CARLOS v4 reference manual

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

Department of Physics and INFN Bologna

December 2003

Outline

What's new in CARLOS v4

- The JTAG port has been encapsulated on the serial back-link input port.
- The *test_output* bus has been extended from 4 to 8 bits.
- All the I/O pins, with the exception of the clock signals (*ck* and *ck_out*), the reset signal and the serial back-link signal, have been put on the boundary scan chain.
- 7 new serial back-link commands (Prepulse 50, 75, 100, 125, 150, 175, 200) have been added for the generation of a *prepulse* output signal with a variable width from 2 to 8 clock periods.
- 2 new serial back-link commands (Stop Acq and Restart Acq) have been added in order to implement the CARLOSrx backpressure over CARLOS.
- A new 1-bit JTAG programmable register (Stop If Error) used to decide whether to stop or not the data transmission process after a synchronization error between AMBRA and CARLOS has been added (this occurs when CARLOS does not sample the *data_end* input during the expected time slot).
- A 128x1 FIFO buffer (*fifo_jtag*) is used to store the *tdo* values before sending out JTAG words on the 16-bit data bus. When programming the GOL chip via JTAG, CARLOS samples its *tdo* output and stores these values into the *fifo_jtag* buffer until the JTAG connection is closed.
- The *fifo_data* buffer has been increased from 16x30 bits to 24x30 bits.
- The BIST feature has been upgraded from 200 to 400 pseudo-random test vectors per CARLOS macro-channel.
- The data encoding table used when the 2D compression is disabled has been modified.
- The 3-bits put in End of Row Summaries reporting parity errors in AMBRA and CARLOS have been encoded in a different way. In particular the parity errors are reset after processing the last sample of each incoming anode data.
- The anode length range has been extended to 0-255 (meaning that up to 256 samples per anode can be processed).

CARLOS v4 main features

- 4x4 mm^2, 100 pins, CERN 0.25 μm CMOS technology.
- double threshold 2D compression on input data; if desired, compression may be disabled and, in this case, a simple 1D encoding is performed on incoming data.
- 40 MHz target working frequency.
- interface to GOL implemented using both Ethernet and G-Link protocols.
- standard IEEE 1149.1 JTAG implemented.
- BIST implemented using 400 pseudo-random test vectors.
- the JTAG port input values are decoded and delivered either towards CARLOS itself, or towards the left or right hybrid (a board containing 4 PASCAL-AMBRA pairs used to acquire data coming from a half-detector SDD), or towards the GOL chip.
- JTAG answer *tdo* and output data share the same 16-bit data channel towards the GOL chip.
- JTAG mode and RUN mode are mutually exclusive in time: it means that when in JTAG mode, the normal behavior of the chip is frozen and vice-versa. CARLOS working mode is determined by the commands "Enter JTAG mode" and "Enter RUN mode" that CARLOSrx sends to CARLOS via the serial back-link.
- adoption of measures against radiation effects: parity errors on RAM words and on JTAG configurable registers are asserted in the error flag words put in output. The same holds also for parity errors detected on AMBRA. Besides that parity errors are also notified in the End Of Row Summaries.

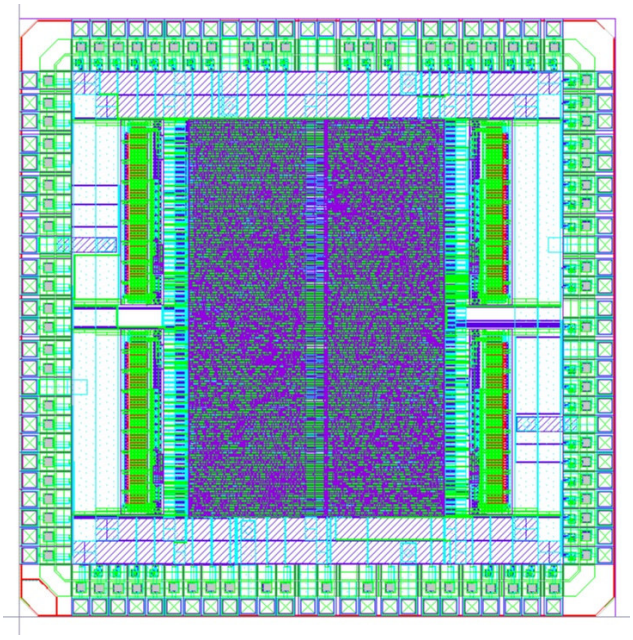


Fig. 1: CARLOS v4 final layout

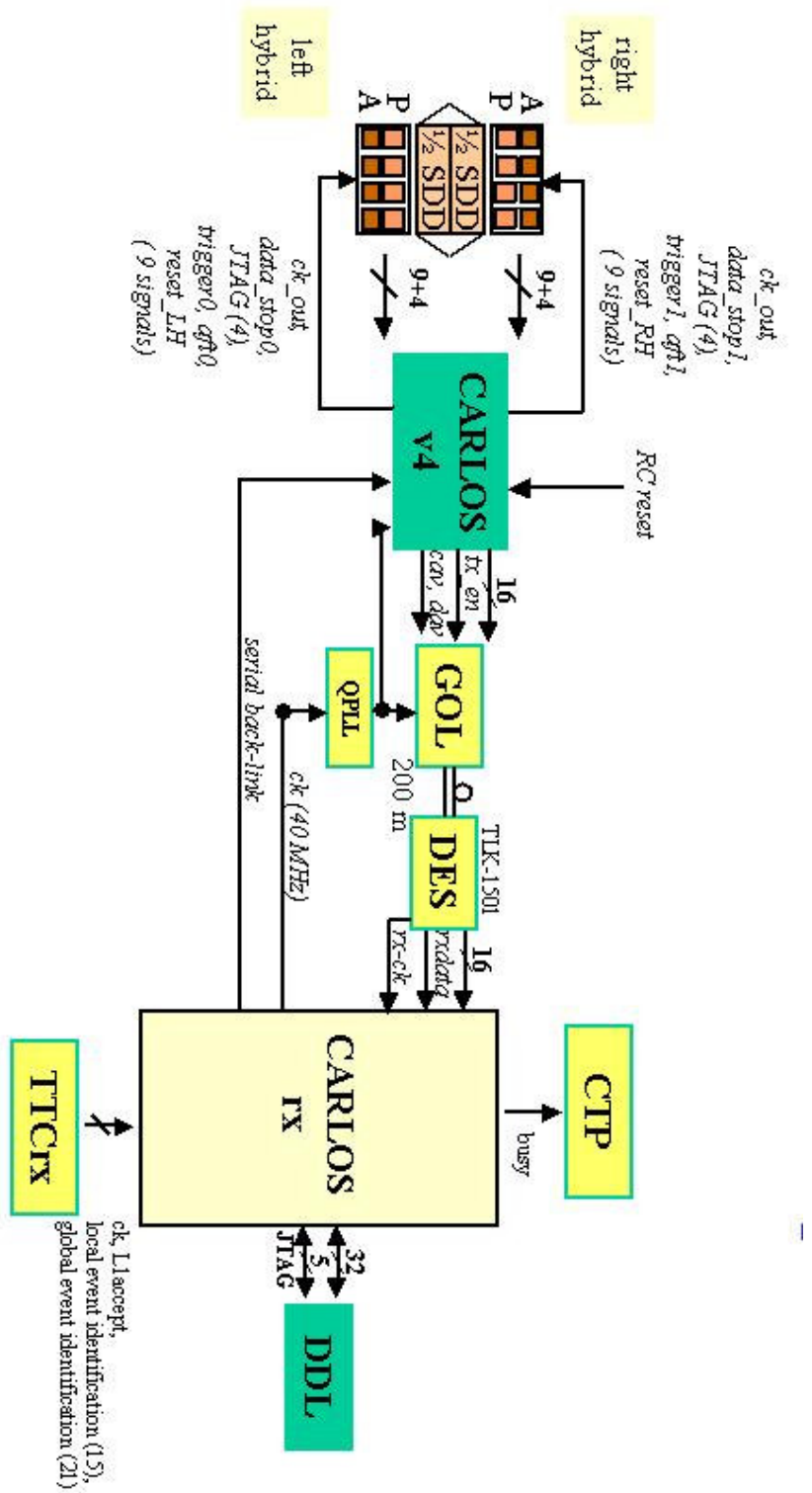


Fig. 2: SDD readout chain

General chip description

SDD readout chain

CARLOS v4 is an ASIC belonging to the SDD readout chain, as shown in Fig. 2. The chip is directly interfaced to different devices:

- 2 hybrids: an hybrid is a board containing 4 PASCAL – AMBRA pairs, being the front-end electronics acquiring data coming from a half-detector;
- GOL: a rad-hard serializer chip sending data to CARLOSrx by means of a 800 Mbit/s optical link. The GOL needs to receive a very low-jitter clock (max 100 ps peak to peak): such a clock is sent to GOL using the QPLL device.
- CARLOSrx: an FPGA-based device put in counting room with the purpose of collecting data coming from different SDD chains and sending them towards the DDL system. Besides that it remotely controls CARLOS using a 40 MHz serial back-link for what concerns CARLOS working mode and JTAG programming. It also manages the interface between CARLOS and the CTP system and the TTCrx device, transferring trigger signals from the TTCrx to CARLOS on the serial back-link and transferring the busy signal from CARLOS towards the CTP. For more details on CARLOSrx see the document *carlosrx_datasheet.pdf* available at the following Web site: http://www.bo.infn.it/~falchier/alice.html
  This document refers to the CARLOSrx device used at the August 2003 CERN test beam.

CARLOS architecture

The purpose of CARLOS v4 is to perform an on-line 2D compression on two incoming 8-bit data streams from the two hybrids, one for each half-detector. Compressed data are then packed in 30-bit long words using a barrel shifter, temporarily stored in a 24x30 flip-flop based FIFO and then multiplexed on a single 16-bit bus output towards the serializer GOL chip (see CARLOS architecture in Fig. 3). The two data channels are processed in parallel and, in each data processing channel, compression, packing and storing steps are performed as successive stages of a pipeline with a clock running at 40 MHz.
Compression performances are completely tunable using a set of registers that can be programmed using a standard JTAG access port. Using this port, 2D compression itself can be switched on or off and threshold levels can be modified by writing and reading internal configuration registers only when CARLOS is in JTAG working mode. Beside that, internal registers are protected against the radiation effects using a parity mechanism: each time a register is accessed, parity is computed and compared with the expected value. In case a mismatch is found, it is reported in the End Of Row Summary at the end of each processed anode. In case a severe parity violation if

found, an error flag is switched on in the output data sent to CARLOSrx through the optical link, so to allow CARLOSrx to program CARLOS internal registers again the sooner the possible. The CARLOSrx device remotely controls CARLOS v4 via a synchronous serial back-link.
CARLOS provides both clock signal and the JTAG signals to the left and right hybrids. The *ck_out* output is directly connected to the *ck* input: it is used to provide the clock to AMBRA and PASCAL.
For what concerns JTAG, CARLOS also acts as a JTAG switch providing 3 4-bits JTAG ports in output: one for the left hybrid, one for the right hybrid and one for the GOL chip. In other words CARLOS receives the JTAG port signals encapsulated over the serial back-link and provides in output 3 JTAG ports, thus allowing to open a JTAG connection towards different devices.
The mechanism is the following one: after receiving the *trst* signal on the serial back-link channel, CARLOS begins waiting for a 7-bit address (encoded with redundancy and parity protected) containing the information of which is the device to be addressed via JTAG. After decoding the address, JTAG information can be switched to CARLOS itself, to the right hybrid, to the left hybrid or to the GOL chip, so far providing JTAG connectivity towards PASCAL, AMBRA, CARLOS and GOL chips. The JTAG connection can then be closed by asserting the *trst* signal again over the serial back-link channel or by resetting CARLOS (beside closing the JTAG connection a reset action also resets all the internal register, so it is neither a good practice nor useful to close a JTAG connection in this way).
CARLOS v4 can work in two distinct and non-overlapping in time working modes, JTAG mode and RUN mode. After a reset action CARLOS is put in JTAG mode, then it can be put in RUN mode after receiving the command "Enter RUN mode" on the serial back-link: in the same way when in RUN mode, CARLOS is put in JTAG mode when receiving the command "Enter JTAG mode".

- When in JTAG mode, CARLOS can process neither any input data from AMBRA (*data_stop1* and *data_stop0* outputs are kept high) nor any trigger input command (when CARLOS is in JTAG mode, CARLOSrx itself has to provide a *busy* = 1 towards the CTP system): only reset and JTAG commands are processed. After CARLOSrx opens a JTAG connection towards one of the hybrids or the GOL, the selected device begins sending the JTAG answer on their output pins *tdo* (from CARLOS point of view: *tdo_from_LH*, *tdo_from_RH*, *tdo_from_GOL*). CARLOS samples the *tdo* values coming from the selected device when they are valid and sends them towards CARLOSrx using the 16-bit output data bus, the same used for sending data and error flags words when in RUN mode.
  After CARLOSrx opens a JTAG connection towards CARLOS itself, CARLOS internal JTAG answer *tdo* is sampled and put in output over the 16-bit output bus. In each case the *tdo* value (either coming from outside or computed internally on CARLOS) is sampled only in the JTAG standard states **Shift-IR** and **Shift-DR**. In the former case the *tdo* output is 1000…0 serially shifted out

(the code length is the same as the number of bits of the JTAG Instruction Register). In the latter case the *tdo* output contains the value of the selected internal register serially shifted out (so far the old register content before being written again).
When a broadcast JTAG operation is performed (more than one chip JTAG-addressed at the same time), no *tdo* value is sampled by CARLOS (CARLOS does not receive any information on whether the operation has been successful or not), in fact in this case more than one chip would be driving CARLOS *tdo* input signal, resulting in a bus contention.
In case CARLOSrx tries to open a JTAG connection towards a device without success, it will not receive the expected answer on the JTAG word, so that it will try to open the JTAG connection again, after a time-out period. No error code concerning the impossibility of opening a JTAG connection has been foreseen.

- When in RUN mode, CARLOS processes data coming from AMBRA and trigger commands coming from CARLOSrx over the serial back-link channel. Data words sent towards the GOL chip contain both processed data and error flag words encapsulated in a protocol compliant both with G-Link and Ethernet standard modes.

CARLOS v4 also hosts a BIST (Built In Self Test) facility in order to ease the chip test and selection: after running the JTAG RUNBIST command, 400 test vectors are fed into the 2 processing channels. Data coming out from the multiplexer block *Outmux* are then analyzed by a *Signature Maker* block performing a deterministic function of its inputs and producing a 16-bit code strictly dependent on its input data. This code can then be read using the standard JTAG port in order to check whether the chip has passed the test or not. It is a pass / no pass test: it gives no indication of where the failure happens, if any.
An other test facility hosted on CARLOS is a multiplexer bringing in output the values of key internal nets, such as the output of the compressor block or the barrel shifter, in order to ease the chip debugging phase. The multiplexer selection lines are directly driven by the 3-bit input bus *set_test*.

