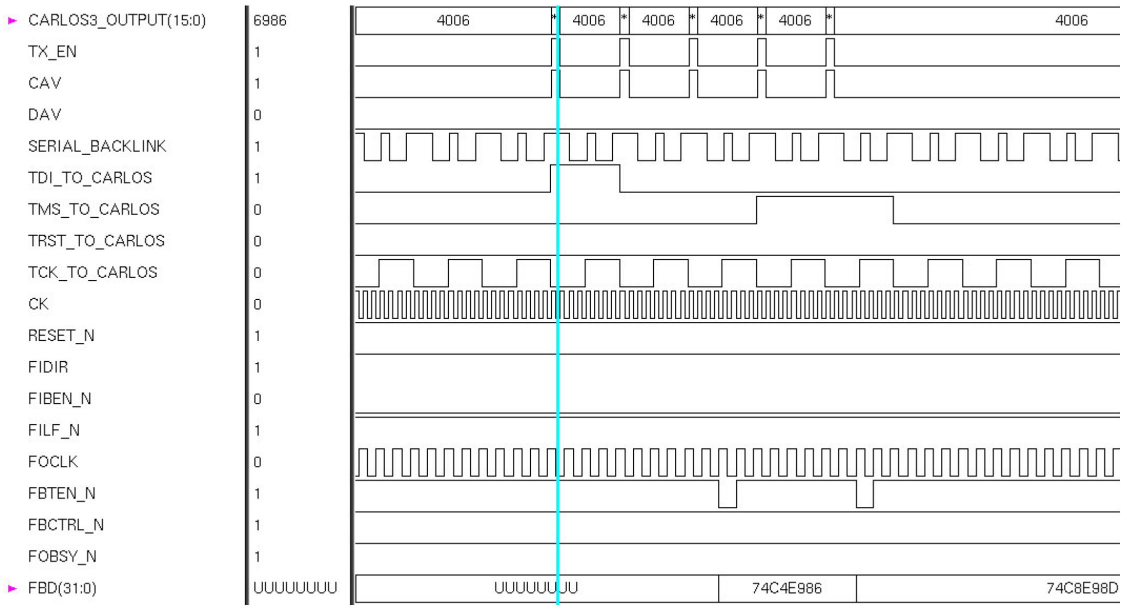
Fig. 4: Schematic blocks of CARLOSrx



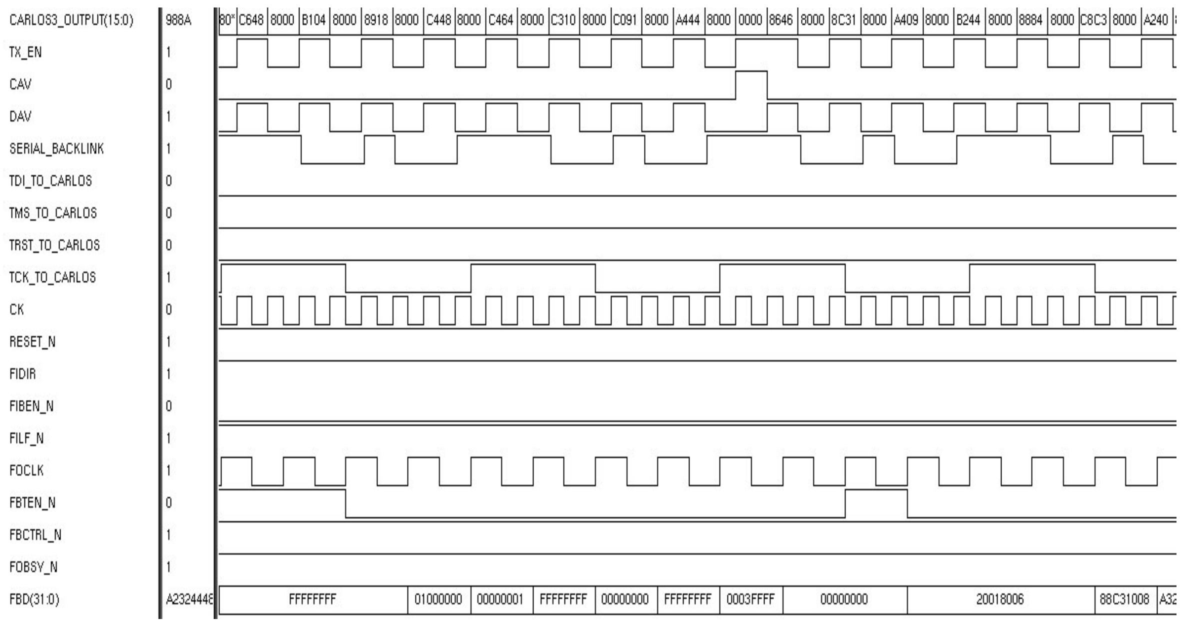
Timing 1: CARLOSrx receives the reset signal (*reset_n*). Then it sends IDLE commands on the *serial backlink*, followed by the reset commands for the front-end chips.



