Alma Mater Studiorum – Università di Bologna

DOTTORATO DI RICERCA IN FISICA

Ciclo XXI

Settore/i scientifico disciplinari di afferenza: FIS/01

Hardware and software development of a multichannel readout board named CARLOSrx for the ALICE experiment

Presentata da: **Filippo Costa**

Coordinatore Dottorato

Prof. Fabio Ortolani

Relatore

Prof. Enzo Gandolfi

Correlatore:

Dott. Davide Falchieri

Esame finale anno 2009

this page intentionally left blank

Alma Mater Studiorum – Università di Bologna

DOTTORATO DI RICERCA IN FISICA

Ciclo XXI

Settore/i scientifico disciplinari di afferenza: FIS/01

Hardware and software development of a multichannel readout board named CARLOSrx for the ALICE experiment

Presentata da: **Filippo Costa**

Coordinatore Dottorato

Prof. Fabio Ortolani

Relatore

Prof. Enzo Gandolfi

Correlatore:

Dott. Davide Falchieri

Esame finale anno 2009

this page intentionally left blank

Table of Contents

Introduction

This thesis describes the readout electronics developed for the **SDD** (*Silicon Drift Detector*), one of the **ALICE** (*A Large Ion Collider Experiment*)[1] sub-detectors.

ALICE**,** that is an experiment held at **CERN** (*European Organization for Nuclear Research*) using the **LHC** (*Large Hadron Collider*), is specialized in analyzing lead-ion collisions. ALICE will study the properties of quark-gluon plasma, a state of matter where quarks and gluons, under conditions of very high temperatures and densities, are no longer confined inside hadrons. Such a state of matter probably existed just after the **Big Bang**, before particles such as protons and neutrons were formed.

The SDD detector is part of the ITS (*Inner Tracking System*), that is composed by 6 cylindrical layers with the innermost one attached to the beam pipe. The ITS tracks and identifies particles near the interaction point, it also aligns the tracks of the particles detected by more external detectors. The two ITS middle layers contain the whole 260 SDD detectors.

A multichannel readout board, called CARLOSrx, receives at the same time the data coming from 12 SDD detectors. In total there are 24 CARLOSrx boards needed to read data coming from all the SDD modules (detector plus front end electronics).

CARLOSrx packs data coming from the front end electronics through optical link connections, it stores them in a large data **FIFO** (*First In First Out*) and then it sends them to the **DAQ** (*Data Acquisition*) system.

Each CARLOSrx is composed by two 9U x 400mm VME64 boards:

- **CARLOSrx data,** that reads data coming from the SDD detectors and configures the **FEE** (*Front End Electronics*).
- **CARLOSrx clock**, that sends the clock signal to all the FEE.

All the 24 CARLOSrx boards have been installed in **3 VME crates** and from the backplane of these crates they get power supply and firmware informations to re-configure the **FPGA**s (*Field Programmable Gate Array*) installed on them.

This thesis contains a description of the hardware design and firmware features of both CARLOSrx data and CARLOSrx clock boards, which deal with all the SDD readout chain.

All the work done during my PhD studies has been concentrated in 2 main areas:

- **firmware development**, writing the code for CARLOSrx firmware;
- **software development**, writing software tools necessary to test and configure the front end electronics.

The thesis is organized as follows.

The first chapter contains a description of the ALICE experiment, especially for

what concerns the SDD detector. A description of the FEE and of the DAQ system will be presented at the end of this section.

The second chapter describes in details all the major hardware components installed on CARLOSrx boards and it reviews their working mechanism.

The third chapter presents the code of the CARLOSrx firmware, focusing on the algorithms used and on the logical blocks to implement them.

The fourth chapter contains the description of the software developed in order to test the firmware code, the FEE and to configure all the SDD readout chain.

Finally the last chapter describes the tests performed at CERN before and after the installation of the SDD modules and the readout electronics in the ALICE experiment. This chapter presents also the first physics results obtained during the cosmic rays tests performed in the 2008.

1. The ALICE experiment

The main experiments held at **CERN** using the **LHC**[2] are:

- **ALICE**
- ATLAS (*A Toroidal LHC ApparatuS*)
- CMS (*Compact Muon Solenoid*)
- LHCb (*Large Hadron Collider beauty experiment*)
- TOTEM (*Total Cross Section Elastic Scattering and Diffraction Dissociation*)
- LHCf (*Large Hadron Collider forward*)

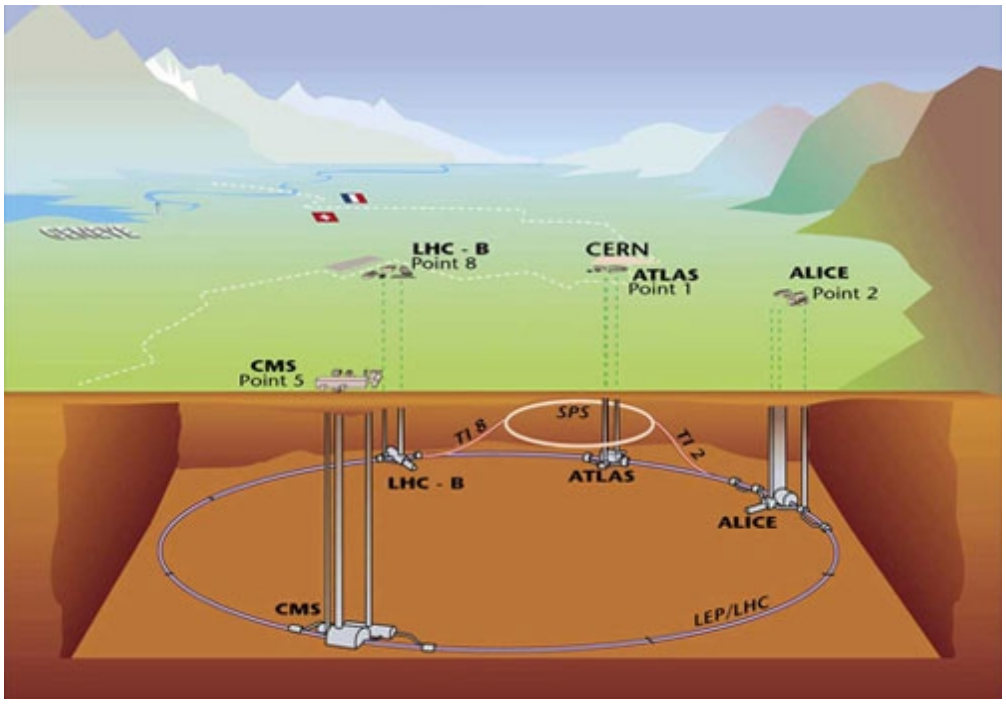


Fig. 1.1: the experiments at the LHC.

Four of them, ALICE, ATLAS, CMS, LHCb, are extremely big detectors and they are installed in four huge caverns situated in different points in the LHC ring (fig. 1.1), the other two are smaller in size and they are attached to CMS (TOTEM), and ATLAS (LHCf).

One of the main goals addressed by the LHC[3] experiments is the connection between phase transitions involving elementary quantum fields, fundamental symmetries of nature and the origin of mass. Theory draws a clear distinction between symmetries of the dynamical laws of nature (i.e. symmetries and particle content of the

Lagrangian) and symmetries of the physical state with respect to which these dynamical laws are evaluated (i.e. symmetries of the vacuum or of an excited thermal state). The experimental programme at the LHC addresses both aspects of the symmetry-breaking mechanism through complementary experimental approaches.

ATLAS and CMS will search for the **Higgs** particle, which is supposed to generate the mass of the electroweak gauge bosons and the bare mass of elementary fermions through spontaneous breaking of the electroweak gauge symmetry. They will also search for supersymmetric particles which are manifestations of a broken intrinsic symmetry between fermions and bosons in extensions of the **Standard Model**.

LHCb, focusing on precision measurements with heavy **b quarks**, will study CP-symmetry violating processes, that measure the misalignment between gauge and mass eigenstates which is a natural consequence of electroweak symmetry breaking via the **Higgs** mechanism.