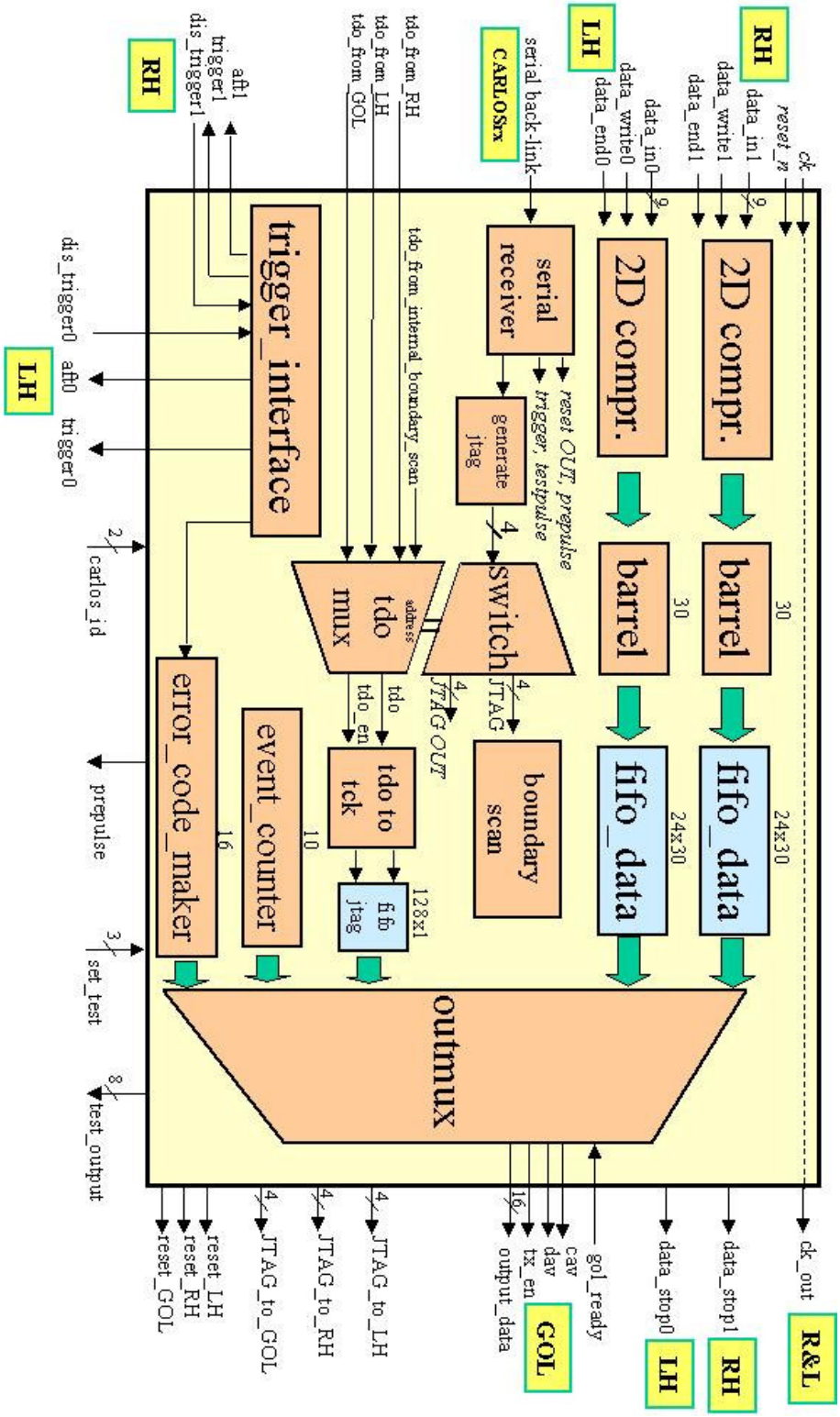


Fig. 3: CARLOS v4 schematic blocks

Main processing unit

The main processing unit of CARLOS v4 is composed by 3 main logic blocks (see Fig. 3):

- 2D compressor
- barrel shifter
- fifo_data

The 2D compressor implements a bi-dimensional double threshold compression algorithm applied to the incoming data stream. Its purpose is to find and save data clusters, while rejecting all noise and not interesting information. It can be disabled by programming the internal register *enable 2D* with the value 0. In this case a simple encoding is performed on input data.
The barrel shifter block has the purpose of packing the valid bits of the compressor output bus into a 30-bit register containing only valid data. The barrel block hosts an internal 60-bit long register: as soon as the 30 LSBs are ready to be put in output, the 30 MSBs are ready to accept data coming as a new input. As the last output data of an event coming from the compressor arrives, the valid bits contained in the 60-bit register are put in output padding the remaining bits with 0 when necessary.
The fifo_data block is a flip-flop based FIFO used to buffer compressed and packed data before they are multiplexed into the 16-bit output. The FIFO contains 24 30-bit words. The FIFO is asymmetric for what concerns the I/O buses: in fact the input bus is 30-bit long and the output bus is 15-bit long. This leads to the fact that one push action is balanced by two pop actions. The FIFO contains several output flags showing the FIFO status: *empty*, *almost_empty*, *half_full*, *almost_full* and *full*.
The *almost_full* flag is asserted when 20 out of 24 memory locations have been written, while the *half_full* signal is asserted when 12 out 24 locations of the FIFO are in use. Starting from these two signals, an hysteresis flag has been created: it is asserted when the FIFO gets almost full and is put back to 0 when at least half of the FIFO locations have been freed. The hysteresis flag is one of the signals that have been put in OR with each others in order to create the *data_stop* output. Having a FIFO with hysteresis is very important since it allows to have a smaller number of transitions on the *data_stop* signal with respect, for example, to the use of the *almost_full* signal for creating *data_stop*. This results in a higher reliability of the complete system. Thanks to the mechanism of the hysteresis flag the FIFO should never get completely full and then overflow. The hysteresis flag is also sent in input to the compressor block in order to deal with this particular situation: at the end of an event fetch, the 2D compressor sends data contained in the RAM one after the other without any back-pressure taking place. In case the data are numerous, they can be put the FIFO in overflow. In order to avoid this situation the hysteresis flag is used as a back-pressure on the 2D compressor as well.

2D compressor block

The 2D-compressor algorithm is essentially based on a double-threshold cluster-finding technique. Compression performances are completely tunable using a set of registers that can be programmed using a standard JTAG port. Using this port, the 2D compression itself can be switched on or off and the threshold levels can be modified by writing and reading the JTAG port.

A cluster is defined by a five cross-like 8-bit pixels structure: the five pixels, each composed of five 8-bit data, are named respectively EAST, CENTER, WEST, NORTH and SOUTH. At every clock cycle the CENTER pixel is analyzed: its value is saved if:

- at least it is higher than a low threshold;
- at least one of the five pixels of the cross is higher than the high-threshold (or higher than the low threshold if CENTER > T_H).

In case both conditions are met, the central point of the cross is saved and encoded by means of a variable length encoder. In this way the chosen 8-bit pixels are converted into 5 to 10-bit encoded data depending on their data values. Thus, going back to the cluster finding technique, three main cases can be described:

1. if the cluster is detected alone, separated from others, not only the encoded data value is transferred but also its position within the matrix. This condition applies always at the beginning of a multiple cluster.
2. if the cluster follows a previously detected adjacent cluster, its position is already known since it is “the previous one + 1” and its position information is not transferred. This is a case of a multiple cluster where only the encoded data are transferred, apart from the first data that satisfies the threshold conditions.
3. if the cluster is not detected, no information is put to output.

The electronics of the 2D-compressor itself masks one or two of these five bytes to zero in case the CENTER is located onto the border of the matrix. For example in case the CENTER is located in the higher right corner the compressor sets the NORTH and EAST bytes to zero. Moreover, during the analysis of the 256-th row the compressor sets to zero the SOUTH bytes. This is in order to avoid detecting clusters even where they are not present.

At the end of each stream of data samples (up to 256) that compose each anode, the compressor sets an End Of Row Summary that summarizes the information of the anode itself. For example, occasional parity errors that may arise into CARLOS registers or during the read/write operation on the RAMs. In addition the summary specifies if the bi-dimensional compressor is active or not. Then other internal counter overflow flags are packed in the summary to be able off-line to reconstruct easily all the parameters of the physics of the event.

The CARLOS compression internal unit is mainly composed of two RAM memories and some logic. The two 256 x 9-bit words memories temporarily store the incoming

data for letting the bi-dimensional compressor analyze the potential clusters. The RAM of the compressor unit is composed of the following registers:

- an 8-bit input register storing data coming from the front-end electronics at every clock cycle;
- 2 256 x 9-bit dual-port RAM blocks RAM0 and RAM1,
- 5 8-bit registers NORTH, SOUTH, EAST, CENTER and WEST.

The data values coming from the input register are sequentially written from address 0 of the first memory RAM0 up to the address 255 of the second RAM1 and then again from the first address of the first memory. In this way the two memories are circularly written. At the same time both memories are read at the same address. In this way two rows of the incoming matrix plus one value are stored during each clock cycle, in order to determine if its center value has to be saved or rejected.

Disabling the 2D compressor

When the 2D compression is disabled (*enable 2D* = 0), the following encoding on input data is performed:

Input data range	Output code	Bits
0-1	1 bit & 000	4 bits
2-3	1 LSB bit & 001	4 bits
4-7	2 LSB bits & 010	5 bits
8-15	3 LSB bits & 011	6 bits
16-31	4 LSB bits & 100	7 bits
31-63	5 LSB bits & 101	8 bits
64-127	6 LSB bits & 110	9 bits
128-255	7 LSB bits & 111	10 bits

After receiving the last data word of an event coming from AMBRA, the compressor block keeps sending in output 16 dummy words whose value depends on the data coming from AMBRA. These values have to be rejected while decoding CARLOS outputs.

End of Row Summary

When the 2D compression is enabled, after processing all the samples belonging to an anode, an End of Row (EOR) Summary is sent in output. It is 21-bit long with the following format:

E/O	VL	VZ	VH	NCL	NCZ	NCH	Parity	001
1 bit	1 bit	1 bit	1 bit	3 bits	4 bits	4 bits	3 bits	3 bits

where:
E/O: Even / Odd anode.
VL: Overflow of the counter NCL, it is put to 1 if NCL reaches 7 counts within an anode;
VZ: Overflow of the counter NCZ, it is put to 1 if NCZ reaches 15 counts within an anode;
VH: Overflow of the counter NCH, it is put to 1 if NCH reaches 15 counts within an anode;
NCL: it counts the CENTER pixel values satisfying TL, but not TH.
NCZ: it counts the CENTER pixel values whose value is 0.
NCH: it counts the CENTER pixel values satisfying TH.
Parity: it is a 3-bit code whose meaning is explained in Table 1.
A parity error on CARLOS is detected if one of the following conditions apply:

- a parity error is detected on CARLOS internal RAMs;
- a parity error is detected on CARLOS JTAG programmable registers.

A parity error on AMBRA is detected if the MSB of AMBRA data bus (bit 8) is set to logic level 1.

Parity	meaning
000	no error on A & C (enable 2D = 0)
001	severe error on both A & C
010	sever error on C, no info on A minor error
011	sever error on A, no info on C minor error
100	minor error on both A & C
101	minor error on C, no error on A
110	minor error on A, no error on C
111	no error on A & C (enable 2D = 1)

Table 1: Parity violation information in the EOR Summary.

with the following definitions:

- sever error: >= 7 parity errors on the samples of one anode;
- minor error: < 7 parity errors on the samples of one anode.

A severe error is also reported in bits 3-0 of the error flag words.

The EOR Summary is appended at the end oh each processed anode also when the 2D compression is disabled. In this case its format is the following one:

14 zeros	E/O	Parity	001
14 bits	1 bit	3 bits	3 bits

Data transmission protocol

JTAG mode

During the JTAG programming of one of the front-end devices, the JTAG answer *tdo* is sent to the GOL chip via the 16-bit bus *output_data*. The *tdo* signal can be generated by one of the following devices:

- by the internal boundary scan block on CARLOS;
- by the PASCAL chips;
- by the AMBRA chips;
- by the GOL chip itself

depending on which is the JTAG-addressed device.
In order to sample the *tdo* value only when valid (in fact CARLOS does NOT receive any *tdo_en* input signal), CARLOS monitors the *tms* input signal received over the serial back-link channel, in order to be always aware of the current state of the JTAG state machine. The *tdo* value will then be sampled only during the Shift-DR and Shift-IR states as foreseen by the standard IEEE 1149.1.
Three cases have been foreseen:

- when CARLOS is JTAG-addressed: the *tdo* coming from the internal boundary scan unit is sampled;
- when an other device is JTAG-addressed: the *tdo* coming from outside (either *tdo_from_LH* or *tdo_from_RH* or *tdo_from_GOL*) is sampled. In this case the *tdi* input sent towards CARLOS internal boundary scan unit is tied to 1: so far CARLOS internal instruction register finds a BYPASS command and no undesired change is applied on CARLOS internal registers.
- when in broadcast mode: no *tdo* value is sampled and put in output at all.

When addressing PASCAL, AMBRA or CARLOS:

After being sampled, the selected *tdo* values are then stored into the *fifo_jtag* 128x1 FIFO. Then if the *gol_ready* signal is 1, the FIFO block is read out and the JTAG words are put in output. So far in normal conditions the FIFO is written during one clock cycle and read at the following one, so it never gets full. If the GOL PLL loses the synchronization with its incoming clock (*gol_ready* = 0) while CARLOS is transmitting JTAG words, the *tdo* values are stored into the FIFO and remain there until the *gol_ready* signal is asserted again. If the *gol_ready* stays inactive for a long time (some ms), then the FIFO might overflow and JTAG programming should be run again.

When addressing the GOL chip:

A special issue has to be dedicated to the GOL JTAG programming. After addressing the GOL chip and during its internal registers programming phase, CARLOS samples its *tdo* values (*tdo_from_GOL*) and stores these values into an internal memory (*fifo_jtag*) consisting of 128 bits. The values stored into this buffer are then sent in output only after the JTAG connection towards the GOL

chip has been closed by asserting the *trst* signal and only if the *gol_ready* signal is active (i.e. the GOL chip is ready to transmit data over the optical fibre). In fact when running JTAG instructions on the GOL chip, such as the Sample / Preload or Extest instructions, the chip is no longer able to transmit data. The same holds when writing GOL internal configuration registers, which may cause the PLL synchronization loss. So far during each JTAG programming stage of the GOL chip, no more than 128 bits have to be expected back as a JTAG answer; otherwise the *fifo_jtag* block overflows and the JTAG answer gets corrupted. In the 128 bits total budget, 5 bits have to be taken into account for each instruction sent to the GOL and then n bits for the data register being accessed. For instance, 55 bits for accessing internal configuration registers and 57 bits for accessing GOL internal boundary scan chain. The CORETEST JTAG instruction can not be run on GOL, since its output can be understood only by GOL designers. All the others JTAG instructions can be run on the GOL chip. If the expected budget exceeds 128 bits, two consecutive JTAG connections towards the GOL chip can be opened and run one after the other.

The *tdo* signal is put in output by the *Outmux* block with the following format (see Fig. 4):

- JTAG code: 0110
- JTAG address (the 7-bit word received when opening a JTAG connection)
- 3-bit counter: it is incremented for every *tdo* sample put in output
- *tdo* value
- not(*tdo*) value (in order to have a better fault tolerance)

When a JTAG word is available on the output bus, the strobe pins *tx_en* and *cav* are asserted. In this way CARLOSrx can easily check JTAG answers and verify the correctness of the internal registers value.
Consecutive values of *tdo* in the JTAG answer have the following meaning:

- 100…0 while sending a JTAG instruction, the number of bits depends on the selected device JTAG IR length. CARLOS and GOL have 5-bit long instruction registers, while PASCAL and AMBRA have 16-bit long instruction registers.
- n bits with the register value: its length depends of the register actually being read. For instance, when loading a 9-bit register, the *tdo* signal carries out serially the old value of the register.