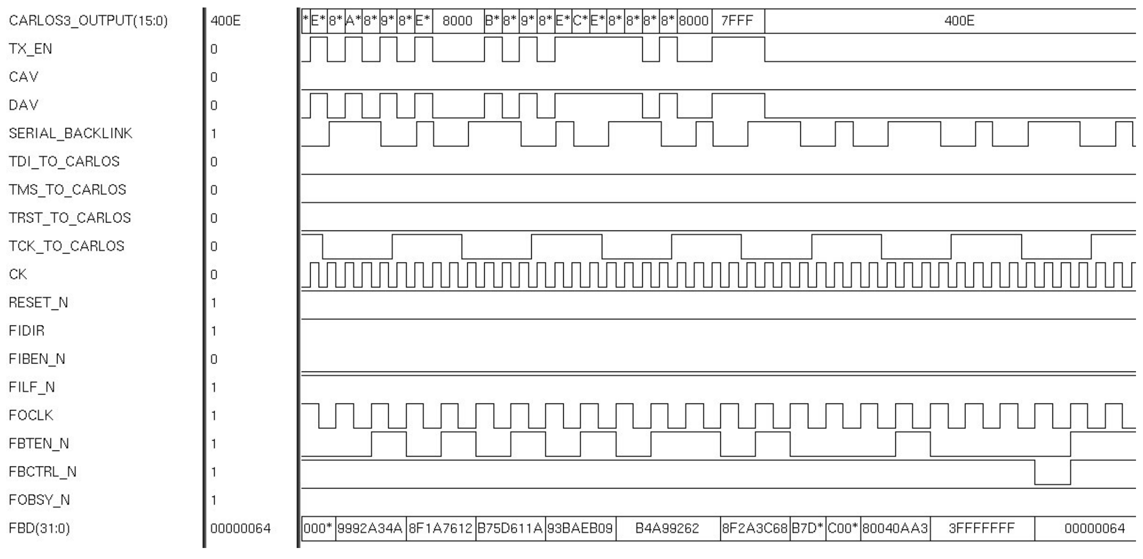
Timing 2: CARLOSrx receives the RDYRX command on the *fbd* bus, then it waits until the SIU changes the *fidir* value. Then CARLOSrx takes possession of the bidirectional bus.



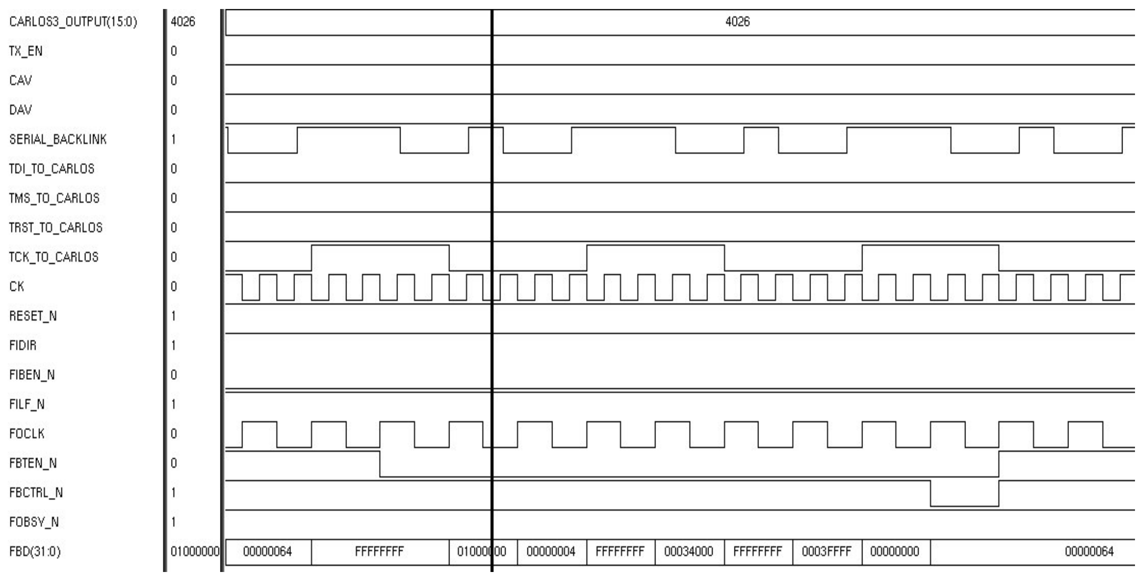
Timing 3: CARLOSrx packs 16-bit long JTAG words coming from CARLOS into 32-bit long words and sends them to the SIU on the *fbd* bus.