ALICE will study the role of chiral symmetry in the generation of mass in composite particles (hadrons) using heavy-ion collisions to attain high-energy densities over large volumes and long timescales. ALICE will investigate equilibrium as well as non-equilibrium physics of strongly interacting matter in the energy density regime $\varepsilon \sim$ 1–1000 GeV fm^{-3} . In addition, the aim is to gain insight into the physics of parton densities close to phase-space saturation, and their collective dynamical evolution towards hadronization (confinement) in a dense nuclear environment. In this way, one also expects to gain further insight into the structure of the **QCD phase** diagram and the properties of the **QGP phase**. The focus of heavy-ion physics is to study **nuclear matter** under conditions of **extreme density and temperature** trying to understand how collective phenomena and macroscopic properties emerge from the microscopic laws of elementary-particle physics.

ALICE will operate in several different running modes with significantly different characteristics. The experiment has been primarily designed to run with **heavy ions beams**, which are characterized by:

- **relatively low rates** (interaction rates <= 10 kHz for Pb-Pb beams at design luminosity of L=1027 $cm^{-2}s^{-1}$),
- **relatively short running time** (order of few weeks per year),
- **very high multiplicity and correspondingly large event size**.

When it is in **pp** running mode:

- **the interactions rates are much higher** than in heavy-ion runs (up to 200 kHz),
- **the event size is small**,
- **the running time is typically of several months per year** in pp mode.

In general, to establish experimentally the collective properties of the hot and dense matter created in nucleus–nucleus collisions, both systematics and luminosity-dominated questions have to be answered at LHC. ALICE aims firstly at accumulating sufficient integrated luminosity in Pb–Pb collisions at $\sqrt{s} = 5.5$ TeV per nucleon pair, to measure rare processes such as jet transverse-energy spectra up to $E_t \sim 200$ GeV and the pattern of medium induced modifications of bottomium bound states. However, the interpretation of these experimental data relies considerably on a systematic comparison with the same observables measured in proton–proton and proton–nucleus collisions as well as in collisions of lighter ions. In this way, the phenomena truly indicative of the hot equilibrating matter can be separated from other contributions. The successful completion of the heavy-ion programme thus requires the study of pp, pA and lighter A–A collisions in order to establish the benchmark processes under the same experimental conditions. In addition, these measurements are interesting in themselves. For example, the study of lighter systems opens up possibilities to study fundamental aspects of the interaction of colour-neutral objects related to non-perturbative strong phenomena, like confinement and hadronic structure. Also, due to its excellent tracking and particle identification capabilities, the ALICE pp and pA programmes complement those of the dedicated pp experiments.

The layout of the ALICE set-up is shown in figure 1.2.

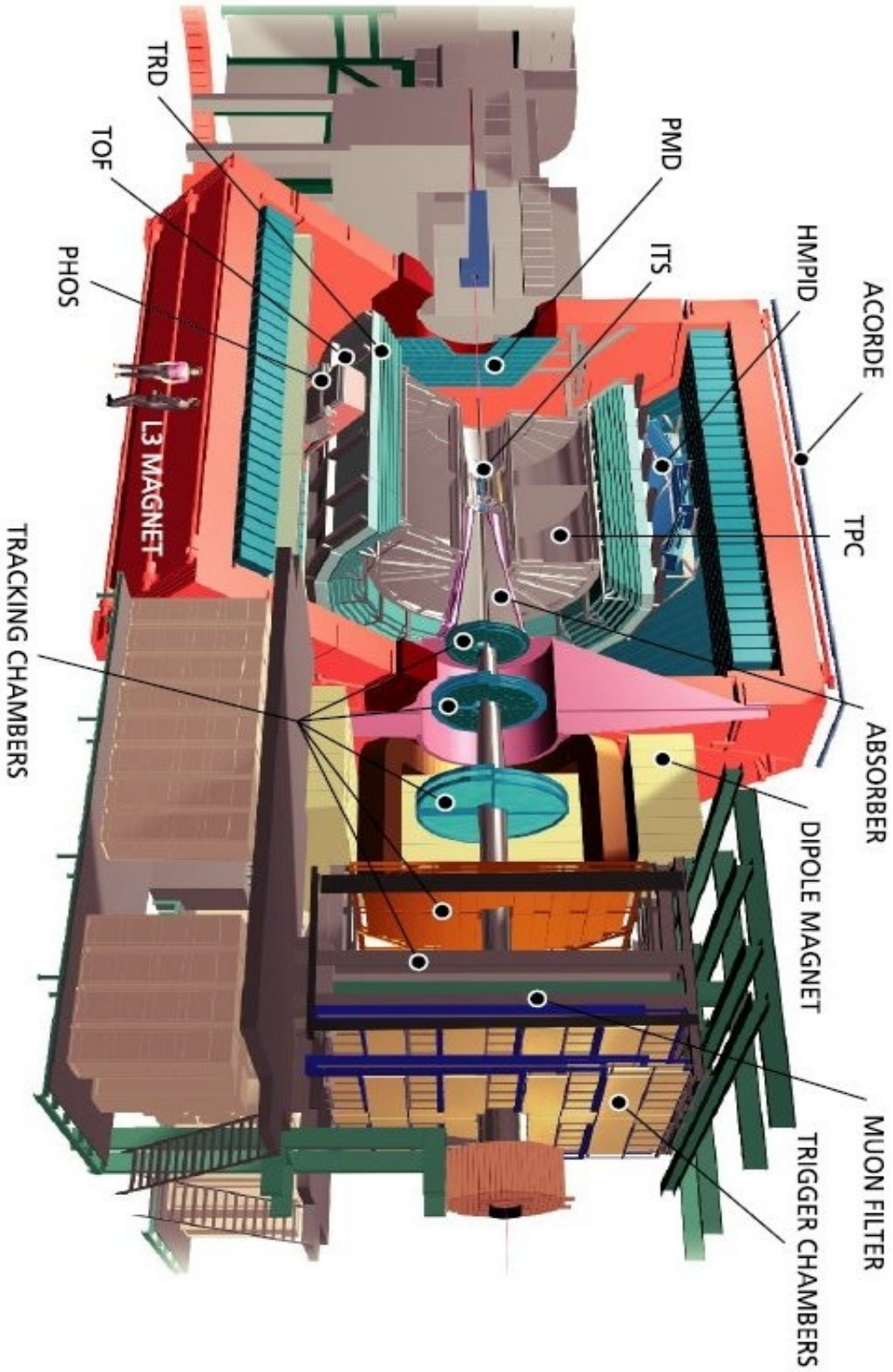


Fig. 1.2: Longitudinal section of the ALICE detector

In figure 1.3 the layout of Point 2 is shown :

- **CR1** (*Counting Room*) **DAQ**: in this room all the DAQ PCs and disks array have been installed. They are needed to acquire data from the readout electronics of different detectors and to store the informations waiting to be moved in **CASTOR** (*Cern Advanced STORage manager*).
- **CR2 HLT** (*High Level Trigger*): a farm of PCs decides on-line which events must be recorded.
- **CR3 DCS** (*Detector Control System*): a software framework controls the status of the complete experiment, rising alarms in case of problems.
- **CR4 detector electronics**: all the CARLOSrx boards have been installed in **3 VME crates** situated in this room. The boards are connected with optical fibers to the SDD detectors and to the DAQ.
- **ALICE cavern**: where the experiment takes place. Here all the SDD modules have been installed in the ALICE detector.