The JTAG answer related to the BIST output is not put in output after sending the command RUNBIST since the BIST tests the two macro-channels and the multiplexer as if it was in RUN mode. The JTAG answer is then activated again after sending the command READ BIST RESULT.

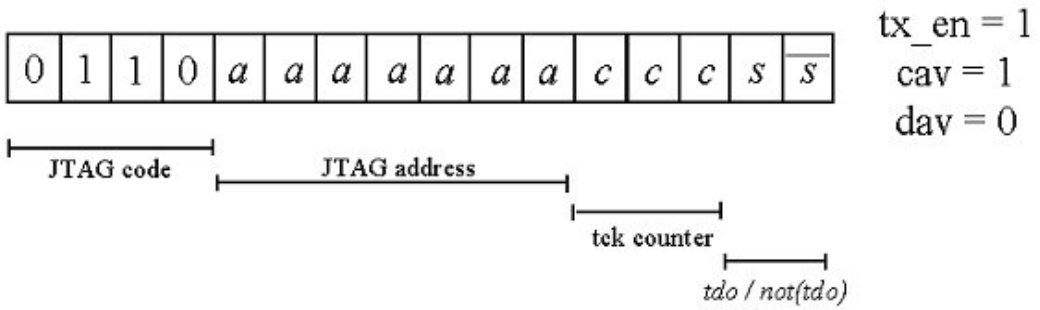


Fig. 4: JTAG answer on the data channel

RUN mode

During the JTAG mode only JTAG words are put in output.
During the RUN mode, processed data are sent in output with the protocol, shown in Fig. 5, that is compliant with both Ethernet and G-Link standard transmission protocols. The *tx_en* strobe is used when choosing the Ethernet mode, while *cav* and *dav* signals have to be used when choosing the G-Link mode. The main difference between the two protocols on the user side is that the G-Link mode allows to distinguish data and control words with different strobe signals, while Ethernet mode implies to make this distinction directly on the 16-bit bus. In order to be able to use both protocols, we decided to identify the output words meaning using the first bits in the following way:

- MSB = 1: data word
- MSBs = 00: error flag word
- MSBs = 010: header
- MSBs = 0111: footer
- MSBs = 0110: JTAG word

The strobe signal *tx_en* is asserted when one of the mentioned words has to be sent to the GOL chip, otherwise it is tied low.
The strobe signal *cav* is asserted when either an error flag word or a JTAG word has to be sent in output (at the same time the *dav* signal is put to 0, *cav* and *dav* are never asserted at the same time).
The strobe signal *dav* is asserted when a data word or header or footer has to be sent in output.
After one of the FIFOs *fifo_data* begins receiving data from the barrel shifter block and when the *gol_ready* signal coming from GOL is 1 (meaning that it is ready to accept input data), a new output data packet is started with 3 identical header words. The header contains a 10-bit event ID word that will be used by CARLOSrx in order to group together data coming from different detectors but belonging to the same event. The 10-bit counter is put to 0 after a reset action, then it is incremented by 1 for every data packet sent in output. The header word also contains a 2-bit CARLOS ID whose value is hardwired externally to CARLOS.

After the header words, data words from the two macro-channels are sent in output in an alternated way starting from ch0 and so on. When a channel has no data to put in output, *tx_en* and *dav* are put to 0 and then, at the next clock cycle, the other channel sends valid output data again, if it has any. So far even in the case one channel only was used, half of the bandwidth would be allocated to the other channel.
At each rising edge of the clock CARLOS samples the *gol_ready* signal value: when 1, CARLOS sends a valid word in output if it has any, otherwise no valid word in sent in output. For a correct data reconstruction on the CARLOSrx side, it is necessary for *gol_ready* to stay active for an even number of clock periods, otherwise the first data after an interruption can be interpreted as belonging to the wrong channel and then cause serious misunderstanding on output data. This has been solved by generating an internal signal on CARLOS that is equal to *gol_ready* when it stays active an even number of clock periods and stays active a period longer when *gol_ready* stays active for an odd number of clock cycles.

Error flag words

The error flag word is sent in output 1 every 64 clock cycles starting from the moment in which CARLOS enters the RUN mode. After an error flag word, data sent in output belongs to the opposite channel with respect to the one sending data before the error word (see Fig. 5). The error word has the following format:

Bit #	**Description**
bit 15-14	00: error flag word identifier
bit 13	L0 acknowledge Asserted after receiving and decoding the L0 command on the serial back-link.
bit 12	L1reject acknowledge Asserted after receiving and decoding the L1reject command on the serial back-link.
bit 11	L2reject acknowledge Asserted after receiving and decoding the L2reject command on the serial back-link.
bit 10	prepulse acknowledge Asserted after receiving and decoding one of the prepulse commands on the serial back-link.
bit 9	testpulse acknowledge Asserted after receiving and decoding the testpulse command on the serial back-link.
bit 8	a flush action has been taken Asserted after the flush has been activated
bit 7	busy It is asserted after CARLOS receives a L0 command and put to 0

	when *dis_trigger1* and *dis_trigger0* go to 0
bit 6	flag error 1 (right hybrid) It is asserted when CARLOS detects a synchronization error with AMBRA on channel 1.
bit 5	flag error 0 (left hybrid) It is asserted when CARLOS detects a synchronization error with AMBRA on channel 0.
bit 4	dis_trigger_mismatch acknowledge It is asserted when CARLOS finds a mismatch between the signals *dis_trigger1* and *dis_trigger0* coming from AMBRA.
bit 3	parity error 1 (right hybrid) It is asserted when CARLOS detects a severe parity error on data coming from AMBRA RH.
bit 2	parity error 0 (left hybrid) It is asserted when CARLOS detects a severe parity error on data coming from AMBRA LH.
bit 1	parity error CARLOS 1 (RH) It is asserted when CARLOS detects a severe parity error on internal RAMs or on internal JTAG programmable registers on channel 1.
bit 0	parity error CARLOS 0 (LH) It is asserted when CARLOS detects a severe parity error on internal RAMs or on internal JTAG programmable registers on channel 0.

Table 2: Format of the error flag word

bits 13-9:

When a command (L0, L1reject, L2reject, one of the prepulse commands, testpulse) is received on the serial back-link and decoded, an internal signal is asserted until an error flag word is sent in output with the corresponding bit set. Then the internal signal is put back to 0 until a new acknowledgment has to be notified to CARLOSrx.

bit 8:

After a synchronization error between AMBRA and CARLOS requiring the use of the flush mechanism, an internal signal is asserted until an error flag word is sent in output with the corresponding bit set. Then the internal signal is put back to 0 until a new notification has to be sent.

bit 7:

The time from the moment when CARLOS receives the trigger command and when the busy signal is asserted is not fixed. In fact it depends on the relative timing from the moment in which the trigger command is received and decoded and when the next error flag word is issued. It is a variable length period from 1 to 64 clock cycles. Then busy signal stays active for the time PASCAL needs to transfer the content of the analog memory to AMBRA or, when all AMBRA buffers are full, the time needed to empty one of the 4 buffers.

bits 6-5:

Once asserted, the flag error remains to logic level 1 until a reset action or until CARLOS in put in JTAG mode, when Stop If Error is 1 (with the exception of the case in which the flush mechanism is activated). When Stop If Error is 0, the flag error is asserted only for one error flag word, then it is reset.

bit 4:

If there are no connection problems, the 2 *dis_trigger* signals coming from the left and right hybrid may differ only when the following situation occurs: when all AMBRA buffers are full, supposing that channel 1 transfers to CARLOS an event in a shorter time than channel 0, there will be a time in which one *dis_trigger* signal is already 0 and the other one still fixed to 1. For this reason CARLOS internally performs the OR of the incoming *dis_trigger* signal for the generation of the *busy* signal.

bits 3-0:

Bits 3-0 of the error flag words are asserted only when a severe parity error occurs either on AMBRA or on CARLOS or on both devices. In fact when finding out 7 parity errors or more while processing data samples belonging to an anode, the related bit is asserted for the upcoming one or two error flag words. The parity error is notified in one or two error flag words depending on the timing the error is found with respect to the timing in which internal flags are sampled before being sent in output. Let's say, for example, that when processing anode number 41 and after processing 201 samples a severe error is detected; let's also suppose that an error flag word is sent in output while processing sample number 221. Since the internal flag reporting a severe parity error will be asserted until the end of the row, its value will also be sent in the forth-coming error flag word.

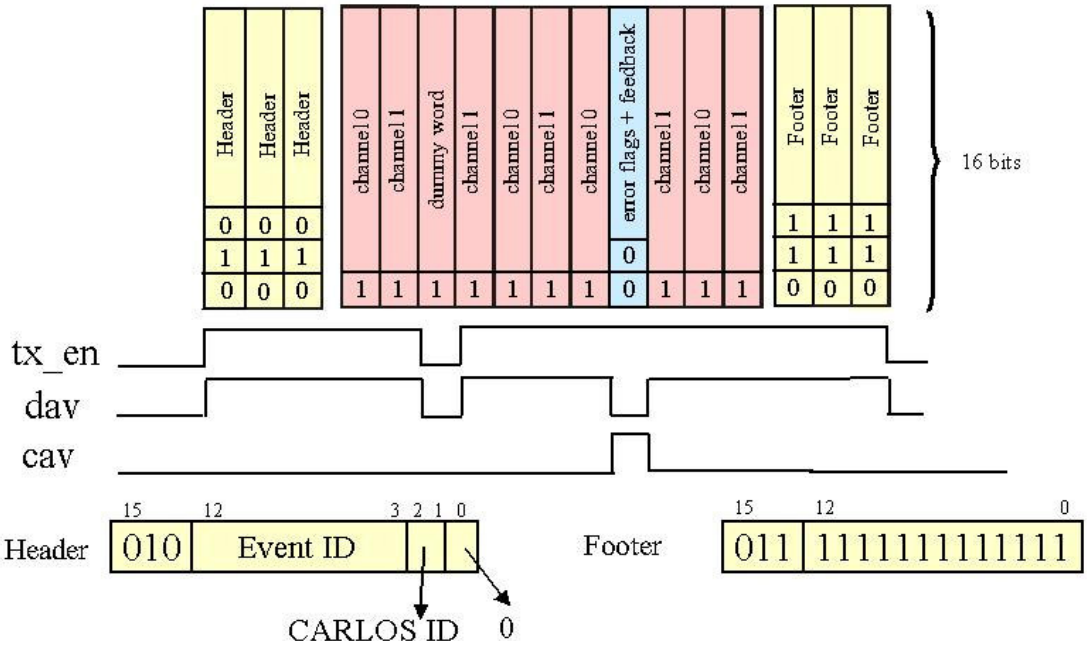


Fig. 5: CARLOS v4 output transmission protocol

CARLOS v4 pin position and function

Terminal name	no	Type	Description
data_in1(8-0)	45-32	I	Input data bus coming from the right hybrid. The MSB is the parity bit from AMBRA: when 1, a parity error has been detected on AMBRA .
data_in0(8-0)	31, 3-11	I	Input data bus coming from the left hybrid. The MSB is the parity bit from AMBRA: when 1, a parity error on AMBRA has been detected.
data_write1	18	I	Input signal coming from RH: when 1 *data_in1* is valid and has to be accepted
data_write0	19	I	Input signal coming from LH: when 1 *data_in0* is valid and has to be accepted
data_end1	20	I	Input signal coming from RH: it is asserted in coincidence with the last valid data of each event.
data_end0	21	I	Input signal coming from LH: it is asserted in coincidence with the last valid data of each event.
data_stop1	48	O	Output signal sent to RH: when 1 it means that CARLOS can no longer accept input data.
data_stop0	47	O	Output signal sent to LH: when 1 it means that CARLOS can no longer accept input data.
trigger1	89	O	After receiving the L0 command on the serial back-link, *trigger1* is asserted for one clock period.
trigger0	90	O	After receiving the L0 command on the serial back-link, *trigger0* is asserted for one clock period.
aft1	77	O	Abort / flush / testpulse signal sent to RH
aft0	78	O	Abort / flush / testpulse signal sent to LH
dis_trigger1	23	I	Input from RH: when asserted no trigger signals have to be sent towards AMBRA
dis_trigger0	22	I	Input from LH: when asserted no trigger signals have to be sent towards AMBRA
gol_ready	46	I	Input from GOL: when asserted the GOL chip is ready to receive data, otherwise data transmission has to be stopped.
tx_en	76	O	Output strobe to GOL: when asserted the output data bus contains a valid data value to be transmitted
cav	50	O	Output strobe to GOL: when asserted the output data bus contains a control word to be transmitted, either a JTAG word or an error flag word, depending on CARLOS working mode.
dav	49	O	Output strobe to GOL: when asserted the output data bus contains a data word to be transmitted (header or footer or data word)
output_data(15-0)	75-55	O	Output bus towards GOL.

ck	39	I	Input clock signal.
reset	25	I	Active low reset. When active all internal registers are initialized.
reset_LH	79	O	Active high signal towards LH. It can be asserted asynchronously by the *reset* input or synchronously when receiving the "Reset LH" command: in this case it stays active for 8 clock periods.
reset_RH	80	O	Active high signal towards RH. It can be asserted asynchronously by the *reset* input or synchronously when receiving the "Reset RH" command: in this case it stays active for 8 clock periods.
reset_GOL	81	O	Active low signal towards GOL. It can be asserted asynchronously by the *reset* input or synchronously when receiving the "Reset GOL" command: in this case it stays active for 8 clock periods.
ck_out	63	O	The input *ck* is directly propagated in the output signal *ck_out*.
serial_backlink	24	I	Input serial signal sending CARLOS reset, JTAG commands and trigger information using 8-bit DC-balanced codes.
tdi_to_LH	82	O	Test Data Input to LH propagated to LH when a JTAG connection towards LH (or a broadcast connection) has been opened.
tms_to_LH	86	O	Test Mode Select to LH propagated to LH when a JTAG connection towards LH (or a broadcast connection) has been opened.
trst_to_LH	95	O	Test Reset to LH propagated to LH when a JTAG connection towards LH (or a broadcast connection) has been opened.
tck_to_LH	98	O	Test Clock to LH propagated to LH when a JTAG connection towards LH (or a broadcast connection) has been opened.
tdi_to_RH	84	O	Test Data Input to RH propagated to RH when a JTAG connection towards RH (or a broadcast connection) has been opened.
tms_to_RH	93	O	Test Mode Select to RH propagated to RH when a JTAG connection towards RH (or a broadcast connection) has been opened.
trst_to_RH	96	O	Test Reset to RH propagated to RH when a JTAG connection towards RH (or a broadcast connection) has been opened.
tck_to_RH	99	O	Test Clock to RH propagated to RH when a JTAG connection towards RH (or a broadcast connection) has been opened.
tdi_to_GOL	85	O	Test Data Input to GOL propagated to GOL when a JTAG connection towards GOL has been opened.

tms_to_GOL	94	O	Test Mode Select to GOL propagated to GOL when a JTAG connection towards GOL has been opened.
trst_to_GOL	97	O	Test Reset to GOL propagated to GOL when a JTAG connection towards GOL has been opened.
tck_to_GOL	100	O	Test Clock to GOL propagated to GOL when a JTAG connection towards GOL has been opened.
tdo_from_RH	13	I	JTAG answer coming from RH.
tdo_from_LH	12	I	JTAG answer coming from LH.
tdo_from_GOL	14	I	JTAG answer coming from GOL.
prepulse	91	O	Output signal asserted from 1 to 8 clock cycles after receiving the prepulse commands on the serial back-link.
carlos_id(1-0)	15, 17	I	Hardwired inputs containing an information that will be put in output in the header word at the beginning of each data packet.
set_test(2-0)	26, 1, 2	I	Test pins driving a multiplexer selection lines used to bring in output internal signals coming from RAMs, compressor, barrel shifter and FIFO.
test_output(7-0)	30-27,54-51	O	Internal signals put in output.