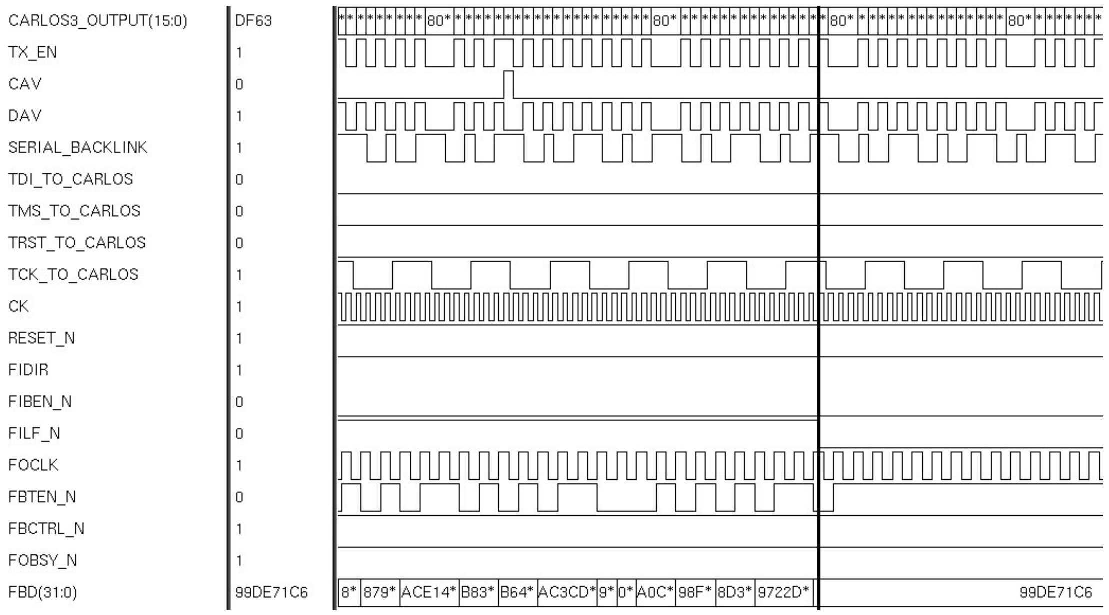
Timing 4: CARLOSrx packs into 32-bit long words the data packet coming from CARLOS. Each data packet from CARLOSrx to the SIU begins with the DDL header (8 32-bits words) followed by 3 CARLOS header words.



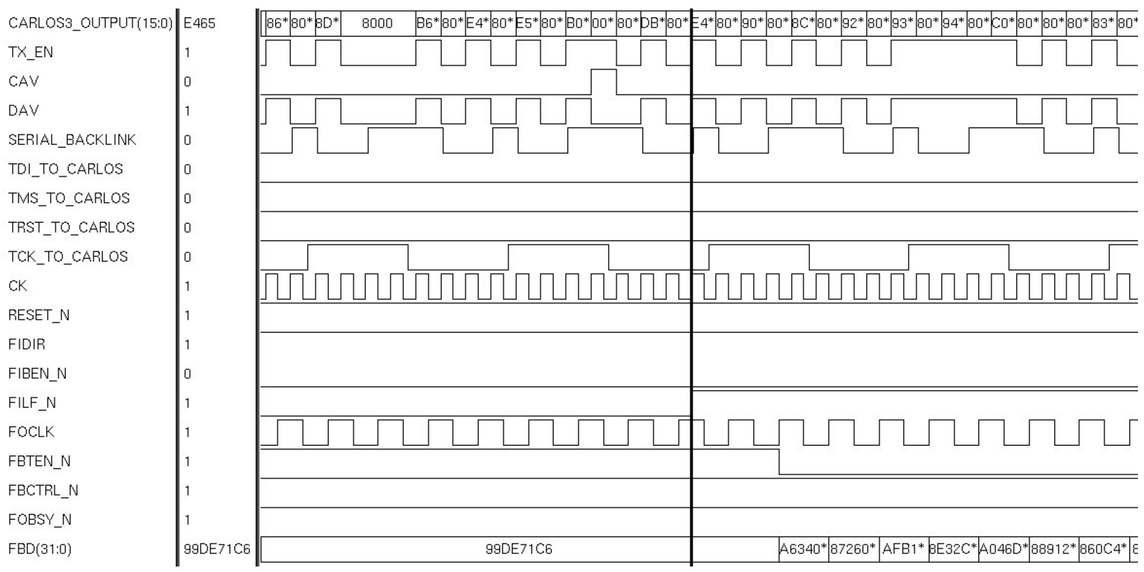
Timing 5: CARLOSrx closes a data packet with 3 footer words, then it sends the FESTW to the SIU.



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.

Interface to CARLOS

CARLOSrx is directly interfaced to CARLOS v3. These are the interface signals:

- *output_data* (16 bits): this is the 16-bit bus containing the data coming from CARLOS v3;
- *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.
- *JTAG* (4 bits): it is a standard JTAG port that CARLOSrx receives from the SIU and forwards directly towards CARLOS as it is. This is used for the front-end chips addressing, programming and reading back. <u>CARLOS expects to receive a 5 MHz JTAG clock.</u>
- *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 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). Then it continues sending IDLE codes until it has to send one of the 3 following commands:

- enter JTAG mode;
- enter RUN mode;
- trigger signal.