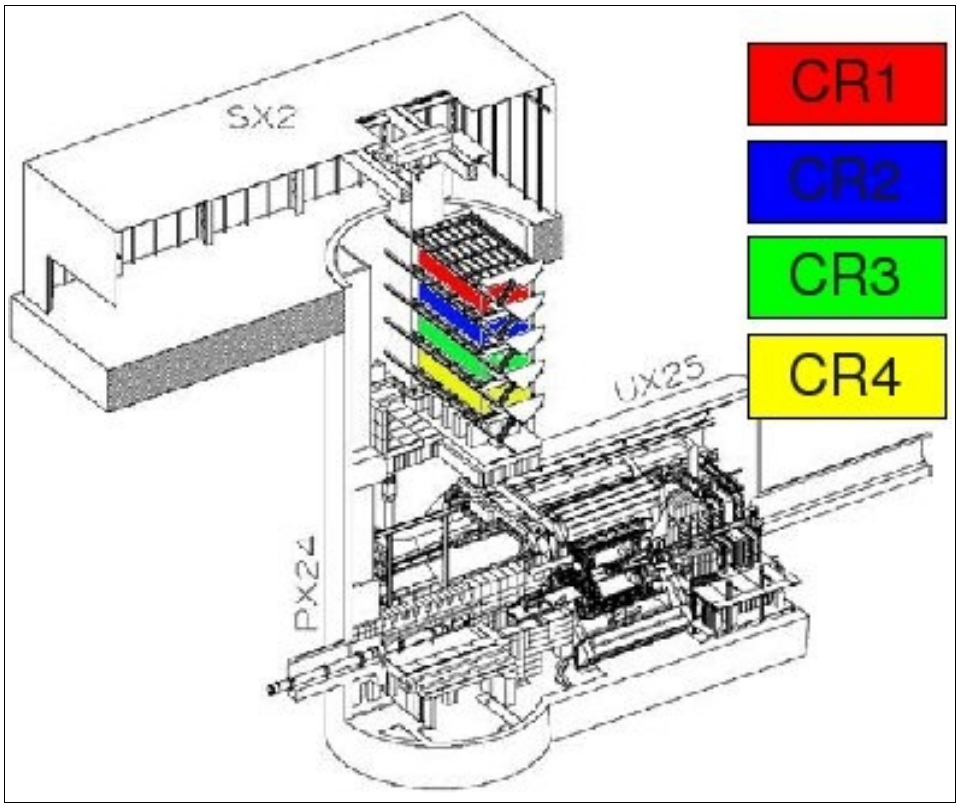


Fig. 1.3: Point 2 where the ALICE experiment takes place

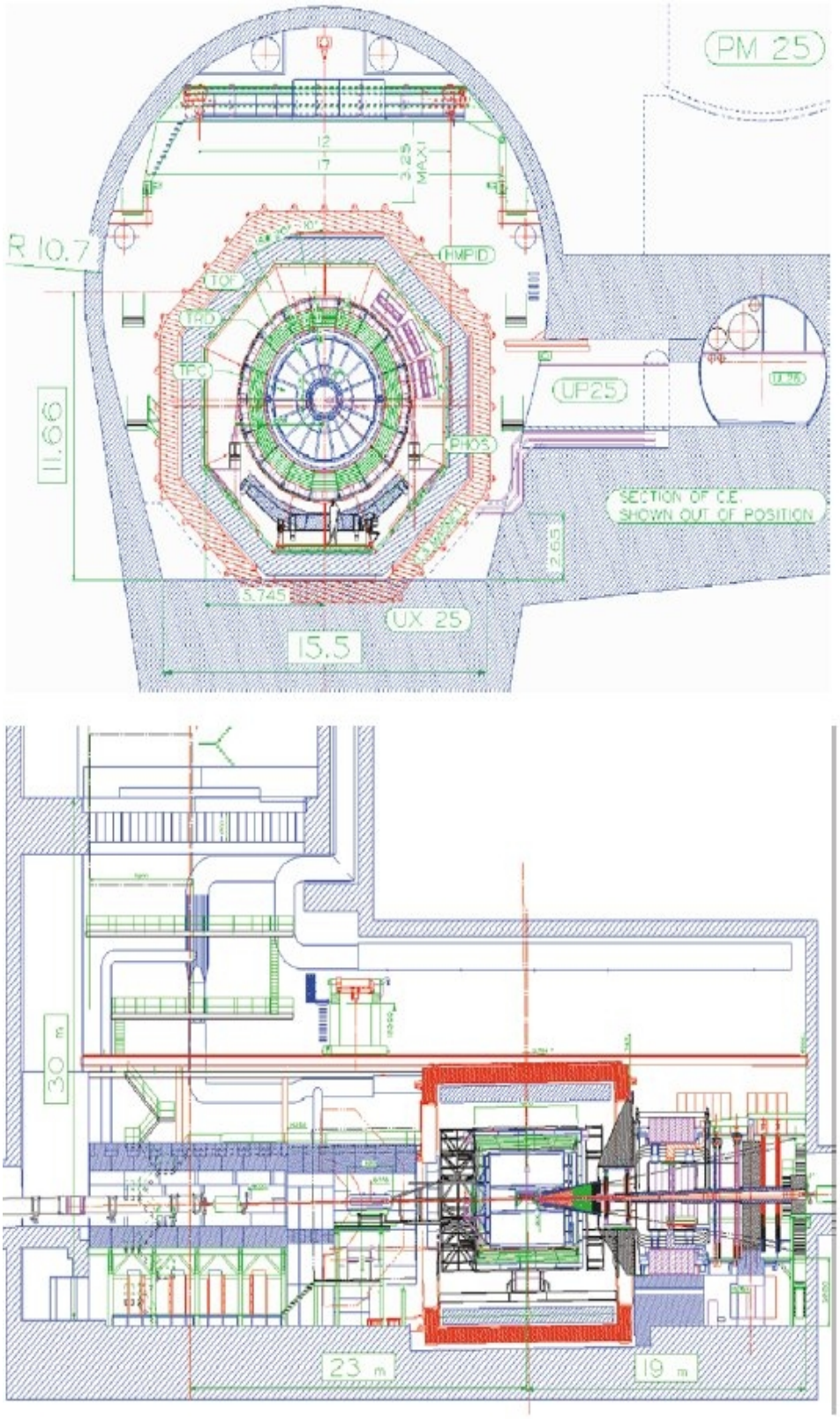


Fig. 1.4: frontal and lateral section of the Point 2 ALICE cavern.

1.1 ITS

Tracking and particle identification relies on different high-granularity detectors (Fig. 1.5):

- **ITS** (*Inner Tracking System)* :
  - **SPD** *(Silicon Pixel Detector).*
  - **SDD** *(Drift Detector).*
  - **SSD** *(Strip Detector).*
- **TPC** (*Time-Projection Chamber*).
- **TRD** (*High-granularity Transition-Radiation Detector*).
- **TOF** (*High resolution array Time Of Flight*).
- ***HMPID*** (*High-Momentum Particle Identification Detector*).
- **PHOS** (*PHOton Spectrometer*).
- ***FMD*** (*Forward Multiplicity Detector*)**.**
- ***V0*****.**
- ***T0*****.**
- **PMD** (*Photon Multiplicity Detector*)**.**
- **ZDC** (*Zero-Degree Calorimeters*).
- **EMCAL** (*ElectroMagnetic CALorimeter*).
- **CPV** (*Charged Particle Veto*)**.**
- **Muon Trigger, Tracking.**
- **ACORDE** (*ALICE Cosmic Ray DEtector*).

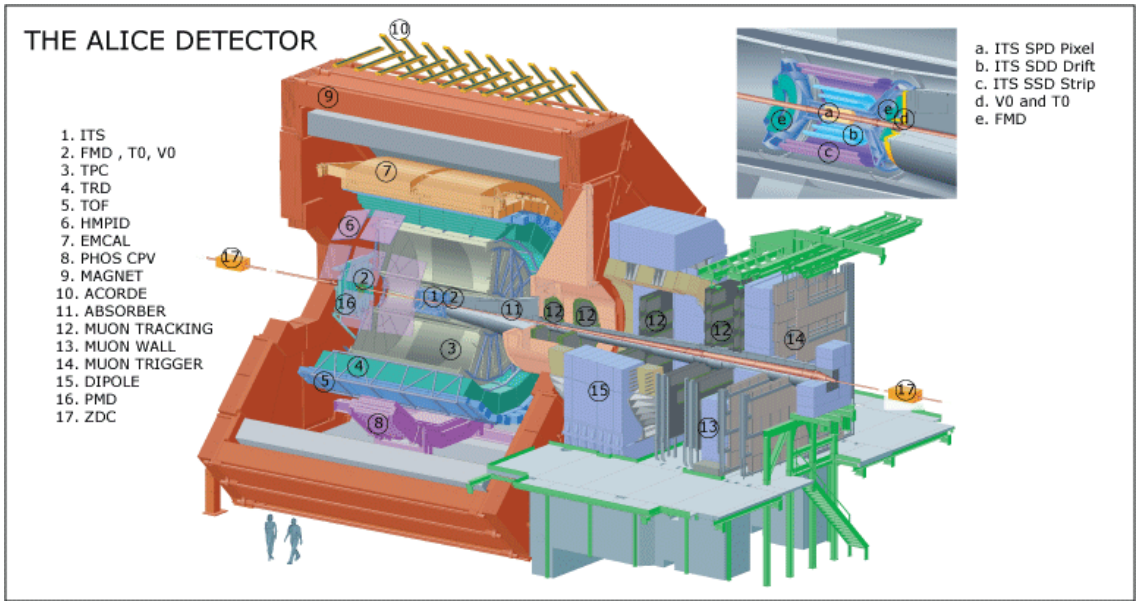


Fig. 1.5: ALICE sub-detectors

The *Inner Tracking System*[3] consists of six cylindrical layers of silicon detectors (Fig. 1.6).