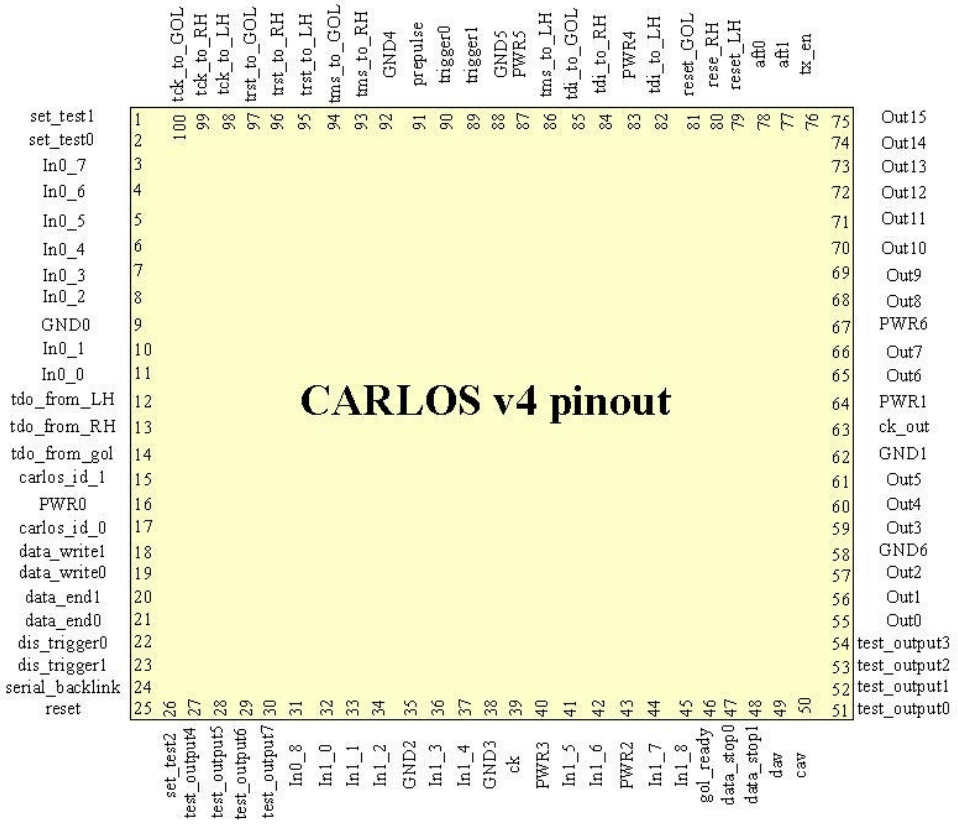


Fig. 6: CARLOS v4 pinout

Programming CARLOS v4

Controlling CARLOS via the serial back-link

CARLOS v4 is remotely controlled using a serial back-link coming from the CARLOSrx device. Data transferred on the serial link are synchronous to the incoming 40 MHz master clock. CARLOS has a synchronization state machine which is responsible for handling link initialization and synchronization (see Fig. 7). Upon power up or external reset via the serial back-link, the state machine enters the acquisition state (ACQ) and searches for the IDLE pattern. Upon receiving three consecutive IDLE patterns after the first one, the state machine enters the synchronization state (SYNC). If an invalid code is received, the state machine transitions to the CHECK state. If, in the CHECK state, CARLOS sees 4 consecutive valid codes, the state machine acknowledges that the link is good and transitions back to the SYNC state for normal operation. If, in the CHECK state, CARLOS sees 3 invalid codes (not required to be consecutive), the state machine determines a loss of the link has occurred and transitions the synchronization state machine back to the ACQ mode. Table 3 shows the list of commands that CARLOS can receive through the serial back-link from CARLOSrx: they are all DC-balanced (same number of 1s and 0s).

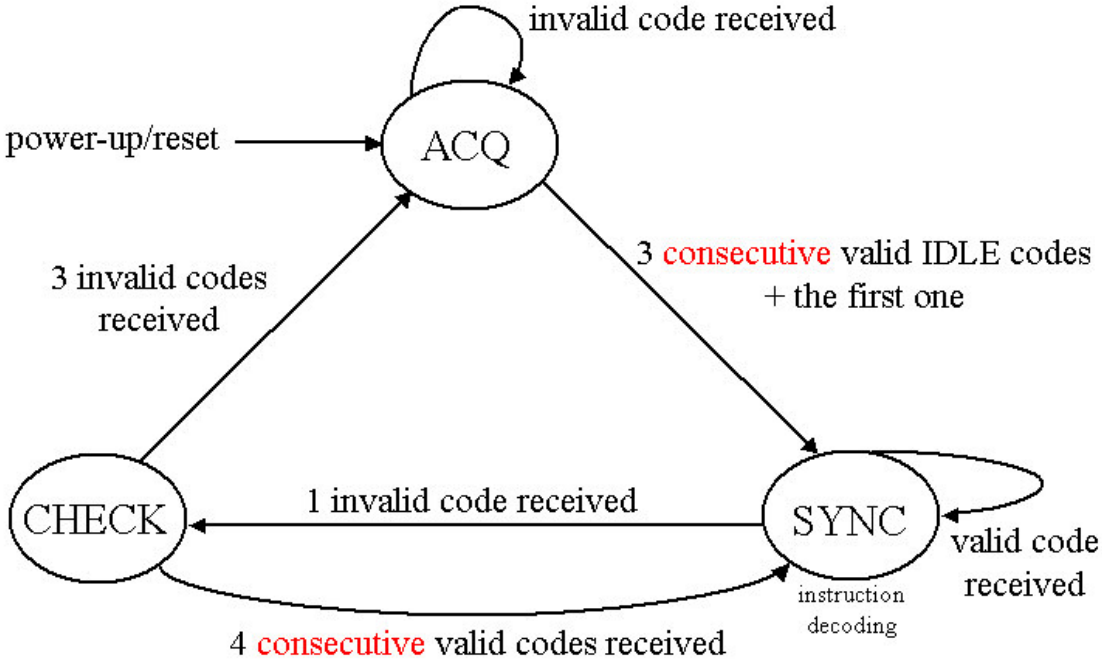


Fig. 7: Synchronous serial back-link state machine decoding

Command	Code	Consequence
reset carlos	1100|1100	a reset is distributed to all CARLOS internal blocks, except for the serial receiver state machine (8 clock cycles long, active low)
reset left hybrid	1100|0011	a reset is sent to the left hybrid (8 clock cycles long, active high)
reset right hybrid	1100|1010	a reset is sent to the right hybrid (8 clock cycles long, active high)
reset GOL	1100|0101	a reset is sent to the GOL chip (8 clock cycles long, active low)
L0	1010|1100	*trigger1* and *trigger0* signals are activated
L1reject	1010|0011	*aft1* and *aft0* are asserted for 1 clock cycle
L2reject	1010|1010	*aft1* and *aft0* are asserted for 1 clock cycle
test pulse	1010|1001	*aft1* and *aft0* are asserted for 3 clock cycles
prepulse25	1001|0011	*prepulse* output is asserted for 1 clock cycle towards the charge injectors
prepulse50	1001|0101	*prepulse* output is asserted for 2 clock cycles towards the charge injectors
prepulse75	1001|0110	*prepulse* output is asserted for 3 clock cycles towards the charge injectors
prepulse100	1001|1001	*prepulse* output is asserted for 4 clock cycles towards the charge injectors
prepulse125	1001|1010	*prepulse* output is asserted for 5 clock cycles towards the charge injectors
prepulse150	1001|1100	*prepulse* output is asserted for 6 clock cycles towards the charge injectors
prepulse175	1110|0001	*prepulse* output is asserted for 7 clock cycles towards the charge injectors
prepulse200	1110|1000	*prepulse* output is asserted for 8 clock cycles towards the charge injectors
idle	0011|1001	used for link synchronization only
enter JTAG mode	1101|0010	CARLOS enters JTAG mode
enter RUN mode	1101|1000	CARLOS enters RUN mode
Stop acquisition	1000|0111	CARLOS stops data transmission by asserting *data_stop0* and *data_stop1*

Restart acquisition	1000|1110	CARLOS restarts data transmission by putting *data_stop0* and *data_stop1* to 0
JTAG	01|*trst*|not(*trst*)|*tms*|not(*tms*)|*tdi*|not(*tdi*)	JTAG values for one tck period

Table 3: List of serial back-link commands

Opening a JTAG connection

CARLOSrx may open a JTAG connection towards CARLOS, PASCAL, AMBRA or GOL by using the serial back-link and sending JTAG commands. Every JTAG command sent to CARLOS over the serial back-link is interpreted by CARLOS as the values of a JTAG port during a JTAG clock period. In fact an internal CARLOS block, named "*generate JTAG*", reconstructs *tdi*, *tms* and *trst* data decoded from the serial back-link and builds a JTAG *tck* clock. For each 8-bit input JTAG command, the *tck* signal stays 0 for 4 clock periods and 1 for the following 4 clock periods, resulting in a total JTAG frequency of 5 MHz. So far the "*generate JTAG*" block provides a standard JTAG port that will be used in the internal JTAG control unit and, when requested, sent to PASCAL, AMBRA and GOL chips.
While keeping *trst* = 0 and *tms* = 1, CARLOSrx sends a JTAG address to CARLOS on the *tdi* input pin, synchronously with *tck*, with the following format:

- 2 IDLE bits: 1-1
- 2 SELECT bits: 0-0
- 14 ADDRESS bits: 7 bits with the swallow protocol (01 stands for 1, 10 stands for 0)
- 2 PARITY bits: 1 bit with the swallow protocol (odd parity, even parity for Synopsys DesignWare manual)
- 2 SELECT bits: 0-0
- 2 IDLE bits: 1-1

If this 24-bit sequence is not exactly recognized by CARLOS or the parity bit received is different from the calculated parity, the JTAG connection is not opened and a new input frame is analyzed searching for a correct address. CARLOSrx realizes that no JTAG connection has been opened since it does not receive any answer from the selected device; after a time-out period it tries to open a new connection again.
The 24-bit sequence is also sent as it is by CARLOS to both hybrids in order to allow them to identify the chip chosen for the connection. After sending the frame, CARLOSrx begins then sending JTAG information that is sent to the selected device only.
A broadcast operation allows to address more than one chip at the same time in order to save time during reprogramming internal registers. This is the only case in which no *tdo* answer is sent back to CARLOSrx on the *output_data* bus.
The address is decoded with the scheme reported in Table 4.

address	device addressed
$b_6b_5b_4 = 000$	left hybrid
$b_6b_5b_4 = 001$	right hybrid
$b_6b_5b_4 = 010$	both hybrids
$b_6b_5b_4 = 011$	GOL
$b_6b_5b_4 = 1xx$	CARLOS
$b_3 = 1$	broadcast address
$b_3 = 0$	individual address
b_2, b_1	one among 4 pairs
$b_0 = 0$	PASCAL
$b_0 = 1$	AMBRA

Table 4: JTAG address decoding scheme

So far it is possible, for instance, to program at the same time all the AMBRA chips of both hybrids or only one hybrid or to program at the same time 2 AMBRA chips on two different hybrids with the same values. This features may be useful in order to have a fast JTAG programming phase, but they do not allow to read back the JTAG registers.

CARLOS internal registers

CARLOS v4 contains nine user-accessible JTAG registers, which are listed in Table 5. The MSB is the parity bit in registers from 1 to 6, calculated with even parity.

Register number	Bits	Perms	Register name	Default content (after reset)
1	9	R/W	T1L left	000000000 (0h000)
2	9	R/W	T1H left	000000000 (0h000)
3	9	R/W	Anode length left	011111111 (0h0FF)
4	9	R/W	T1L right	000000000 (0h000)
5	9	R/W	T1H right	000000000 (0h000)
6	9	R/W	Anode length right	011111111 (0h0FF)
7	1	R/W	Enable 2D	1
8	1	R/W	Stop If Error	1
9	16	R	BIST register	0000000000000000 (0h0000)

Table 5: List of CARLOS JTAG internal registers

After CARLOS has been reset, all the registers are initialized to a default content, then registers 1-8 can both be written or read using the standard JTAG port. On the contrary register 9, the 16-bit long BIST register, can only be read using the JTAG port: its value is written by the *Signature Maker* block after the running the JTAG command RUNBIST.

T1L left:

- low threshold value for the macro-channel receiving data from the left hybrid

T1H left:

- high threshold value for the macro-channel receiving data from the left hybrid

Anode length left:

- number of samples decremented by 1 belonging to a SDD anode being read from the AMBRA chips belonging to the left hybrid

T1L right:

- low threshold value for the macro-channel receiving data from the right hybrid

T1H right:

- high threshold value for the macro-channel receiving data from the right hybrid

Anode length right:

- number of samples decremented by 1 belonging to a SDD anode being read from the AMBRA chips belonging to the right hybrid

Enable 2D:

- when asserted compression 2D is enabled, otherwise a lossless encoding of data is performed

Stop If Error:

- when asserted it causes the data transmission to stop after a synchronization error between AMBRA and CARLOS, by putting the *data_stop* signals to a high level until a *reset* is received (except for the case when the flush mechanism is activated). When 0, an occurring synchronization error would be detected without affecting the data transmission process.

BIST register:

- BIST signature final value (the expected value when running the BIST with the default values for the other programmable registers is 0h1990).

Register access via the JTAG bus

The JTAG standard defines a serial communication protocol for testing and programming purposes. In CARLOS v4 the JTAG interface supports 4 tasks:

- boundary scan;
- access (R/W) to internal registers;
- test of the PCB (Using the Sample/Preload instruction the chip input values can be sampled and stored into the boundary scan register, until they are shifted out and compared to the expected values. In the same way after running the Sample/Preload and Extest instructions, one after the other, the chip outputs can be set to a given value. In this way, for instance, it is possible to transmit data words to the GOL chip and check if the connection CARLOS-GOL on the PCB is correct or not).
- BIST

The different functions are reflected in a number of scan registers. A specific scan path can be selected by writing its 5-bit code into the instruction register (IR) inside the on-chip JTAG controller. CARLOS v4 contains scan paths from *tdi* to *tdo* of different lengths, from 1 to 82 bits. Table 6 shows the JTAG instruction set implemented on CARLOS v4.