Interface to the SIU

A list follows of the signals involved in the CARLOSrx- SIU interface (see Fig. 5):

- *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.

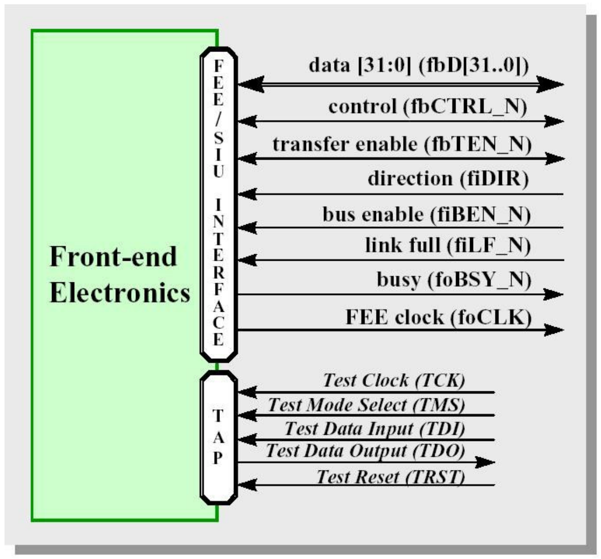


Fig. 5: CARLOSrx – SIU interface

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 one clock period long.
- *busy*: it is the busy signal from CARLOSrx to the trigger system. When CARLOS is in JTAG mode, CARLOSrx asserts the *busy* signal high. When CARLOS is in RUN mode the busy value is the one received from CARLOS in the error flag word (one every 64 clock cycles).

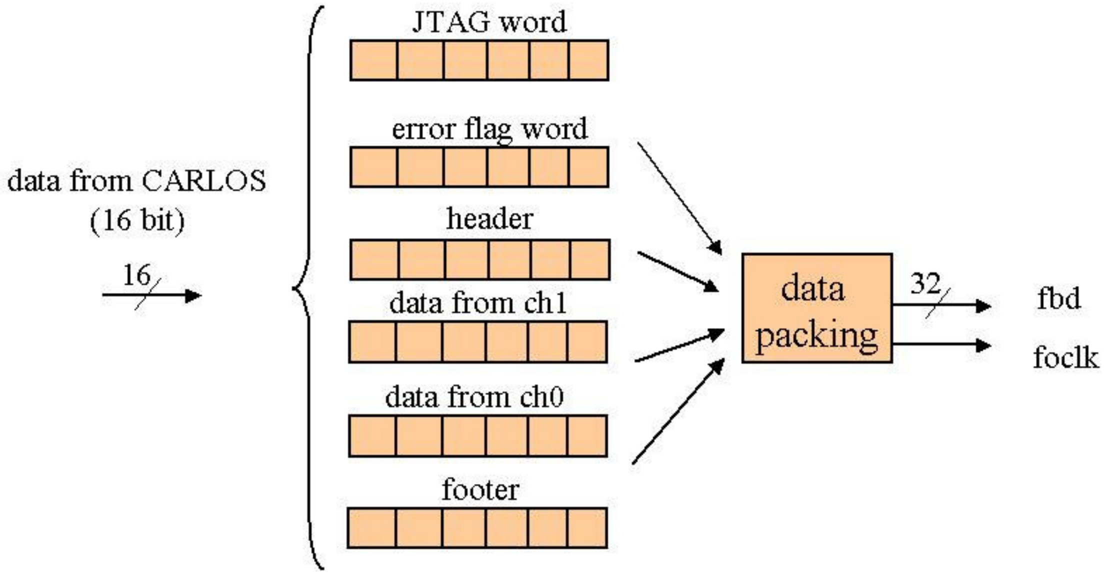


Fig. 6: Data processing on CARLOSrx

<u>Data storing and processing</u>

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. <u>Error flag words coming from CARLOSrx are stored only during the timing frame of a data event: error flag words arriving outside these frames are discarded</u>. 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. 7). Then data are popped from the FIFO synchronously with *foclk*, that is half the system frequency.
It may also happen that one of the data channels from CARLOS v3 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.