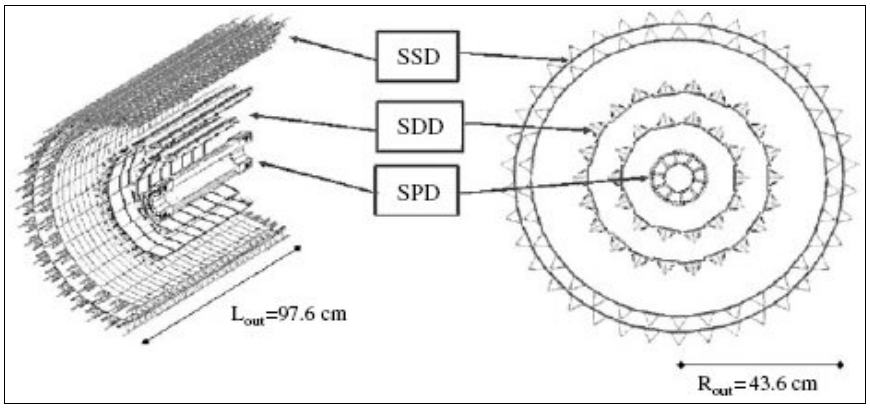


Fig. 1.6: ITS layout

It covers the rapidity range of $|\eta| < 0.9$ for all vertices located within the length of the interaction diamond ($\pm 1\sigma$), i.e. 10.6 cm along the beam direction. The outer radius is determined to match tracks with those from the TPC, and the inner radius is the minimum allowed by the radius of the beam pipe (3 cm). The first layer has a more extended coverage ($|\eta| < 1.98$) to provide, together with the *Forward Multiplicity Detectors*, a continuous coverage in rapidity for the measurement of charged-particles multiplicity.

The basic functions of the **ITS** are:

- to localize the primary vertex with a resolution better than 100 μm;
- to reconstruct the secondary vertices from decays of hyperons and D and B mesons;
- to track and identify particles with momentum below **100 MeV**;
- to improve the momentum and angle resolution for the high-pt particles which also traverse the TPC;
- to reconstruct particles traversing dead regions of the TPC.

Starting from the innermost 2 layers the detectors of the ITS are (Fig. 1.6):

- *Silicon Pixel Detector*: particle density, up to 80 particles cm^{-2},
- *Silicon Drift Detector*: particle density, up to 7 particles cm^{-2},
- *Silicon Strip Detector*: particle densities, below 1 particle cm^{-2}.

With the exception of the two innermost pixel planes, all layers are equipped with analogue readout for particle identification via dE/dx measurements in the non-relativistic region. This will give the ITS a stand-alone capability as a low-pT particle spectrometer.

Layer	**Type**	**r (cm)**	**±z (cm)**	**Area (m^2)**	**Ladders**	**Lad./stave**	**Det./ladder**	**Channels**
1	**Pixel**	3.9	14.1	0.07	80	4	1	3276800
2	**Pixel**	7.6	14.1	0.14	160	4	1	6553600
3	**Drift**	15.0	22.2	0.42	14	-	6	43008
4	**Drift**	23.9	29.7	0.89	22	-	8	90112
5	**Strip**	37.8/38.4	43.1	2.09	34	-	22	1148928
6	**Strip**	42.8/43.4	48.9	2.68	38	-	25	1459200
Total area				6.28				

Tab. 1.1: Dimensions of the ITS detectors (active areas).

Parameter	Silicon PIXEL	Silicon DRIFT	Silicon STRIP
Spatial precision rϕ (μm)	12	38	20
Spatial precision z (μm)	100	28	830
Two track resolution rϕ (μm)	100	200	300
Two track resolution z (μm)	850	600	2400
Cell size (μm^2)	50x425	150x300	95x40000
Active area per module (mm^2)	12.8x69.6	75.5x75.3	73x40
Readout channels per module	40960	2x256	2x768
Total number of modules	240	260	1698
Total number of readout channels (k)	9835	133	2608
Total number of cells (M)	9.84	23	2.6
Average occupancy (inner layer) (%)	2.1	2.5	4
Average occupancy (outer layer) (%)	0.6	1.0	3.3
Power dissipation in barrel (W)	1500	1060	1100
Power dissipation end-cap (W)	500	1750	1500

Tab. 1.2: Parameters of the various detector types. A module represents a single sensor element.

1.2 SDD detector

The middle two layers of the ITS are equipped with SDD[4] (Fig. 1.7), because they couple a very good multi track capability with dE/dx information. SDDs, like gaseous drift detectors, exploit the measurement of the transport time of the charge deposited by a transversing particle to localize the impact point in two dimensions, thus enhancing resolution and multi-track capability at the expense of speed. A linear SDD has a series of parallel implanted p^+ field strips, connected to a voltage divider on both surfaces of the high-resistivity n-type silicon wafer. The voltage divider is integrated on the detector substrate itself. The field strips provide the bias voltage to fully deplete the volume of the detector and they generate an electrostatic field parallel to the wafer surface, thus creating a drift region (figure 1.7). Electron-hole pairs are created by the charged particles crossing the detector. The holes are collected by the nearest p^+ electrode, while the electrons are focused into the middle plane of the detector and driven by the drift field towards the edge of the detector where they are collected by an array of anodes composed of n^+ pads. So far an electronic charge cloud drifts from the impact point to the anode region: the cloud shows a bell-shaped Gaussian distribution that, owing to the diffusion and mutual repulsion, during the drift becomes smaller and larger. In this way a charge cloud can be collected by one or more anodes depending on the charge released by the ionizing particle and on the impact position with respect to the anode region. The small size of the anodes, and hence their small capacitance (50 fF), implies low noise and good energy resolution.

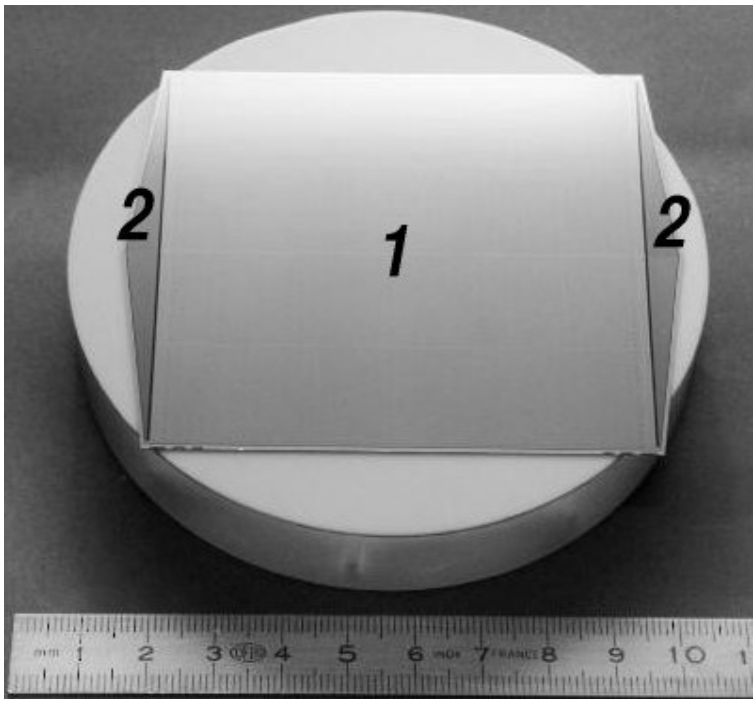


Fig. 1.7: SDD detectors (**1 sensible area – 2 guard area**)

1.2.1 SDD layout

The ALICE SDDs have been produced from very homogeneous *high-resistivity 300 μm* thick *Neutron Transmutation Doped* (**NTD**)[2] *silicon*. They have a sensitive area of 70.17 x 75.26 mm^2 (point 1 in figure 1.7) and a total area of 72.50 x 87.59 mm^2 (point 1+2 in figure 1.7). The sensitive area is split into two drift regions by the central cathode strip to which a nominal bias of -2.4 kV is applied. In each drift region, and on both detector surfaces, 291 p^+ cathode strips, with 120 μm pitch, fully deplete the detector volume and generate a drift field parallel to the wafer surface (Fig. 1.8). To keep the biasing of the collection region independent on the drift voltage, a second bias supply of -40V is added (Fig. 1.8). The degrading of the high voltage to the zero potential of the detector boundary is implemented by two insensitive guard regions biased by 145 cathode strips with 32 μm pitch. To improve the detector reliability, all the drift and guard regions have their own built-in voltage dividers. Their total power dissipation is 1W per detector and it is removed by an appropriate air circulation system.