JTAG instruction	JTAG IR value	Length of scan register involved
Extest	00000	82
Sample / preload	00010	82
Bypass	11111	1
Intest	00011	82
Runbist	00100	16

Load T1L left	00101	9
Load T1H left	00110	9
Load Al left	00111	9
Load enable 2D	01001	1
Read T1L left	01010	9
Read T1H left	01011	9
Read enable 2D	01110	1
Read BIST result	01111	16
Load T1L right	10101	9
Load T1H right	10110	9
Load Al right	10111	9
Read T1L right	11010	9
Read T1H right	11011	9
Read Al right	11100	9
Load Stop If Error	01000	1
Read Stop If Error	01101	1

Table 6: List of CARLOS JTAG instructions

After reset, all internal registers are given an initial value. After opening a JTAG connection towards CARLOS, they can be easily programmed by serially providing a JTAG instruction followed by the register value on the serial back-link channel, following the standard JTAG state machine. While writing a register new value, the old one is put in output, so to allow a quick check of the register values being written. On the other side while reading a register, the *tdi* input value is ignored and the register is shifted on itself in a circular way, so that after *n* shift operations the register holds the right value again (for a *n* bit register). This allows to avoid to use a double register instead of one.
When a JTAG connection is opened towards other devices, CARLOS internal boundary unit continues monitoring the *tms* and *tck* values in order to be aware of the current JTAG state (so to sample the incoming *tdo* value at the correct clock periods), while its internal JTAG control unit receives a masked *tdi* input (*tdi* = 1 fixed). In this way each incoming JTAG instruction is interpreted as the BYPASS command and the internal bypass register is addressed, so far avoiding to modify the other JTAG registers. The mechanism so far shown is correct since the number of bits used to code the JTAG instruction register value on CARLOS is less or equal than for the IR on AMBRA and the GOL chips: in fact GOL's IR is 5-bit long, while AMBRA's IR is 16-bit long. Should a device receiving the JTAG signals from CARLOS have a 3-bit IR, the internal boundary unit on CARLOS would not always decoded the IR value as BYPASS and CARLOS internal register corruption would be unavoidable.
The boundary scan register (BSR) includes all the I/O signals with the exception of clocks, the reset signal and the serial back-link signal. Table 7 shows the list of

CARLOS I/Os included in the Boundary Scan register together with the read-out order.

Order for shift out	Pin #	Name	Type
1	100	*tck_to_GOL*	out
2	99	*tck_to_RH*	out
3	98	*tck_to_LH*	out
4	97	*trst_to_GOL*	out
5	96	*trst_to_RH*	out
6	95	*trst_to_LH*	out
7	94	*tms_to_GOL*	out
8	93	*tms_to_RH*	out
9	91	*prepulse*	out
10	90	*trigger0*	out
11	89	*trigger1*	out
12	86	*tms_to_LH*	out
13	85	*tdi_to_GOL*	out
14	84	*tdi_to_RH*	out
15	82	*tdi_to_LH*	out
16	81	*reset_GOL*	out
17	80	*reset_RH*	out
18	79	*reset_LH*	out
19	78	*aft0*	out
20	77	*aft1*	out
21	76	*tx_en*	out
22	75	*output_data(15)*	out
23	74	*output_data(14)*	out
24	73	*output_data(13)*	out
25	72	*output_data(12)*	out
26	71	*output_data(11)*	out
27	70	*output_data(10)*	out
28	69	*output_data(9)*	out
29	68	*output_data(8)*	out
30	66	*output_data(7)*	out
31	65	*output_data(6)*	out
32	61	*output_data(5)*	out
33	60	*output_data(4)*	out
34	59	*output_data(3)*	out
35	57	*output_data(2)*	out
36	56	*output_data(1)*	out
37	55	*output_data(0)*	out
38	54	*test_output(3)*	out

39	53	*test_output(2)*	out
40	52	*test_output(1)*	out
41	51	*test_output(0)*	out
42	27	*test_output(7)*	out
43	28	*test_output(6)*	out
44	29	*test_output(5)*	out
45	30	*test_output(4)*	out
46	50	*cav*	out
47	49	*dav*	out
48	48	*data_stop1*	out
49	47	*data_stop0*	out
50	46	*gol_ready*	in
51	45	*data_in1(8)*	in
52	44	*data_in1(7)*	in
53	42	*data_in1(6)*	in
54	41	*data_in1(5)*	in
55	37	*data_in1(4)*	in
56	36	*data_in1(3)*	in
57	34	*data_in1(2)*	in
58	33	*data_in1(1)*	in
59	32	*data_in1(0)*	in
60	31	*data_in0(8)*	in
61	26	*set_test(2)*	in
62	23	*dis_trigger1*	in
63	22	*dis_trigger0*	in
64	21	*data_end0*	in
65	20	*data_end1*	in
66	19	*data_write0*	in
67	18	*data_write1*	in
68	17	*carlos_id(0)*	in
69	15	*carlos_id(1)*	in
70	14	*tdo_from_GOL*	in
71	13	*tdo_from_RH*	in
72	12	*tdo_from_LH*	in
73	11	*data_in0(0)*	in
74	10	*data_in0(1)*	in
75	8	*data_in0(2)*	in
76	7	*data_in0(3)*	in
77	6	*data_in0(4)*	in
78	5	*data_in0(5)*	in
79	4	*data_in0(6)*	in
80	3	*data_in0(7)*	in

81	2	*set_test(0)*	in
82	1	*set_test(1)*	in

Table 7: Boundary scan register list

Board level testing via JTAG

The JTAG standard also allows to perform board level testing by using the boundary scan chain properties. The following procedure is usually performed:

- the Sample / Preload instruction is run on chip A, loading a predefined pattern on chip A boundary scan chain;
- the Extest instruction is run on chip A in order to load the predefined pattern on the chip A outputs;
- the Sample / Preload instruction is run on chip B, that samples chip A outputs, stores these values in the boundary scan register and then shifts this information serially out on the *tdo* pin.

In this way it is possible to check the electrical connection between two chips.
It is not possible to apply this procedure to AMBRA, CARLOS and GOL chips. In fact a JTAG connection can be addressed to one device at a time or to several devices of the same kind. Just as an example let's try to repeat the test procedure for the AMBRA – CARLOS system:

1. the Sample / Preload instruction is run on AMBRA, loading a predefined pattern on AMBRA outputs;
2. the Extest instruction is run on AMBRA in order to load the predefined pattern on AMBRA outputs;
3. a *trst* = 1 signal is sent to AMBRA in order to close the JTAG connection;
4. CARLOS is JTAG-addressed;
5. the Sample / Preload instruction is run on CARLOS, that samples AMBRA outputs, stores these values in the boundary scan register and then shifts this information out using the 16-bit JTAG words.

Unfortunately point 3 puts all AMBRA outputs to their default value, overriding the predefined pattern (the same holds also for CARLOS and GOL).
An alternative procedure is proposed for the AMBRA and CARLOS chips:

- the Sample / Preload instruction is run on AMBRA, loading a predefined pattern on AMBRA outputs (with *data_write* = 1);
- the Extest instruction is run on AMBRA in order to load the predefined pattern on AMBRA outputs;
- CARLOSrx sends to CARLOS the command "Enter RUN mode" on the serial back-link;
- CARLOS outputs are decoded in order to understand which are the values of the input bus received from AMBRA and compared with the expected pattern.

An alternative procedure is proposed for the CARLOS and GOL chips:

- the Sample / Preload instruction is run on CARLOS, loading a predefined pattern on CARLOS outputs (with *tx_en* and *dav* = 1);
- the Extest instruction is run on CARLOS in order to load the predefined pattern on CARLOS outputs;
- the data transmitted through GOL over the optical fibre is de-serialized and compared with the expected pattern.

Alternative proposals are welcome.

Chip level testing via JTAG

The quickest way to run an internal chip test via JTAG is to use the BIST utility.
On the contrary the Intest instruction does not work as expected from the standard at least for what concerns the *output_data* bus. Let's suppose to run the following procedure:

1. the Sample / Preload instruction is run on CARLOS, loading a predefined pattern on its boundary scan chain;
2. the Intest instruction is run in order to send the predefined pattern as input to CARLOS core;
3. the Sample / Preload instruction is run on CARLOS, for sampling the test outputs and sending them in output.

The problem is that, when running point 3, the *output_data* bus is used for sending in output the acknowledgment of the received instruction (1 followed by n zeros) and, in this way, the value of the test output is over-ridden and lost.
So far the Intest instruction can be used for testing all CARLOS core outputs but the *output_data* bus, *tx_en*, *cav* and *dav*.

Running CARLOS v4

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

How to run CARLOS v4

CARLOS v4 utilization should include the following sequence of actions:

1) power supply to the chip is turned on and CARLOS receives a signal reset (active low and at least one clock cycle long) from an external RC network. This reset signal is propagated from CARLOS to the right hybrid (active high), left hybrid (active high) and GOL (active low). See Timing 1.
2) after asserting the reset signal CARLOS is put by default in JTAG mode, during which *data_stop1* and *data_stop0* outputs are kept high. During this working mode the *busy* signal towards the CTP is kept high by CARLOSrx (CARLOS does not send any busy information in output). See Timing 2.
3) an IDLE command sequence is sent on the serial back-link channel from CARLOSrx towards CARLOS, so to allow to get the synchronization with the input pattern. After synchronization has been reached, selective reset commands can be sent to CARLOS if desired. See Timing 3a and 3b.
4) a JTAG pattern consisting in a redundant 7-bit address on the serial back-link input pin is sent in order to select which JTAG connection to open. If the address is correctly recognized with the right format and the parity is OK, the JTAG connection is opened: from this time on the input JTAG information are sent in input to the selected device only. Otherwise CARLOS begins waiting for a new and correct JTAG address. Since JTAG commands are encapsulated on the serial back-link using 8-bit commands, it results in a total JTAG frequency of 5 MHz (40 MHz/8). See Timing 4, in which a JTAG connection is opened towards CARLOS itself.
5) The selected device (CARLOS or left hybrid or right hybrid or GOL) is programmed using the standard JTAG IEEE 1149.1 protocol that allows both internal and external chip test and writing and reading internal registers. The JTAG answer *tdo* is put in output on the 16-bit data bus. See Timing 5.

   Also in case CARLOS has not been selected for the JTAG connection, its boundary scan unit monitors the *tck* and *tms* input signals in order to be able to sample the *tdo* signal coming from the selected device only when valid (it has not to be sampled, for instance, when in high impedance). In this case CARLOS internal boundary unit receives a masked *tdi* (*tdi* = 1), so that each instruction is interpreted as a BYPASS command. In this way no CARLOS internal register is modified, when other devices are being JTAG-addressed.

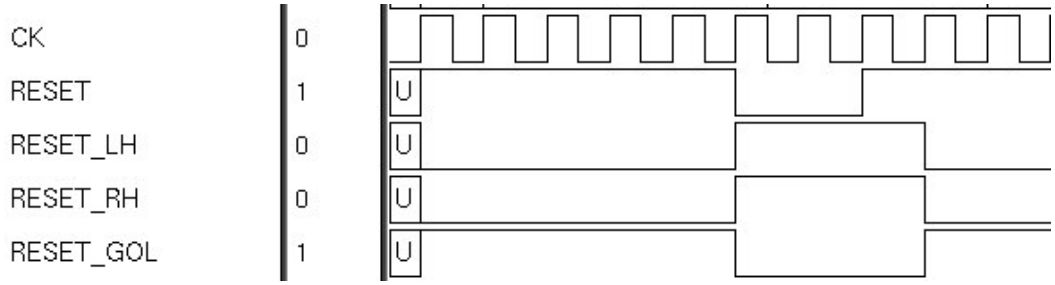
   The JTAG connection can be closed in either of two ways: either when the input *trst* signal is asserted or when CARLOS receives the *reset* signal.

Nevertheless it is recommended to close the connection with *trst* instead of resetting CARLOS.

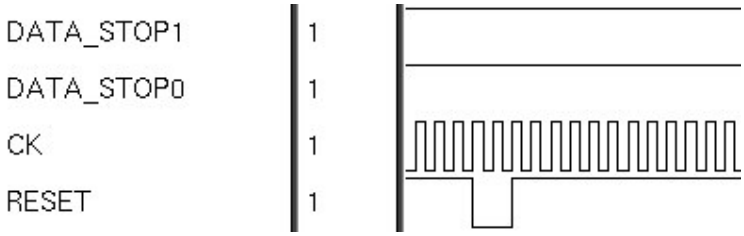
6) By opening JTAG connections, testing and programming and then closing the connections, all PASCAL, AMBRA, CARLOS and GOL chips can be programmed with the desired values before entering the RUN phase. Using the Sample / Preload and Extest JTAG instruction, it is possible to set all CARLOS outputs with a predefined pattern. In Timing 6 all CARLOS outputs have been put to logic state 1.
7) After completing the JTAG programming task, the command "Enter RUN mode" can be sent to CARLOS on the serial back-link. After this happens, CARLOS puts *data_stop1* and *data_stop0* low and begins waiting for input trigger commands from the serial back-link and then for data on the two input channels. From this time on an error flag word is transmitted in output every 64 clock cycles (1 every 1.6 μs). Among the other information the error flag word also contains the *busy* signal used to stop incoming trigger signals when PASCAL is transferring data to AMBRA or when all the 4 AMBRA buffers are full (*dis_trigger* signals high). See Timing 7.
8) After receiving a trigger command, the *trigger1* and *trigger0* signals are asserted for one clock period. AMBRA replies by setting the *dis_trigger* signals to 1 until it is ready to accept a new trigger signal. See Timing 8.
9) AMBRA begins sending data to CARLOS and then data compression process begins on the incoming samples. Compressed data are then sent in output towards the GOL chip using either Ethernet or G-Link modes. See Timing 9.
10) If AMBRA sends to CARLOS less data words than expected or if AMBRA sends to CARLOS from 1 to 4 more words than expected, a synchronization error occurs in the communication AMBRA – CARLOS. If *Stop If Error* is 1, the error is highlighted in the next error flag word and the data transmission process is stopped by setting the *data_stop1* and *data_stop0* signals. If *Stop If Error* is 1 the transmission process goes on. See Timing 10a and 10b.
11) When the flush mechanism is activated: CARLOS asserts the *data_stop* signal after receiving the last expected data word and then, if it does not receive the *data_end* signal within 4 clock periods, it asserts the *aft* signal for 2 clock periods flushing AMBRA and highlights this situation in the next error flag word. After being flushed, AMBRA asserts the *data_end* signal for one clock cycle. After this happens, CARLOS internal error flag signal is reset, the *data_stop* signals turn to 0 after the internal FIFOs have been emptied and a new event can be fetched and processed. In this case CARLOS behavior is the same regardless of the Stop If Error value. See Timing 11.
12) When the *gol_ready* signal goes to 0, CARLOS stops sending data towards GOL until the *gol_ready* signal is asserted again. When CARLOS internal

FIFO get almost full, the *data_stop* signals are asserted, thus stopping backward the data transmission. See Timing 12a and 12b.

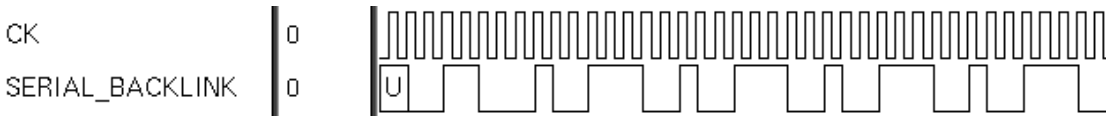
13) When CARLOSrx applies the back-pressure towards CARLOS, the *data_stop* signals are asserted and the data transmission is frozen until the "Restart Acq" command is received over the serial back-link. During the pause time, it is possible to access CARLOS and the other front-end chips via JTAG. See Timing 13.