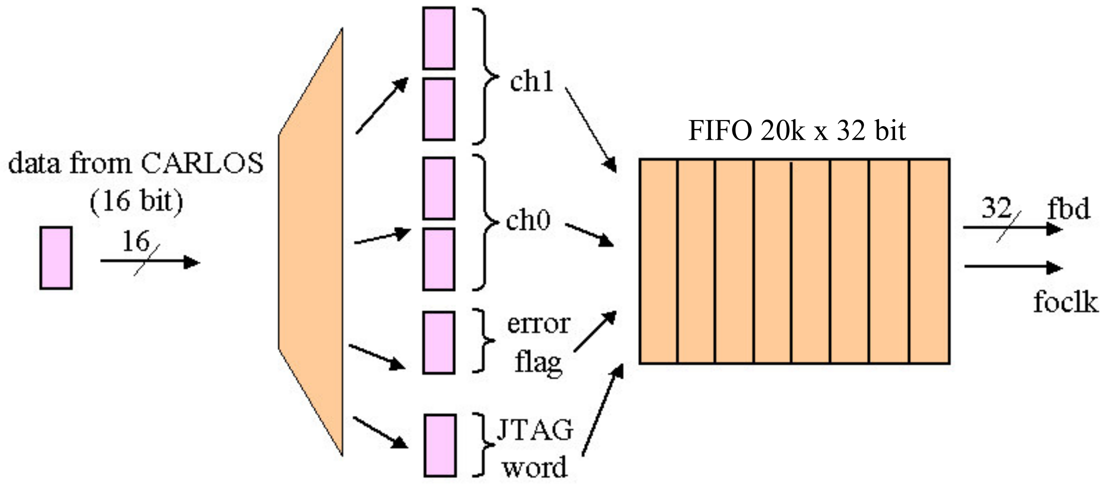


Fig. 7: Data packing and storing on CARLOSrx

<u>Data transmission protocol</u>

After the transaction has been opened, CARLOSrx begins sending data towards the SIU in the same order as the words received from CARLOS. So far during JTAG programming, CARLOSrx sends to the SIU the 32-bit JTAG words, 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. 8). The DDL header is required by the DATE v4 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 during the beam test the CTP (Central trigger Processor) is not used, most of these fields are not used and, for this reason, they are either put to 0 or put to 1 as required by DATE v4.
The only meaningful information fields are:

- **<u>format version</u>**: 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. 9) 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.

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.