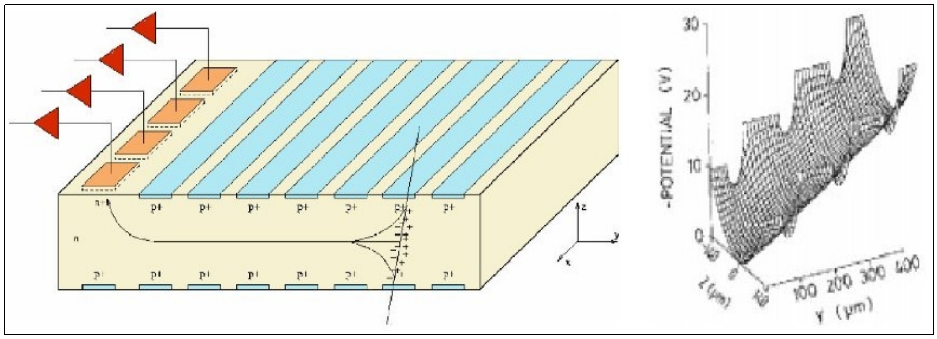


Fig. 1.8: working principle of a SDD detector

Each drift region has 256 collection anodes with 294 μm pitch and three rows of 33 point-like (20x100 μm^2) MOS charge injectors to monitor the drift velocity which depends on temperature: $vdrift \propto T^{-2.4}$.

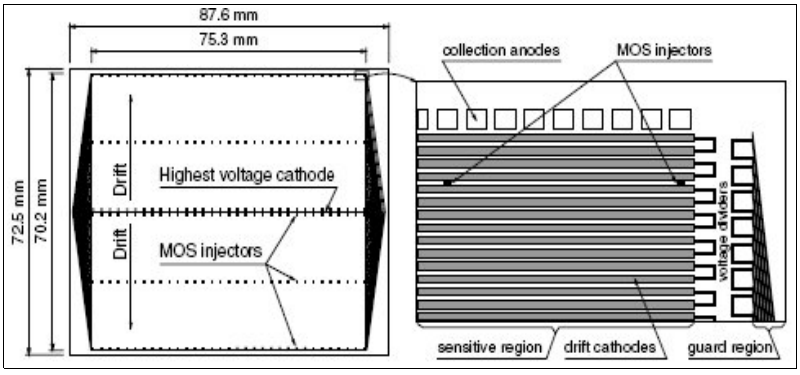


Fig. 1.9: layout of SDD (MOS charge injector are showed)

These devices inject charge in the silicon sensor when they are triggered externally. By measuring the time between injection of this charge and its arrival at the anodes the drift-speed can be calibrated. Applying this procedure at regular intervals during the gap between the LHC orbits ensures a proper calibration at all times. At the nominal bias voltage of -2.4 kV the drift velocity is 8.1 μm ns^{-1}. Since the front-end electronics samples the signal of each anode at a frequency of 40.08 MHz, the size of the sensitive element (cell) is 294 x 202 μm^2 , corresponding to $89.1x10^3$ cells per detector,which are readout by 512 channels. The space precision along the drift direction is better than 38 μm over the whole detector surface. The precision along the anode axis (z) is better than 30 μm over 94% of the detector surface and reaches 60 μm close to the anodes where a smaller fraction of clusters affect more than one anode. The detection efficiency is larger than 99.5% for amplitude thresholds as high as 10 times the electronic noise. The relative distance at which two clusters are disentangled with a 70% efficiency grows almost linearly from 600 μm near the anodes to 800 μm at the maximum drift distance.

The main parameters of the ALICE SDD are summarized in table 1.3.

Sensitive area	70.17 x 75.26mm^2
Collection anodes (readout channels)	2 x 256
Total area	72.50 x 87.59mm^2
Anode pitch	294 μm
Nominal operating voltage	−2.4 kV
Nominal bias of the collection region	−40V
Nominal drift velocity	8.1 μm ns^{-1}
Nominal maximum drift time	4.3 μs
Cell size at nominal drift velocity	294x202 μm^2
Cells per detector at nominal drift velocity	2x256x174
Total number of cells (260 SDDs)	$23.16x10^6$
Average resolution along the drift (rϕ)	35 μm
Average resolution along the anode (z)	25 μm
Detection efficiency	99.5%
Average double-track resolution at 70% efficiency	700 μm

Tab. 1.3: SDD detector

Silicon drift sensors provide true two-dimensional position information (Fig. 1.10). Along one side of the sensor 256 anodes, collect the charge drifting perpendicularly to the beam direction. So far each SDD detector contains 2 x 256 readout channels: taking into account that the layer 3 and 4 contain 260 SDD modules, the total number of SDD readout channels is around 133k.

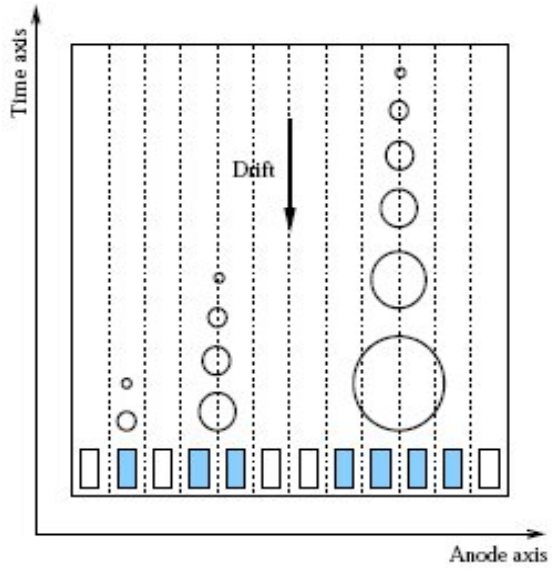


Fig. 1.10: Charge distribution evolution scheme

1.2.2 SDD layer

The SDDs and the front-end electronics are mounted on linear structures called **ladders** (Fig. 1.11). There are 36 ladders in total:

- **14 ladders with 6 detectors each on layer 3**.
- **22 ladders with 8 detectors each on layer 4**.

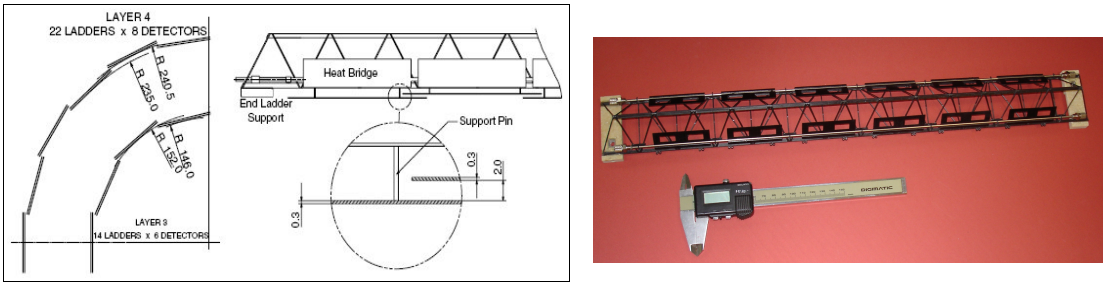


Fig. 1.11 : Ladders for layer 3 - 4

The layers sit at the average radius of 14.9 and 23.8 cm from the beam pipe. The main geometrical parameters of the SDD layers and ladders are summarized in table 1.4. The ladder space frame is a lightweight triangular truss made of *Carbon-Fibre Reinforced Plastic* (**CFRP**) and has a protective coating against humidity absorption. The ladders are assembled on a CFRP structure made of a cylinder, two cones and four support rings (figure 1.12).