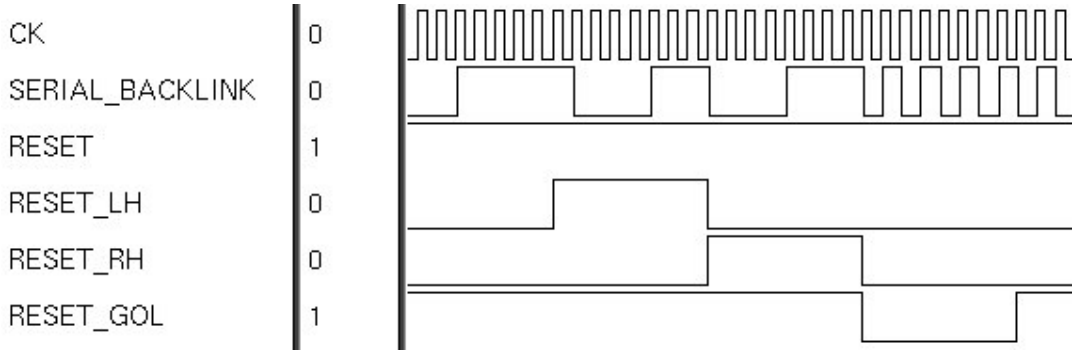
Timing 1: Power-on reset sequence



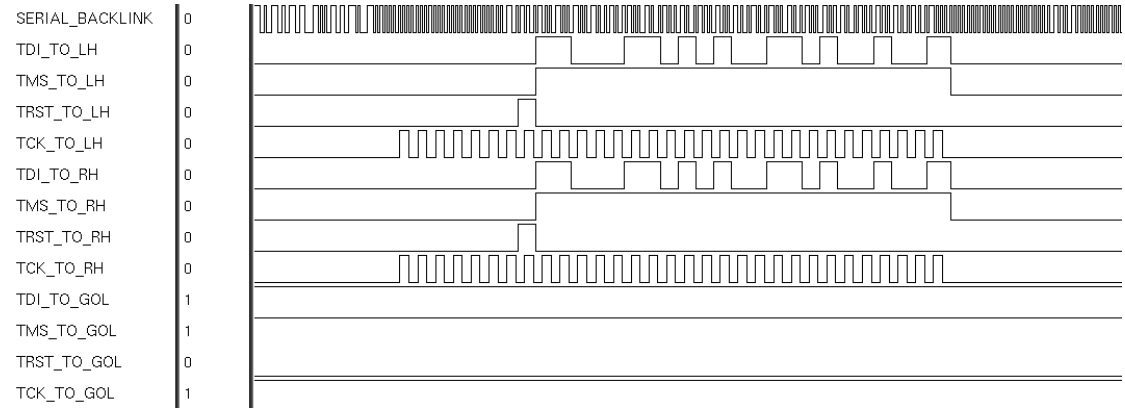
Timing 2: Entering JTAG mode after reset



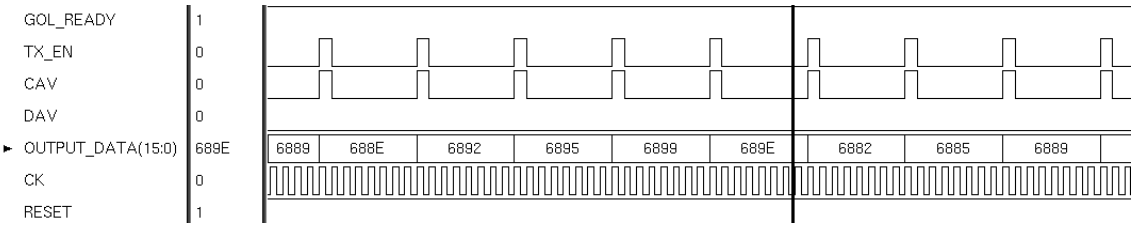
Timing 3a: IDLE sequence for link synchronization



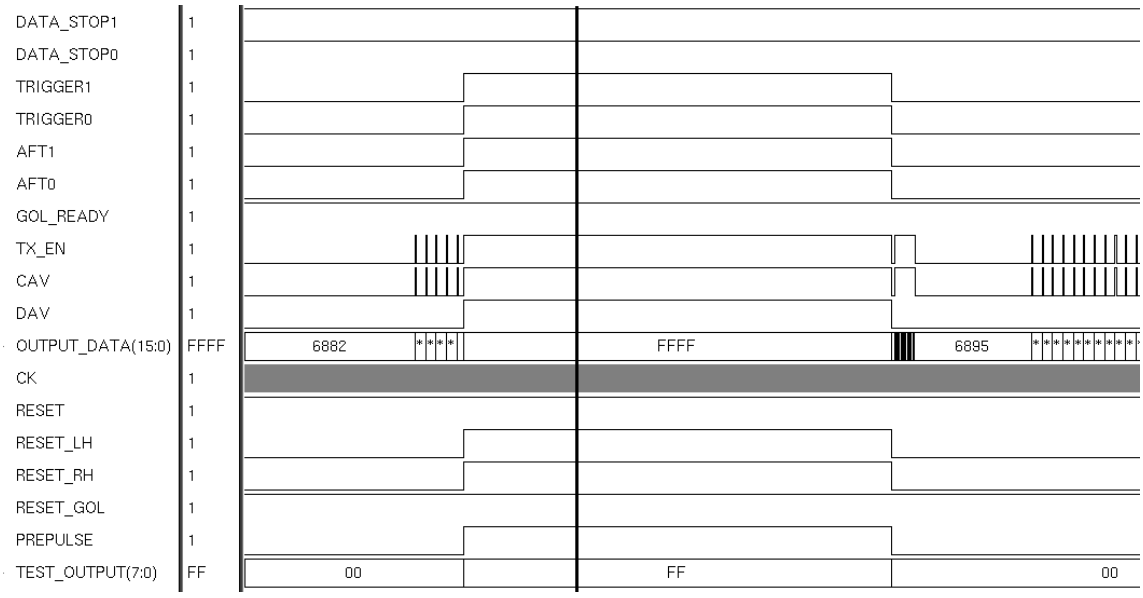
Timing 3b: Selective reset commands



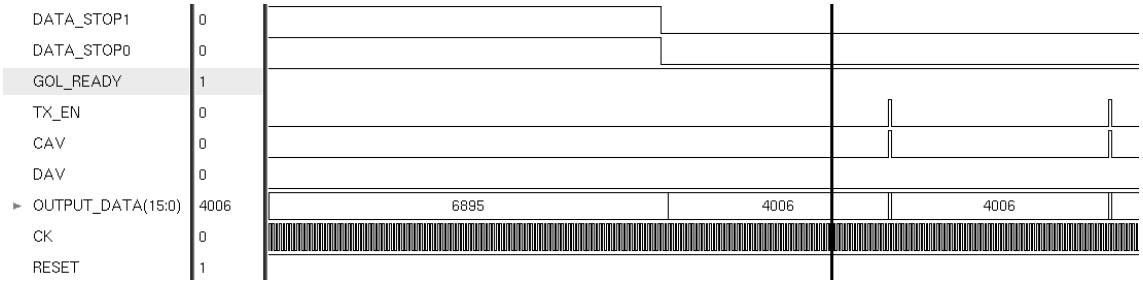
Timing 4: Opening a JTAG connection (towards CARLOS, in this case)



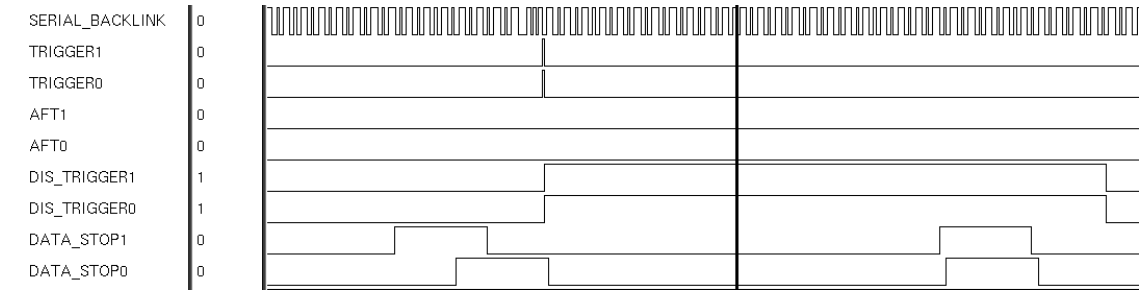
Timing 5: JTAG answer over the 16-bit data bus



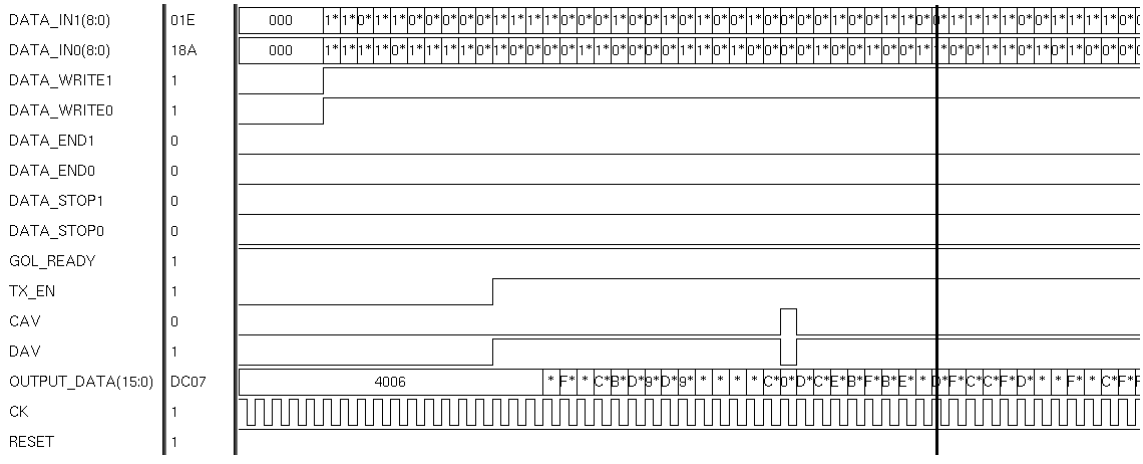
Timing 6: After running the Sample / Preload and Extest JTAG instructions, it is possible to set CARLOS outputs with a predefined value. In the figure all CARLOS outputs are asserted.



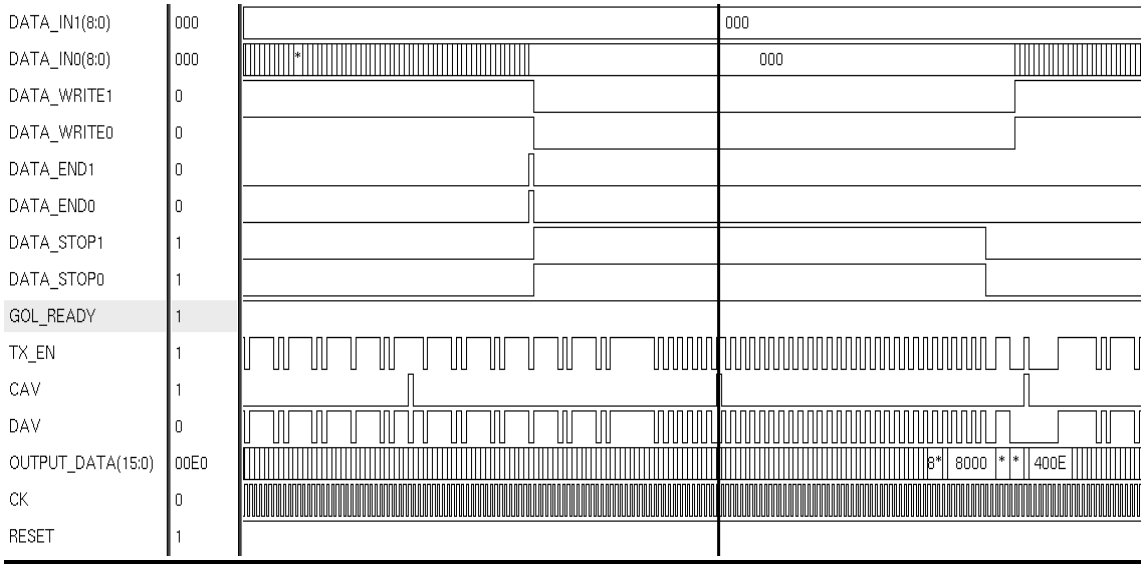
Timing 7: From JTAG mode to RUN mode



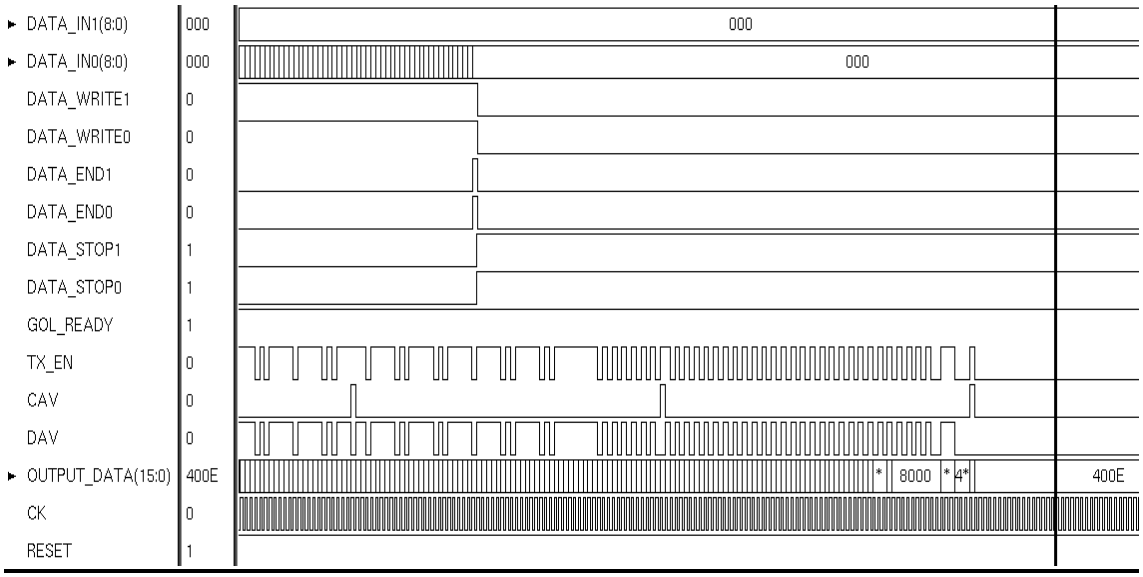
Timing 8: Receiving L0 command on the serial back-link and sending it back to AMBRA



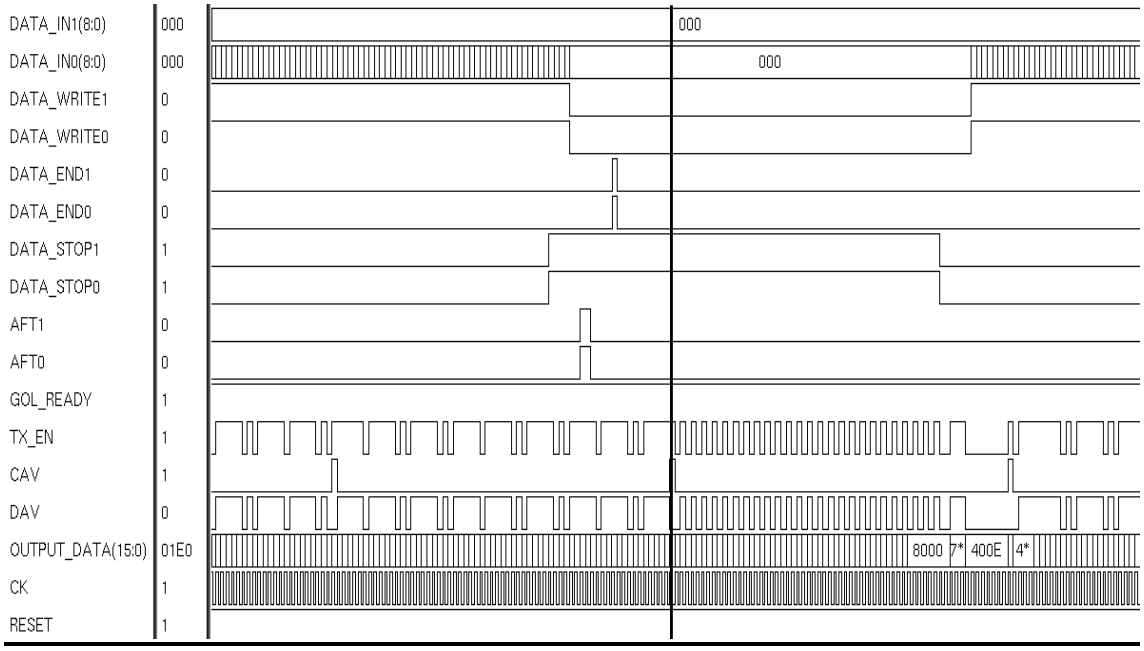
Timing 9: Event processing and transmission to the GOL chip