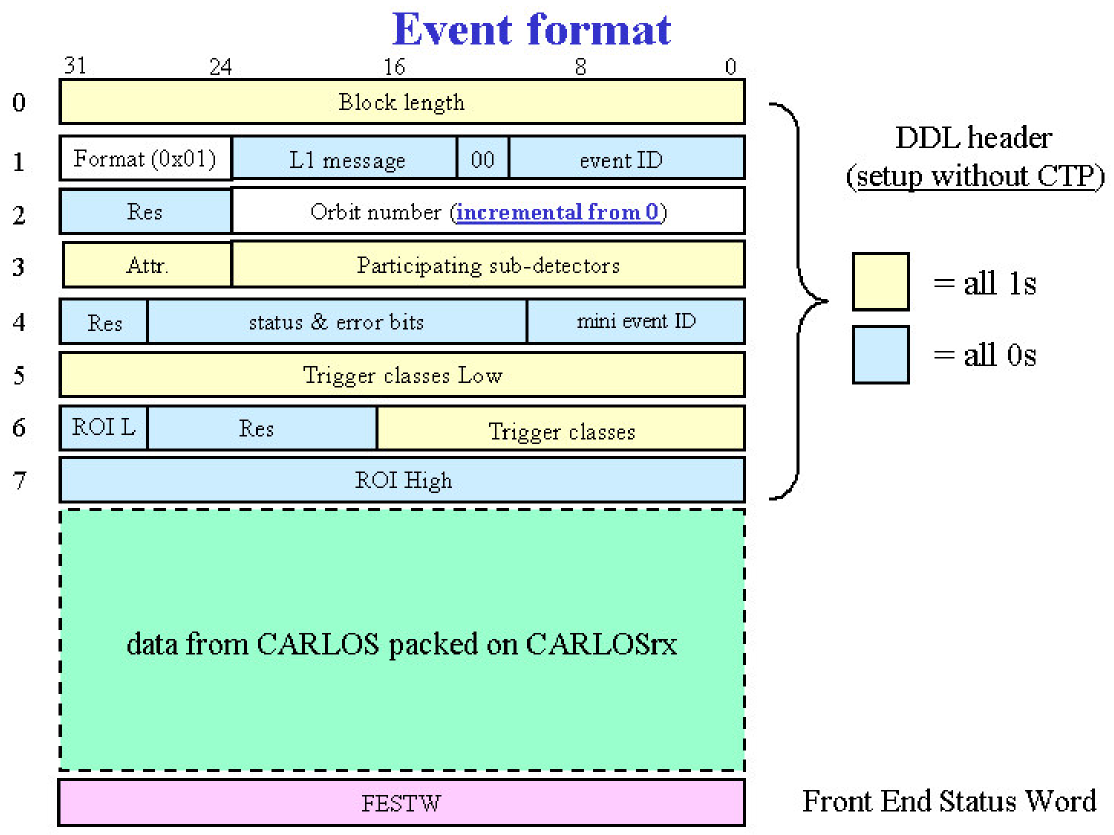


Fig. 8: CARLOSrx event data format

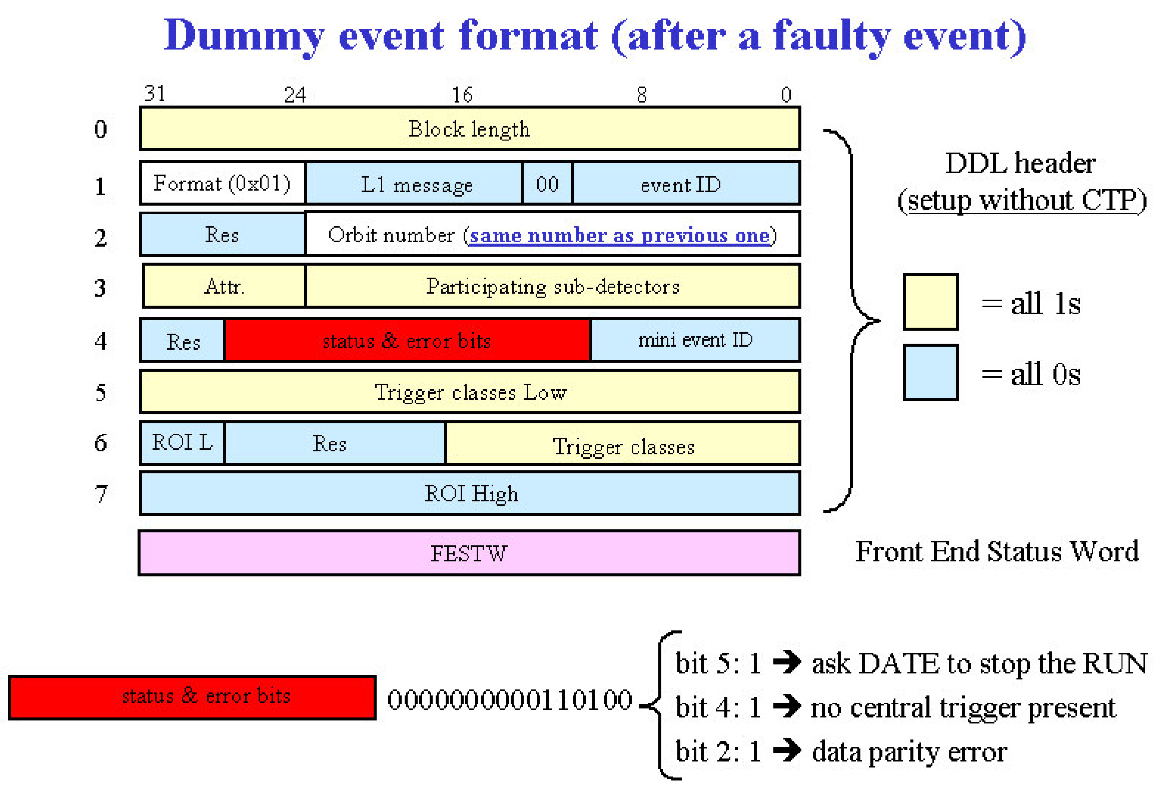


Fig. 9: CARLOSrx dummy event format

Integration steps for CARLOSrx

We foresee 3 steps for the integration of the CARLOSrx device into the SDD data acquisition chain:

- **STEP 1:** A chain containing CARLOS v3, CARLOSrx and the SIMU (SIU Simulator) is tested. A Tektronix pattern generator is used for sending data to CARLOS from the AMBRA side, for emulating the trigger system and for driving the JTAG port towards CARLOSrx (see Fig. 10). The SIMU device is tested in the configuration "Event data transmission".
- **STEP 2:** Starting from the chain realized at Step 1, the SIMU device is replaced by the DDL, while the JTAG port is still driven by the Pattern Generator (see Fig. 11). This step is performed at CERN in conjunction with the DAQ group.
- **STEP 3:** The JTAG port is no longer driven by the Tektronix pattern generator, but by the DDL itself (see Fig. 12).
- **STEP 4:** CARLOS input data buses are no longer driven by the Pattern Generator, but by 4 AMBRA chips per channel (see Fig. 13).
- **STEP 5:** CARLOSrx directly interfaces the trigger system. The Pattern Generator is no longer used. The data acquisition chain is complete (as from Fig. 3).