	Layer 3	Layer 4
Detectors per ladder	6	8
Ladders per layer	14	22
Detectors per layer	84	176
Ladder sensitive half-length (cm)	22.16	29.64
Ladder length (cm)	45.56	60.52
Average layer radius (cm)	15.03	23.91
Ladder space-frame weight (g)	11	15
Weight of ladder components (g)	87	121

Tab. 1.4: SDD layer 3 and layer 4 geometrical parameters

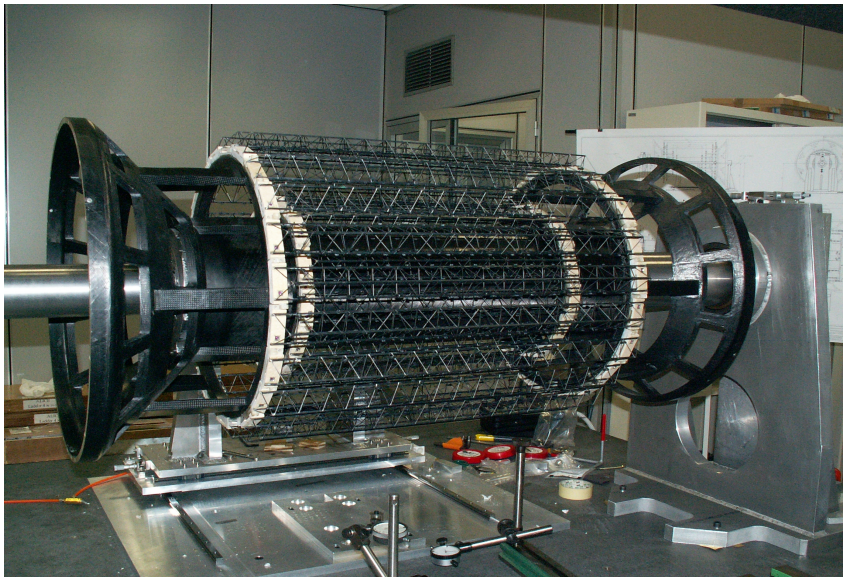


Fig. 1.12: SDD barrel

1.3 Readout chain

For each SDD detector the readout electronics is composed by (figure 1.13):

- **8 PASCAL-AMBRA** chip pairs connected directly to the silicon detector (a module is composed by the SDD detector and the 8 **P-A** chip pairs).
- **1 CARLOS** board, to receive data coming from 1 module and to apply on them a compression algorithm to reduce the data volume.
- **1 multichannel readout board CARLOSrx** that stores all the informations coming from 12 CARLOS and sends them to a PC for future elaboration.
- **LDC** (*Local Data Concentrator*)**/GDC** (*Global Data Collector*)**,** these are component of the DAQ infrastructure. They are standard PCs that receive the data coming from the readout electronics and save them on the disk or different storage media, like CASTOR.

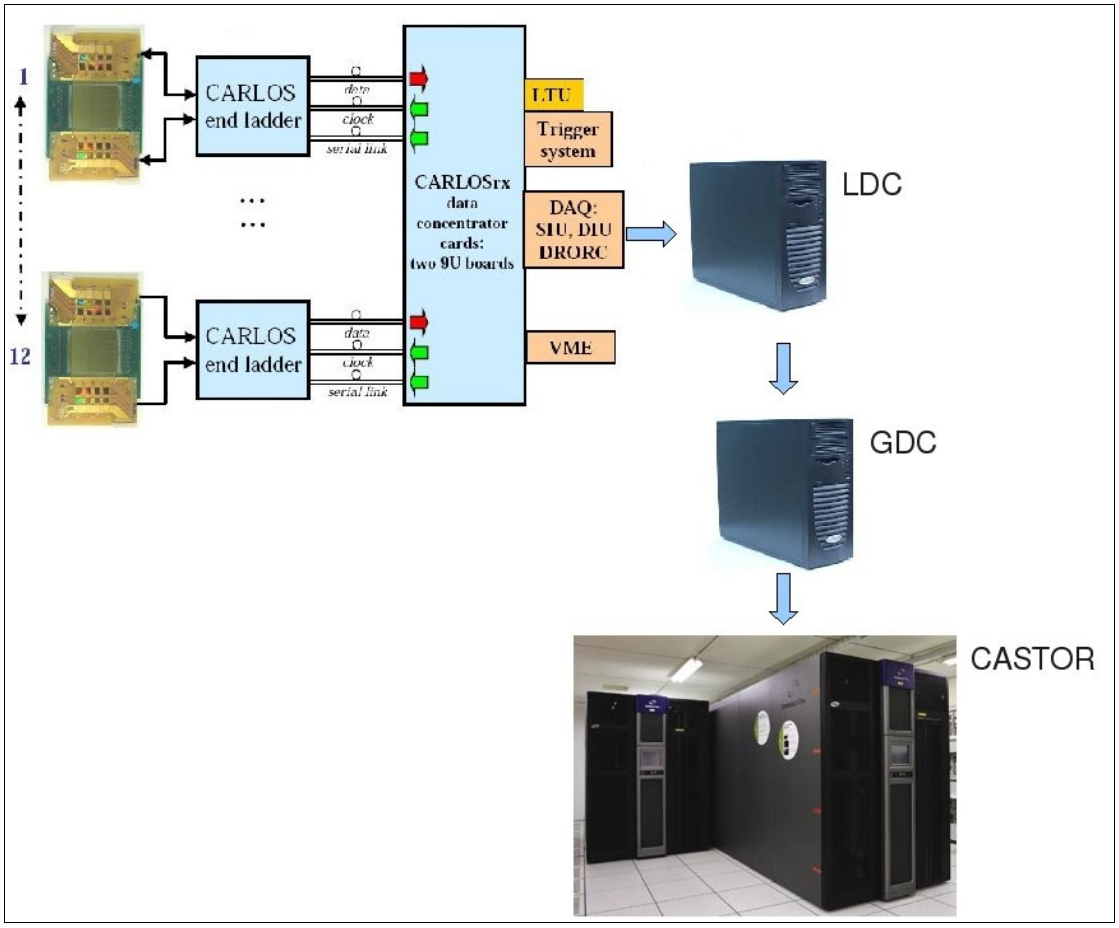


Fig. 1.13: SDD readout chain

The full SDD readout chain is made by:

- **260 SDD** detectors distributed in two layers.
- **2080 P-A chip pairs.**
- **260 CARLOS** boards.
- **24 CARLOSrx** board pairs (data/clock).
- **4 LDCs** (6 CARLOSrx for each one).
- **1 GDC.**

1.3.1 Front End module (Pascal-Ambra)

The SDD front-end electronics is based on three **ASIC**s (*Application Specific Integrated Circuit*):

- **PASCAL**,
- **AMBRA**,
- **CARLOS**.

The first one, PASCAL[3] (Fig. 1.14) assembled on the front-end hybrid, contains three functional blocks:

- preamplifier,
- analogue storage,
- *Analogue-to-Digital Converter* (**ADC**).

The second integrated circuit, AMBRA[3] (Fig. 1.14), also on the hybrid, is a digital four-event buffer which performs:

- data derandomization,
- baseline equalization on an anode-by-anode basis,
- 10 to 8-bit non linear data compression,
- sends the data to the third ASIC, CARLOS, which it is a zero-suppressor and data compressor mounted in one of the end-ladder boards (will be described in the next section).

The connections between the detectors and the front-end electronics have been assured with flexible micro cables, **TAB bonded**, which carry both data and power supply lines (Fig. 1.15). Each detector is assembled together with its front-end electronics and high-voltage connections as a unit, hereafter called a module, which has been fully tested before it has been mounted on the ladder.