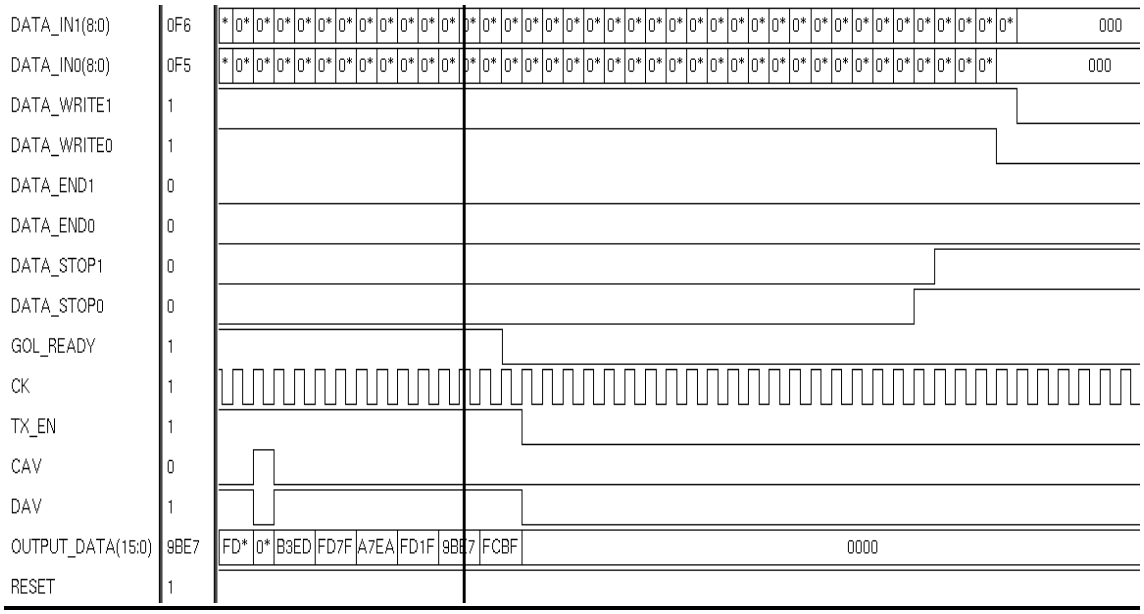
Timing 10a: End of a data packet transmission with a synchronization error (CARLOS receives less data than expected) and Stop If Error = 0.



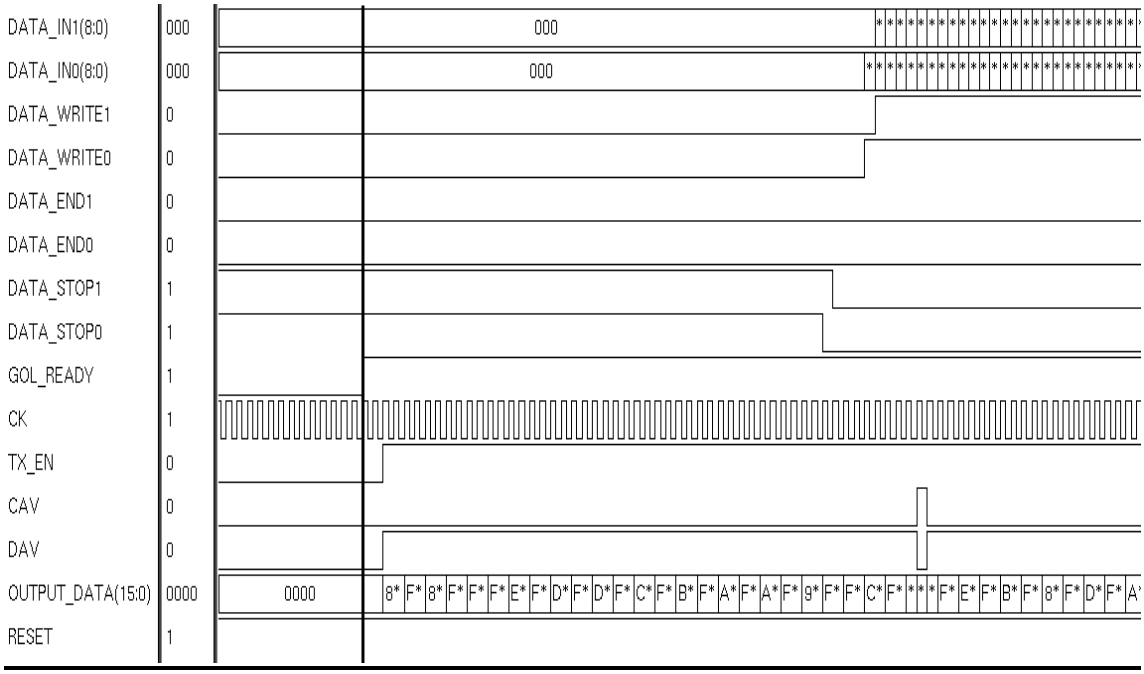
Timing 10b: End of a data packet transmission with a synchronization error (CARLOS receives less data than expected) and Stop If Error = 1.



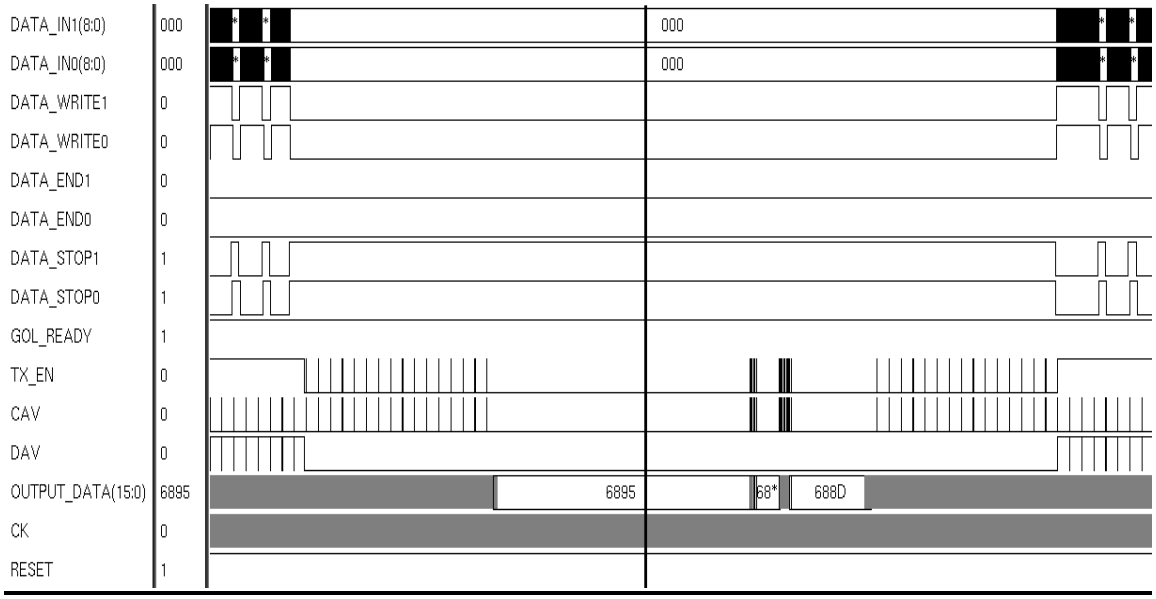
Timing 11: End of a data packet transmission with a synchronization error (CARLOS receives more data than expected).



Timing 12a: When the *gol_ready* signal is put to 0, CARLOS stops transmitting data towards the GOL chip.



Timing 12b: When the *gol_ready* signal is put back to 1, CARLOS restarts transmitting data towards the GOL chip and, after half of its internal FIFO locations get empty, AMBRA begins sending data again.



Timing 13: When CARLOSrx applies a back-pressure towards CARLOS, the *data_stop* signals are asserted. Then, if desired, CARLOS can be put in JTAG mode and some JTAG operations can be performed (the reading of an internal register in figure). Then CARLOS can be put in RUN mode again and data acquisition process restarted.

Managing synchronization errors

An event flush mechanism has been introduced in CARLOS v4 in order to manage error situations that may occur in the handshake between AMBRA and CARLOS. A flush signal from CARLOS to AMBRA has been foreseen with the purpose of deleting the current event on AMBRA and passing to the next one. This may be the case, for instance, of a SEU (Single Event Upset) over the Anode Length register on AMBRA or on CARLOS which may cause communication errors. Error situations like these are notified to CARLOSrx through the error flag word, so to stop data acquisition and check the values of the internal registers.
The flush signal is encapsulated in CARLOS output pins *aft1* and *aft0* that contain the following information:

1) abort (*aft1* and *aft0* stay active for 1 clock period): the abort signal is activated when L1reject or L2reject commands are received from CARLOSrx on the serial back-link;
2) flush (*aft1* or *aft0* stay active for 2 clock periods): the flush signal is activated when one CARLOS channel does not receive *data_end* = 1 within 4 clock periods after the expected time slot;
3) test pulse (*aft1* and *aft0* stay active for 3 clock periods): the testpulse signal is activated after receiving the related command on the serial back-link.

The highest priority is assigned to the abort signal, while the lowest priority is assigned to the testpulse: this means that if the 3 signals should occur at the same time, the first signal transmitted on *aft1* and *aft0* would be abort, followed by flush and then by test pulse.
The flush signal is activated just in one case: CARLOS expects the input *data_end* to be asserted for 1 clock cycle in coincidence with the last data coming from AMBRA, that is data number 256*(Anode_Length+1); if CARLOS does not receive *data_end* = 1 within 4 clock cycles after the expected period the flush signal is activated. As AMBRA receives the flush signal, it asserts the *data_end* signal for one clock period regardless of the *data_stop* value; this is just to inform CARLOS that AMBRA has received the flush signal, flushed the internal buffer and it is ready to send a new event.
8 different cases have been evaluated in the AMBRA – CARLOS communication, depending on the Stop If Error value:

Stop If Error = 1

1) *data_end* = 1 in the expected period
   - CARLOS asserts the *data_stop* signal towards the hybrid that sent the *data_end*;
   - the compressor goes on until it empties the RAM;

- after the last data coming from both hybrids have been received and both CARLOS FIFOs have been emptied, *data_stop* is put to 0 so to begin processing a new event.
- the *flush* signal is not activated.

2) *data_end* = 1 before the expected period
   - CARLOS asserts the *data_stop* signal towards the hybrid that sent the *data_end*;
   - CARLOS asserts the related error flag in the error flag word;
   - the compressor goes on until it empties the RAM;
   - after the faulty macro-channel has stopped generating data, CARLOS waits for the other macro-channel to receive the last data;
   - when both channels have sent all data, CARLOS leaves both *data_stop1* and *data_stop0* high until either CARLOS is reset or until CARLOSrx puts CARLOS in JTAG mode and reprogram the faulty registers.
   - the *flush* signal is not activated

3) *data_end* = 1 from 1 to 4 clock cycles after the expected period
   - CARLOS asserts the *data_stop* signal in the clock cycle it expects to receive the *data_end* signal (exceeding words coming from AMBRA are discarded);
   - CARLOS asserts the related error flag word and waits for AMBRA to put *data_write* low;
   - after receiving the *data_end* signal, the internal error flag signal stays high;
   - the compressor goes on until it empties the RAM;
   - after the faulty macro-channel has stopped generating data, CARLOS waits for the other macro-channel to receive the last data;
   - when both channels have sent all data, CARLOS leaves both *data_stop1* and *data_stop0* high until either CARLOS is reset or until CARLOSrx puts CARLOS in JTAG mode and reprogram the faulty registers.
   - the *flush* signal is not activated

4) *data_end* = 0 for > 4 clock cycles after the expected period
   - CARLOS asserts the *data_stop* signal in the clock cycle it expects to receive the *data_end* signal (exceeding words coming from AMBRA are discarded);
   - CARLOS asserts the related error flag and waits for AMBRA to put *data_write* low;
   - Since CARLOS does not receive the *data_en*d signal for 4 more clock cycles, it asserts the internal signal *flush*, that will be sent to the hybrid through the *aft1* or *aft0* output pin;
   - As AMBRA receives the flush signal, it asserts the *data_end* signal for one clock period regardless of the *data_stop* value;

- when CARLOS receives the *data_end* signal, the internal error flag signal is reset;
- the compressor goes on until it empties the RAM;
- after the faulty macro-channel has stopped generating data, CARLOS waits for the other macro-channel to receive the last input data;
- when both events have been completely received, CARLOS resets the *data_stop* signals informing AMBRA to be ready to accept new events.

N.B. If AMBRA does not send the *data_end* signal after a flush action, CARLOS internal error flag signal stays high and the *data_stop* signals are kept high after closing the data packet, until a reset is received or until CARLOSrx puts CARLOS in JTAG mode.

Stop If Error = 0

5) *data_end* = 1 in the expected period
   - CARLOS asserts the *data_stop* signal towards the hybrid that sent the *data_end*;
   - the compressor goes on until it empties the RAM;
   - after the last data coming from both hybrids have been received and both CARLOS FIFOs have been emptied, *data_stop* is put to 0 so to begin processing a new event.
   - the *flush* signal is not activated.

6) *data_end* = 1 before the expected period
   - CARLOS asserts the *data_stop* signal towards the hybrid that sent the *data_end*;
   - CARLOS asserts the related error flag in the error flag word (in 1 error flag only);
   - the compressor goes on until it empties the RAM;
   - after the faulty macro-channel has stopped generating data, CARLOS waits for the other macro-channel to receive the last data;
   - when both channels have sent all data, CARLOS put both *data_stop1* and *data_stop0* to 0 again, so that data transmission can continue.
   - the *flush* signal is not activated.

7) *data_end* = 1 from 1 to 4 clock cycles after the expected period
   - CARLOS asserts the *data_stop* signal in the clock cycle it expects to receive the *data_end* signal (exceeding words coming from AMBRA are discarded);
   - CARLOS asserts the related error flag word and waits for AMBRA to put *data_write* low;
   - after receiving the *data_end* signal, the internal error flag signal is reset;
   - the compressor goes on until it empties the RAM;

- after the faulty macro-channel has stopped generating data, CARLOS waits for the other macro-channel to receive the last data;
- when both channels have sent all data, CARLOS resets *data_stop1* and *data_stop0*, so that a new event can be fetched and processed.
- the *flush* signal is not activated

8) *data_end* = 0 for > 4 clock cycles after the expected period
   - CARLOS asserts the *data_stop* signal in the clock cycle it expects to receive the *data_end* signal (exceeding words coming from AMBRA are discarded);
   - CARLOS asserts the error flag in the forthcoming error flag word (only one!) and waits for AMBRA to put *data_write* low;
   - Since CARLOS does not receive the *data_en*d signal for 4 more clock cycles, it asserts the internal signal *flush*, that will be sent to the hybrid through the *aft1* or *aft0* output pin;
   - As AMBRA receives the flush signal, it asserts the *data_end* signal for one clock period regardless of the *data_stop* value;
   - the compressor goes on until it empties the RAM;
   - after the faulty macro-channel has stopped generating data, CARLOS waits for the other macro-channel to receive the last input data;
   - when both events have been completely received, CARLOS resets the *data_stop* signals informing AMBRA to be ready to accept new events.