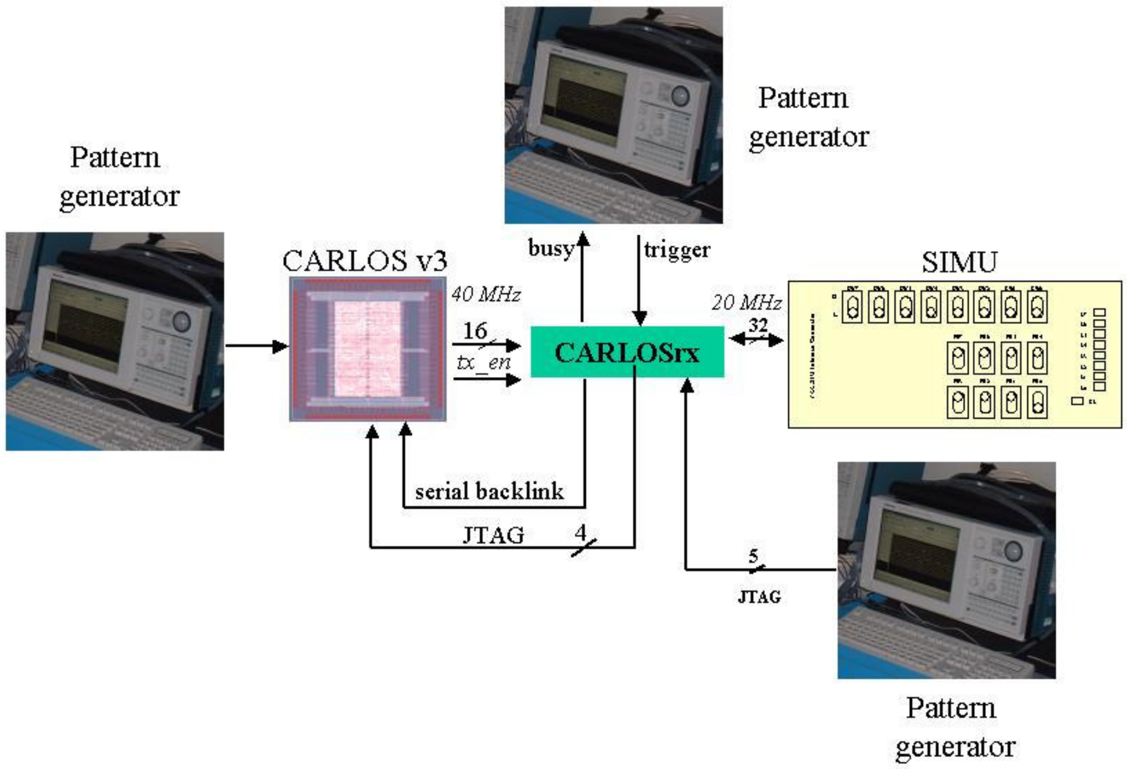


Fig. 10: CARLOSrx integration process: **STEP 1**

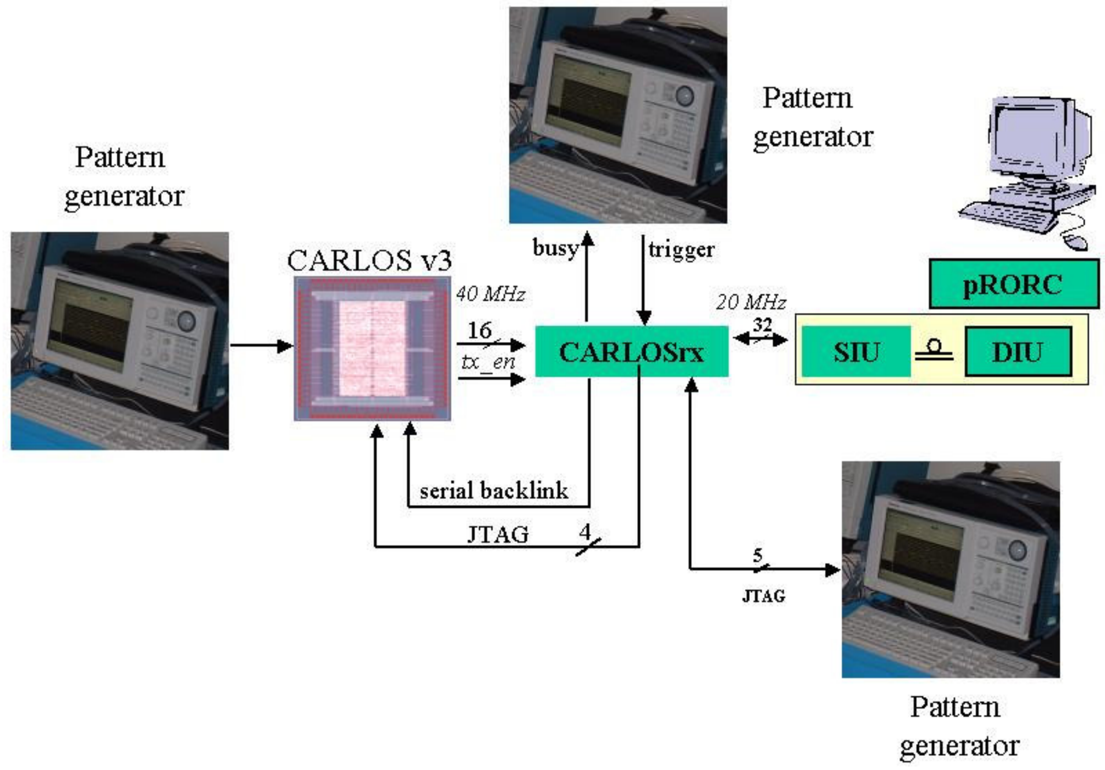


Fig. 11: CARLOSrx integration process: **STEP 2**