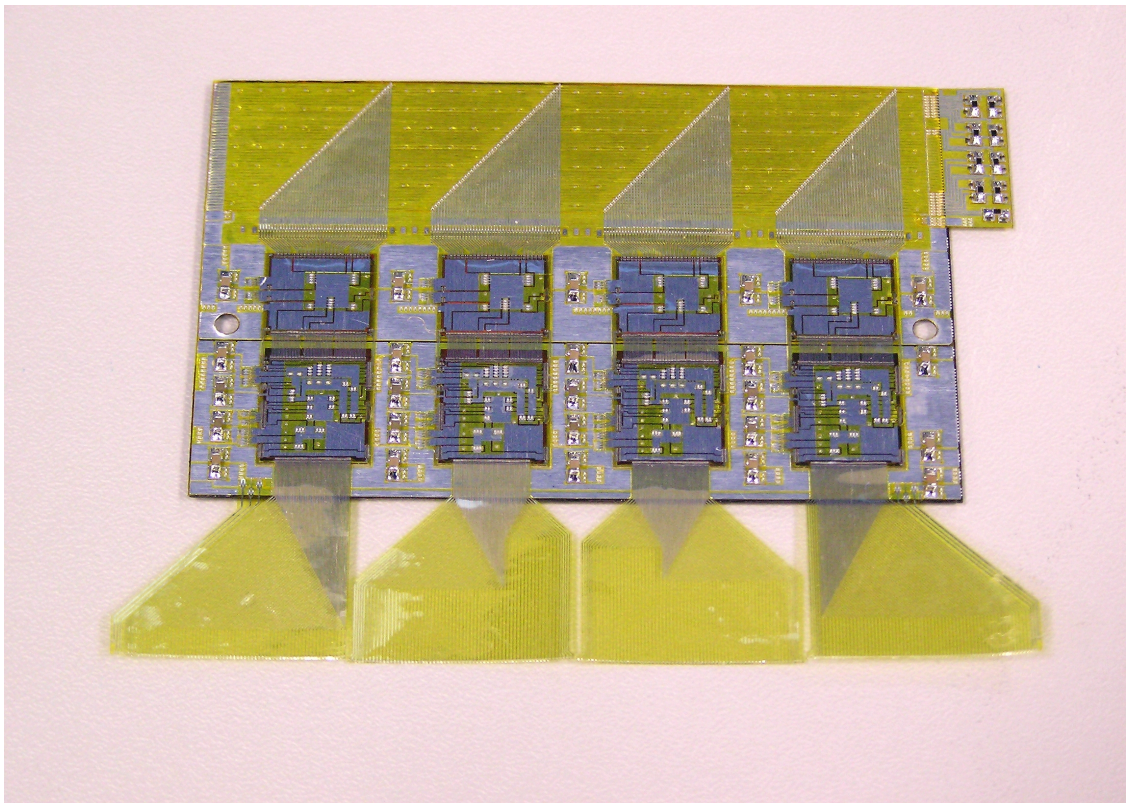


Fig. 1.14.: 4 PASCAL-AMBRA chip pairs installed on front end hybrid

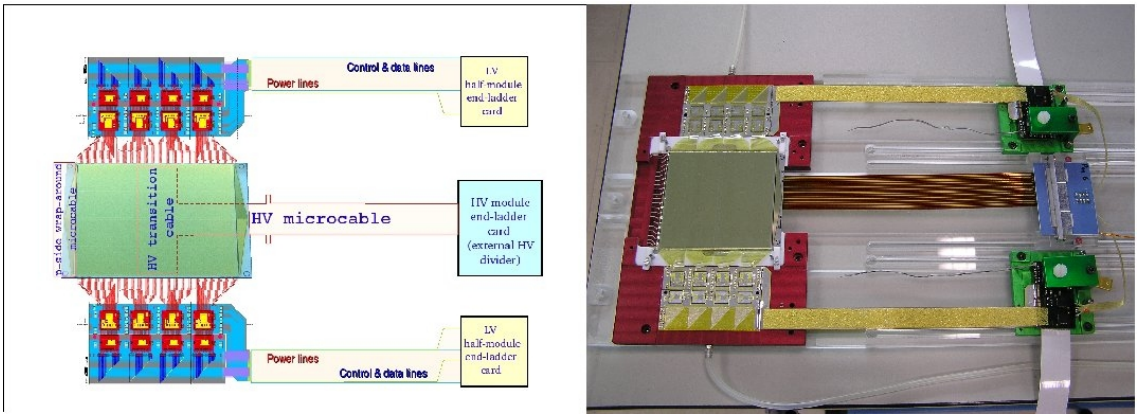


Fig. 1.15 : SDD module (SDD detector with front end electronics)

The three ASICs have been designed using a radiation-tolerant layout technique (enclosed gate geometry) based on a commercial deep sub-micron process (0.25 μm). The PASCAL prototype designed with this technology has proved to be insensitive to total ionization dose up to 300 kGy. The average power dissipation of each PASCAL–AMBRA front-end channel is estimated to be about 6mW. The signal generated by an SDD anode feeds the PASCAL trans-impedance amplifier-shaper which has a peaking time of about 40 ns and a dynamic range of 32 fC (the charge released by an 8-MIP particle hitting near the anode). The amplifier output is sampled at 40.08 MHz by a ring analogue memory with 256 cells per anode. This is the mode of operation in the idle state of the front-end. On a run-by-run basis, PASCAL can be programmed (during the JTAG configuration) to use half of this nominal frequency for the sampling, thus reducing the amount of data and, therefore, the sub-system dead-time. Analysis of beam test data and simulation have shown that the cost in terms of both spatial resolution and double track-resolution is negligible. The advantages of a front-end A/D conversion are the noise immunity during signal transmission, and the possibility of inserting a multiple-event buffer to reduce the busy time of acquisition, to derandomize the data and, therefore, to slow down the transfer rate to the DAQ system. This greatly reduces the material budget of the cabling. The digitization lasts for about 230 μs (120 μs when the half frequency mode is programmed) and can be aborted by the absence of the L1 trigger or by the arrival of an L2-reject signal; in both cases, the front-end electronics reset the SDD BUSY and returns to the idle state within 100 ns. On the successful completion of the analogue-to-digital conversion the SDD BUSY is reset if at least one buffer is still available in the AMBRAs. As soon as the conversion is completed, all the AMBRAs transmit the data reading one buffer at the time to the CARLOS chips on the end-ladders, an operation which takes 1.64 ms (0.82 ms when the half frequency mode is programmed).

1.3.2 CARLOS

The end-ladder modules, called CARLOS[19], are located at the ends of each ladder. They receive data coming from the modules and perform data compression by means of a two-dimensional two-thresholds algorithm and with no additional dead time, the CARLOS chips reduce the SDD event size from the raw 22.1 MB by more than one order of magnitude and they send these informations to the CARLOSrx board.

Major operations of this ASIC are:

- to compress data coming from one Silicon Drift Detector using a double threshold compression algorithm (for details on the compression algorithm refer to **Appendix C.1**),
- to send data to the concentrator board CARLOSrx via a **800Mb/s** optical link,
- to receive through optical link:
  - clock,
  - reset,
  - JTAG informations.

All these signals and data are propagated to the FEE.

CARLOS has to face several constraints due to the position where the board has been installed:

- size (54x49 mm) and a maximum thickness of 16 mm,
- radiation tolerance.

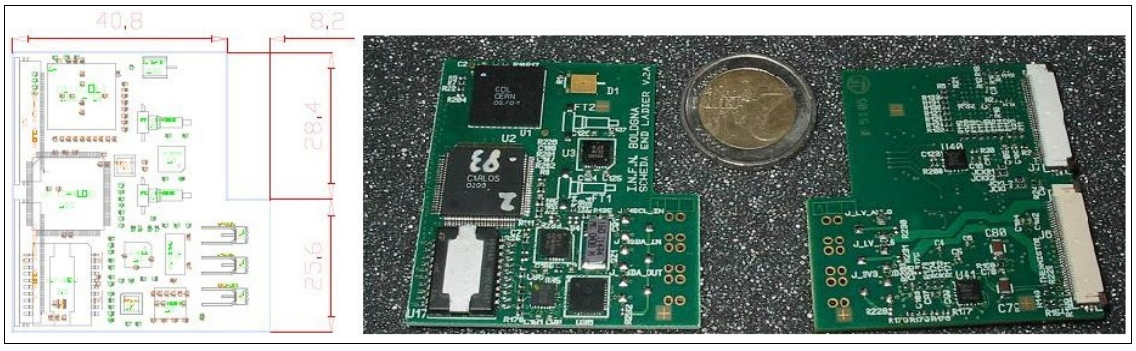


Fig. 1.16: CARLOS board

The board has been built in Bologna using several CERN developed ASICs.