Notes
In this paragraph AL stands for Anode Length + 1.

1. AL all AMBRAs < AL CARLOS
   The total number of EOR Summaries that CARLOS puts into an event is smaller than the expected number of 256. If, for example, anode length on AMBRA is 49 and its value on CARLOS is 50, the total number of CARLOS EOR Summaries is 250. In fact CARLOS receives 49*256 data words and frames them 50 at a time ((49*256)/50 = 250.88). So far finding a small number of EOR Summaries can give an indication of some problems to be solved.

2. AL all AMBRAs > AL CARLOS
   This is a situation in which the flush mechanism is activated. In this case when *enable 2D* = 1, the compressor block on CARLOS discards nearly every data of the last anode and the total number of counted EOR Summaries is 255. Otherwise, if *enable 2D* = 0, no data is lost and the number of EOR Summaries is 256.

3. AMBRA0: AL = n-1
   AMBRA1: AL = n-1
   AMBRA2: AL = n-1
   AMBRA3 (master): AL = n

CARLOS: AL = n
No synchronization error is found, but, for some time, CARLOS input data bus is not driven (high impedance).

4. AMBRA0: AL = n
   AMBRA1: AL = n
   AMBRA2: AL = n
   AMBRA3 (master): AL = n-1
   CARLOS: AL = n
   A synchronization error is found and, for some time, CARLOS input data bus is not driven (high impedance).

5. AMBRA0: AL = n
   AMBRA1: AL = n-1
   AMBRA2: AL = n-1
   AMBRA3 (master): AL = n-1
   CARLOS: AL = n
   A synchronization error is found, a bus contention is found in the transition from AMBRA0 to AMBRA1 and a period in which CARLOS input bus is in high impedance during the transition from AMBRA3 to AMBRA0.

6. AMBRA0: AL = n+2
   AMBRA1: AL = n+1
   AMBRA2: AL = n+1
   AMBRA3 (master): AL = n+1
   CARLOS: AL = n
   The flush mechanism is activated, a bus contention is found in the transition from AMBRA0 to AMBRA1 and a period in which CARLOS input bus is in high impedance during the transition from AMBRA3 to AMBRA0.

Backpressure from CARLOSrx to CARLOS

Since CARLOSrx has to concentrate different data streams coming from several detectors onto one DDL, it may need to stop the data flow from CARLOS for a while, until its internal FIFOs are empty and ready to accept data again. For this reason CARLOSrx can send to CARLOS two commands for stopping and restarting the data acquisition:

- Stop Acquisition;
- Restart Acquisition.

The Stop Acquisition command can be sent to CARLOS at every time and its effect is to assert the *data_stop1* and *data_stop0* signals towards AMBRA, so far stopping all the data transmission until a Restart Acquisition command is received.
If desired, after sending the Stop Acquisition command, CARLOSrx can also put CARLOS back in JTAG mode by sending the Enter JTAG mode command and perform some JTAG operations, such as write or read internal registers. In case only a JTAG read operation is performed, when restarting the acquisition, the event that was stopped before will be completely transmitted without any error. Obviously if some JTAG operation modifies CARLOS internal registers, such as thresholds, anode length or working mode, the event data transmission will result corrupted for what concerns the current event, while all the next coming events will be processed and transmitted correctly. An other JTAG operation to be avoided in the meantime after stopping the acquisition and restarting it is the BIST, since it would overwrite all the internal registers and FIFOs with new values and the current event would be corrupted.

Debugging CARLOS v4

Debugging facility

In order to ease the debugging of CARLOS, a multiplexer has been added with the aim of bringing in output (*test_output*) the value of interesting internal nets. The selection lines of the multiplexer can be driven using the *set_test* input bus.
The following table shows the multiplexer truth table:

set_test	***channel***	***test_output***
000	1	RAM a output (3:0) & RAM b output (3:0)
001	0	RAM a output (3:0) & RAM b output (3:0)
010	1	compressor output (7:0)
011	0	compressor output (20:13)
100	1	barrel output (7:0)
101	0	barrel output (29:22)
110	1	FIFO output (7:0)
111	0	FIFO output (14:7)

Table 8: multiplexer truth table for debugging purposes.

BIST

The BIST facility can be run via the serial back-link sending the JTAG instruction RUNBIST. It consists in feeding the 2 CARLOS macro-channels with 400 pseudo-random test-vectors and a 16-bit signature is produced as the test result. When run with the default values for the JTAG programmable values, the expected signature is 0x1990, otherwise its value changes. The signature value can be read out by running the JTAG instruction READ BIST.
If the chip passes the BIST test, it is highly probable that the chip works correctly.

Analyzing CARLOS v4

A C++ software tool has been designed with the aim of decoding CARLOS v4 outputs and helping test people to analyze data flowing out from the chip.

Why parsepack ?

A software tool is very useful in order to analyze in a very quick way very huge amounts of data. Parsepack main job consists in reconstructing CARLOS inputs from its actual outputs and comparing them to the data actually sent as inputs to the chip. Then parsepack has to compare the original data with the reconstructed ones and decide if there is some significant mismatch or not. This is not a trivial job since CARLOS 2D compression algorithm introduces a distortion on data, so a 1 to 1 comparison between original and reconstructed data can only be performed when the 2D compression is disabled. The two following criteria have been implemented in parsepack:

- every pixel in the reconstructed data is also present in the original one;
- every significant cluster is the original data is also present, with the same size and position, in the reconstructed data.

In this way it is possible to check very quickly if there are mismatches between the two data sets.

Parsepack also performs other useful tasks:

- performs the calculation of the compression coefficient for each channel.
  Its value is calculated in the following way:
  - the number of CARLOS output words with *dav* = 1 is computed per each event and per each channel;
  - the resulting number is incremented by 48 bits (taking into account for 3 header or footer words);
  - the resulting number is divided by the number of incoming bits (anode length * 256 * 8) in order to get the compression coefficient value.
- produces files with the original and reconstructed values;
- compares CARLOS actual output data with VHDL simulation data;
- decodes CARLOS JTAG words and creates a file jtag.dat containing the list of JTAG instruction followed by the related result;
- checks the 3 bits tck_counter in the JTAG words in order to verify that this value is incremented without any gap (errors are reported in jtag.dat),
- checks the 7 bits jtag_address in the JTAG words in order to verify that the correct device is sending the JTAG answer (errors are reported in jtag.dat);
- produces a file with the EOR Summaries for both channels and compares their values with the expected ones;
- produces a file with the list of error flag words.

How to get parsepack

The software tool parsepack2d.cpp source is available at the following Web site:
http://www.bo.infn.it/~falchier/carlos4.html
The following files have to be downloaded:

- parsepack2d.cpp;
- comp2d.cpp;
- gopt.cpp.

It can be compiled and run on different operating systems.

How to compile parsepack

Under the UNIX environment it can be compiled in the following way:

```
> gcc -g parsepack2d.cpp gopt.cpp -o parsepack
```

How to use parsepack

The SW tool parsepack can be run in the following way in order to get some information on how to use it:

```
> parsepack -h
-f [name] :        CARLOS output data
-c [name] :        VHDL output data
-j [name] :        JTAG commands
-d [name] :        input data
-al [num] :        anode length
-t1h [ch] [num] : high threshold
-t1l [ch] [num] : low threshold
-s [num] :         skip num lines
-enable2d [mode] :enable 2d compression
```

with the following meaning:

1) `-f [name] :` CARLOS output data

Output data is a file containing the following CARLOS outputs in binary format: data_stop1, data_stop0, trigger1, trigger0, aft1, aft0, tx_en, cav, dav, output_data. An output data file sample is given below.

```
0 0 0 0 0 0 1 0 1 1100000101000010
0 0 0 0 0 0 1 1 0 0000000010000000
0 0 0 0 0 0 0 0 0 1101000000010000
0 0 0 0 0 0 1 0 1 1000001101000010
0 0 0 0 0 0 0 0 0 1101000000010000
```

2) `-c [name] :` VHDL output data

VHDL output data contains the same data as CARLOS output data as they come from the VHDL simulation. This feature can be useful in order to compare actual chip outputs with the expected simulation values.

3) `-j [name] :` JTAG commands

A file containing the JTAG instructions run on the CARLOS chip can be used in order to ease the decoding of CARLOS JTAG output words. A sample of this file is shown below:

```
# AMBRA
A:WRITE_BASELINE 16
0
A:READ_BASELINE 16
9
# CARLOS
C:RUNBIST 5
0
C:READ_BIST 5
16
C:LOAD_T1L_LEFT 5
9
C:READ_T1L_LEFT 5
9
```

where:

- # is the first character of a comment line,
- P, A, C or G as the first character defines the chip who receives the related JTAG instruction (PASCAL, AMBRA, CARLOS or GOL),
- the number on the right of a JTAG instruction defines the length of the related JTAG instruction (16 bits on PASCAL and AMBRA, 5 bits on CARLOS and GOL),
- the number on the line following a JTAG instruction denies the number of bits to be read during the Shift-DR state (0 means that the Shift-DR state is not entered after decoding the JTAG instruction).

4) `-d [name] :` input data

Input data file contains the data words sent as inputs to CARLOS in binary format in the following order:

- an incremental number (it is incremented by 1 after each line);
- data_in1 (9 bits);
- data_in0 (9 bits);
- data_write1;

- data_write0;
- data_end1;
- data_end0.

An output data file sample is given below:

```
75433 000000000 011001000 1 1 0 0
75434 000000000 011010000 1 1 0 0
75435 000000000 011011000 1 1 0 0
75436 000000000 000000000 1 1 0 0
75437 000000000 001000000 1 1 0 0
```

5) `-al[num] :  anode length`

Anode length defines the number of samples to be taken for each input anode. Its value is the same as the one JTAG programmed on the corresponding internal register of CARLOS incremented by 1. So far if CARLOS has been programmed with an anode length value of 199, when using parsepack an anode length value of 200 has to be used.

6) `-t1h [ch] [num] : high threshold`

This parameter defines the high threshold for the two CARLOS processing channels. Its value is the same as the one JTAG programmed on the chip internal registers.

6) `-t1l [ch] [num] : low threshold`

This parameter defines the low threshold for the two CARLOS processing channels. Its value is the same as the one JTAG programmed on the chip internal registers.

7) `-s [num] :  skip num lines`

This parameter allows to skip 1 output data file line out of 2, when num is put to 1. Otherwise all the lines are processed. This feature may be useful when oversampling CARLOS output data, for instance when using the logic Analyzer internal clock.

8) `-enable2d [mode] : enable 2d compression`

This parameter has the same value of the CARLOS internal register enable 2D.

Just as an example the parsepack tool could be run with the following command:

```
> parsepack -f ../testbench/bench_out.tv
```

```
-j ./jtag_list.tv.carlos -d ../testbench/bench_in.tv -al
50 -t1h 1 31 -t1h 0 0 -t1l 1 15 -t1l 0 0 -enable2d 1 >
log.txt
```

What are parsepack outputs?

This is a list of the files produced as outputs:

- log.txt:
  it contains the result of the comparison of the original matrix with the reconstructed one for each event and, when they are present, a list of the mismatches. It also gives the compression coefficient and the number of EOR Summaries found for each event. Follow a brief sample with errors:

```
Event: 1
Event ID: 0
Carlos ID: 3
Ch. 0 Ratio 0.803919 Words 7958
      Low threshold 0 High threshold 0 Anode length 50
      End of row summaries: 255
Ch. 1 Ratio 17.728532 Words 358
      Low threshold 15 High threshold 31 Anode length 50
      End of row summaries: 255
The cluster coordinates in original not found in reconstructed event on
channel 0 are:
Anode: 255 Sample: 11 Value: 160
Anode: 255 Sample: 12 Value: 192
Anode: 255 Sample: 13 Value: 200
```

  and an other one without errors:

```
Event: 1
Event ID: 0
Carlos ID: 3
Ch. 0 Ratio 0.830198 Words 7706
      Low threshold 0 High threshold 0 Anode length 50
      End of row summaries: 256
Ch. 1 Ratio 17.630854 Words 360
      Low threshold 15 High threshold 31 Anode length 50
      End of row summaries: 256
Every cluster in original is found in reconstructed event on channel 0
Every cluster in original is found in reconstructed event on channel 1
Every cluster in reconstructed event is found in original on channel 0
Every cluster in reconstructed event is found in original on channel 1
```

- jtag.dat:
  This file contains the JTAG instructions sent to CARLOS and the related answers, both in hexadecimal and binary format.
  Follows a brief sketch:

```
# AMBRA
```

```
A:WRITE_BASELINE 1
A:READ_BASELINE 1
000000000 h0000
# CARLOS
C:RUNBIST 1
C:READ_BIST 1
0001100110010000 h1990
C:LOAD_T1L_LEFT 1
000000000 h0000
C:READ_T1L_LEFT 1
000000000 h0000
```

So far the baseline value read from the AMBRA chip is 0 and the BIST result read from CARLOS is 0x1990.

- event#_ch0.dat, decoded#_ch0.dat, event#_ch1.dat, decoded#_ch1.dat
  The files event*.dat contain the original events, while the files decoded*.dat contain the reconstructed events for each channel (# is the ordinal number of the event being considered). The files contain one value for each line, starting from anode 0 – sample 0 to anode 0 – sample 255 and then anode 1 –sample 0 and so on.

- packetdiff.dat
  This file contains the list of the lines with some difference between the CARLOS actual outputs and VHDL simulation outputs.

- headfoot.dat
  This file contains a list of the headers (all 3 words) and footers (only one is reported in the file per event) of all the events contained in the output data file, without the 3 MSBs. Follows a brief sample:

```
Event: 4
001E
001E
001E
1FFF

Event: 5
0026
0026
0026
1FFF
```

- errflag.dat
  The file contains a list of all the error flag words present in the output data file in hexadecimal format.

- errsummary0.dat, errsummary1.dat

The files contain the list of the EOR Summaries detected in every event contained in the output data file in a binary format. The summaries, reported one per line, have the following format when enable 2D = 1:

000	E/O	VL	VZ	VH	NCL	NCZ	NCH	Parity
3 bits	1 bit	1 bit	1 bit	1 bit	3 bits	4 bits	4 bits	3 bits

When enable 2D = 0, the summaries have the following format:

17 zeros	E/O	Parity
14 bits	1 bit	3 bits

Parsepack checks the value of each of the fields contained in the EOR Summaries with the exception of the Parity bits. If some mismatch is found, a list of errors is appended to the file after the EOR Summaries.

- compr0.dat, compr1.dat
  These files contain the values of the valid outputs of the *compressor* block hosted in channel 0 and channel 1 during the transmission of an event. Their values can be useful for debugging purposes only.
- barrel0.dat, barrel1.dat
  These files contain the values of the valid outputs of the *barrel shifter* hosted in channel 0 and channel 1 during the transmission of an event. Their values can be useful for debugging purposes only.
- fifo0.dat, fifo1.dat
  These files contain the values of the valid outputs of the *fifo_data* block hosted in channel 0 and channel 1 during the transmission of an event. Their values can be useful for debugging purposes only.