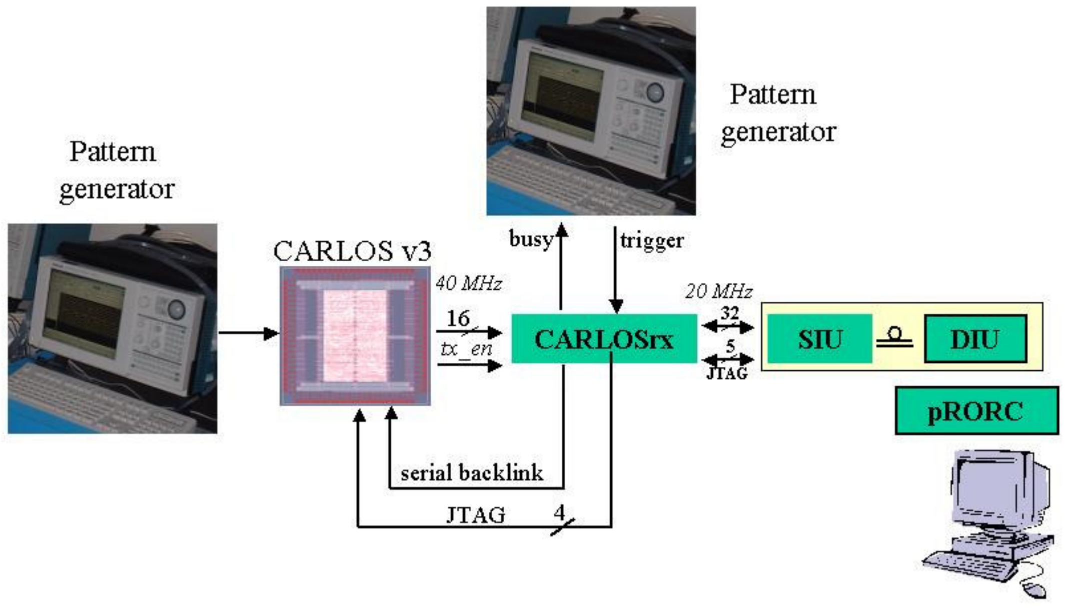


Fig. 12: CARLOSrx integration process: **STEP 3**

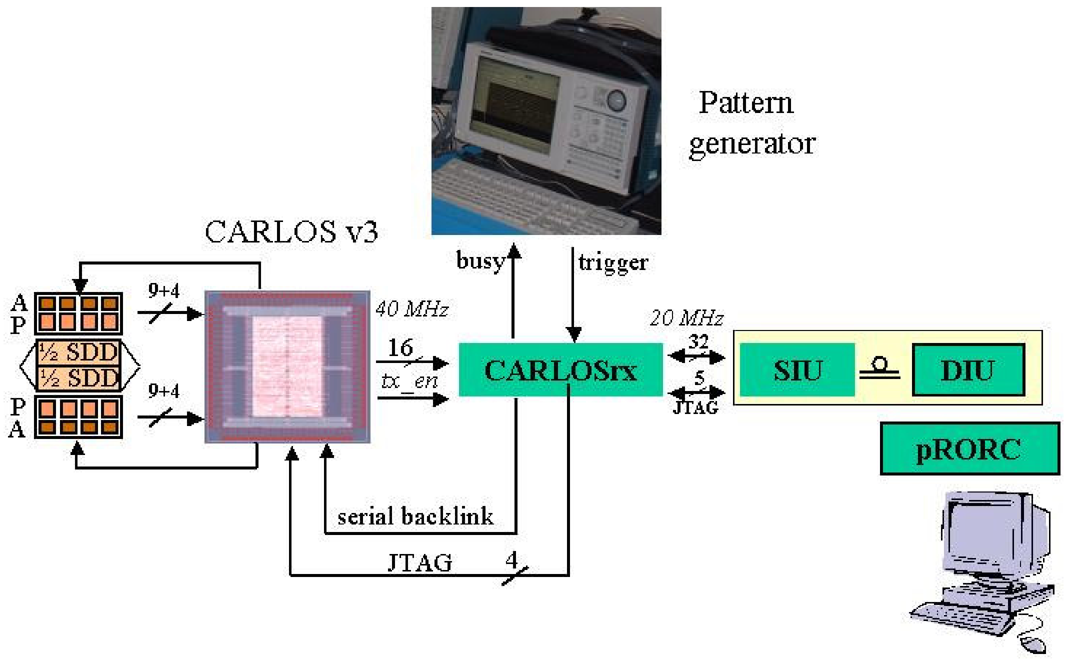


Fig. 13: CARLOSrx integration process: **STEP 4**