CARLOS[17] receives 2x8-bit bus in input, one for the half module-left and the other one for the right, and generates a 16-bit output bus using *8B/10B Ethernet protocol* encoded by the **GOL** (*Gigabit Optical Link*). Data are sent to a single mode optical fiber using *1310nm* optical laser with a a total data throughput of 800 Mb/s.

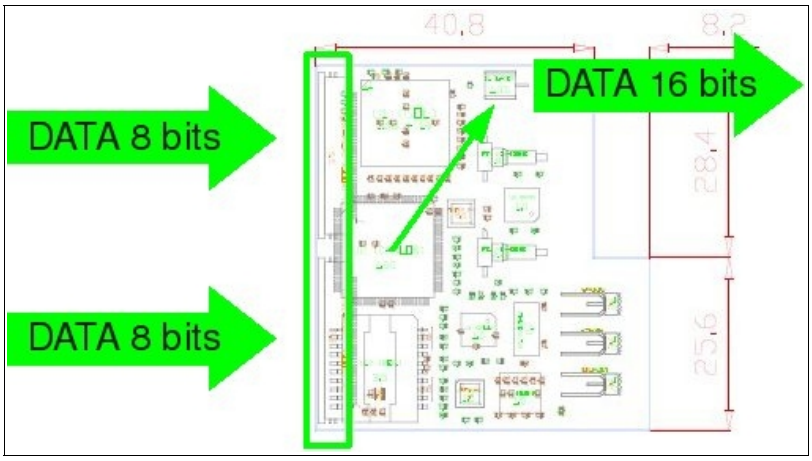


Fig. 1.17: CARLOS data flow

Figure 1.17 shows the layout of the CARLOS board: it is possible to see the components that handle data and clock. Looking at the left of figure 1.17 it is possible to see:

- the 1310 nm edge-emitting laser diode that sends data to CARLOSrx,
- a pin diode that receives the clock,
- a pin diode that:
  - receives system configuration at startup phase,
  - receives trigger signals,
  - stops the acquisition in case of event congestion.

As already mentioned all the components embedded into the CARLOS end ladder boards are ASICs designed to be radiation-tolerant to the total ionizing dose that has been estimated for 10-years of data taking in the ALICE-ITS environment, nearly 30 krads.

1.4 DAQ system

In this section a general description of the DAQ system[5] and its components will be presented. The hardware part of the data acquisition infrastructure of ALICE is composed by:

- **LDC,**
- **GDC,**
- **CASTOR.**

The software components that control the data acquisition are:

- **DATE** (*ALICE Data Acquisition and Test Environment*)**,**
- **ECS** (*Experiment Control System*).

A view of the ALICE data-acquisition architecture is illustrated in Fig. 1.18.

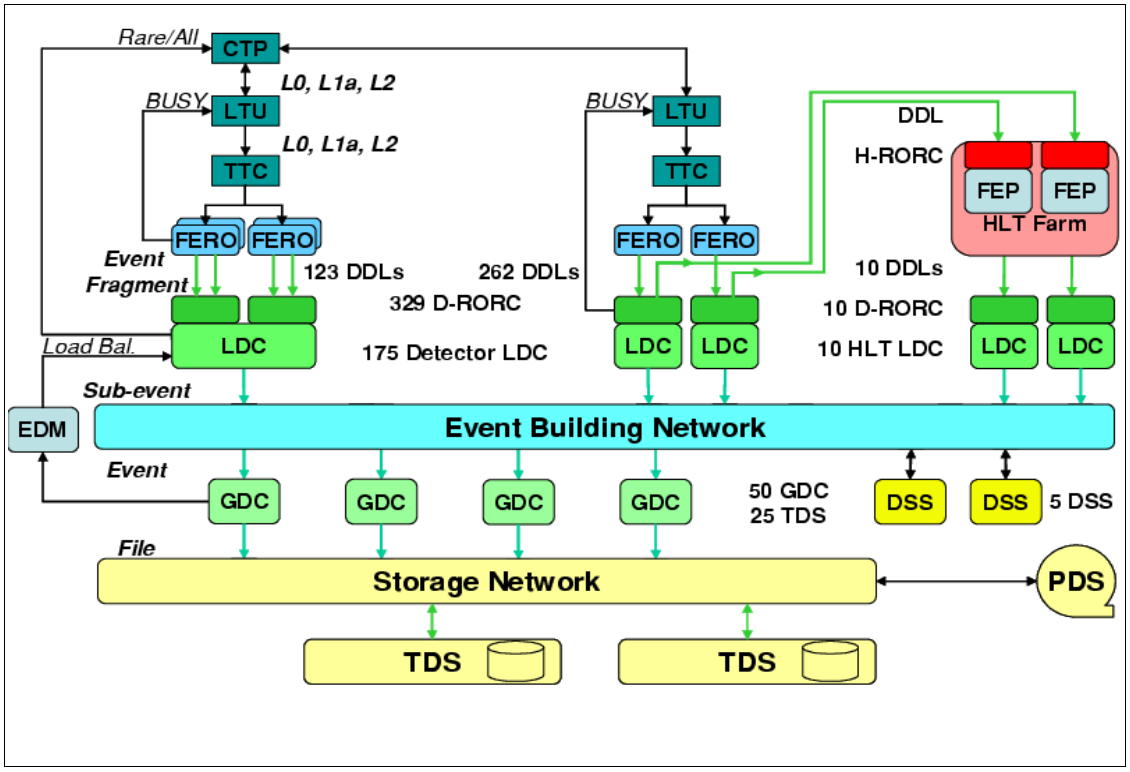


Fig. 1.18 : ALICE DAQ.

The detectors receive the trigger signals and the associated informations from the *Central Trigger Processor* (**CTP**)[18], through a dedicated *Local Trigger Unit* (**LTU**)[12] interfaced to the *Timing, Trigger and Control* (**TTC**) system. The readout electronics of all the detectors is interfaced to the ALICE standard *Detector Data Links* (**DDL**). The data produced by the detectors (event fragments) are injected on the DDLs. At the

receiving side of the DDLs there are PCI boards, called *DAQ Read-Out Receiver Cards* (**D-RORC**). The D-RORCs are hosted by PCs, named LDCs. Each LDC can handle one or more D-RORCs. In the LDCs, the event fragments originated by the various D-RORCs are logically assembled into sub-events. The role of the LDCs is to ship the sub-events to a farm of PCs called GDCs, where the whole events are built (from all the sub-events pertaining to the same trigger). Besides having a DDL common to all the sub-detectors, the other major architectural feature of the ALICE data acquisition is the event builder, which is based upon an event building network. The sub-event distribution is performed by the LDCs, which decide the destination of each sub-event. This decision is taken by each LDC independently from the others (no communication between the LDCs is necessary); the synchronization is obtained using a data-driven algorithm. The algorithm is designed to fairly share the load on the GDCs. The event-building network does not take part in the decision about the destination; it is a standard communication network supporting the **TCP/IP protocol**. The role of the GDCs is to collect sub-events and to assemble them into whole events. The GDCs also feed the *Transient Data Storage* (**TDS**) located at the experimental area with the events that, eventually, will be migrated onto P*ermanent Data Storage* (**PDS**), **CASTOR**, in the computing center. All these "players" are controlled by software tools that coordinate each single operations. In particular there are DATE and ECS that have been developed to start and to stop the acquisition. DATE is the official acquisition software used by ALICE to acquire data from all the detectors. The general idea is to have a distributed program running into all the LDCs that reads the data coming from the D-RORCs installed in each LDC. The DATE software can be used in a little test setup to simulate a run like the real one in the experiment, in this way all the detector teams can use the latest software release in their test setup to acquire data. With DATE it is possible to control only the acquisition part, but for the other components of the system, like the trigger or the **DCS** (*Detector Control System*), other programs are requested to start and to stop the run. For this reason the DAQ team has provided the ECS. The main feature of this program is to have the control of all the players during the acquisition, so it takes care to start the trigger and the data acquisition and check if the system is working properly only clicking one button; it also stops the run if one of the parameters being checked goes below a certain threshold.

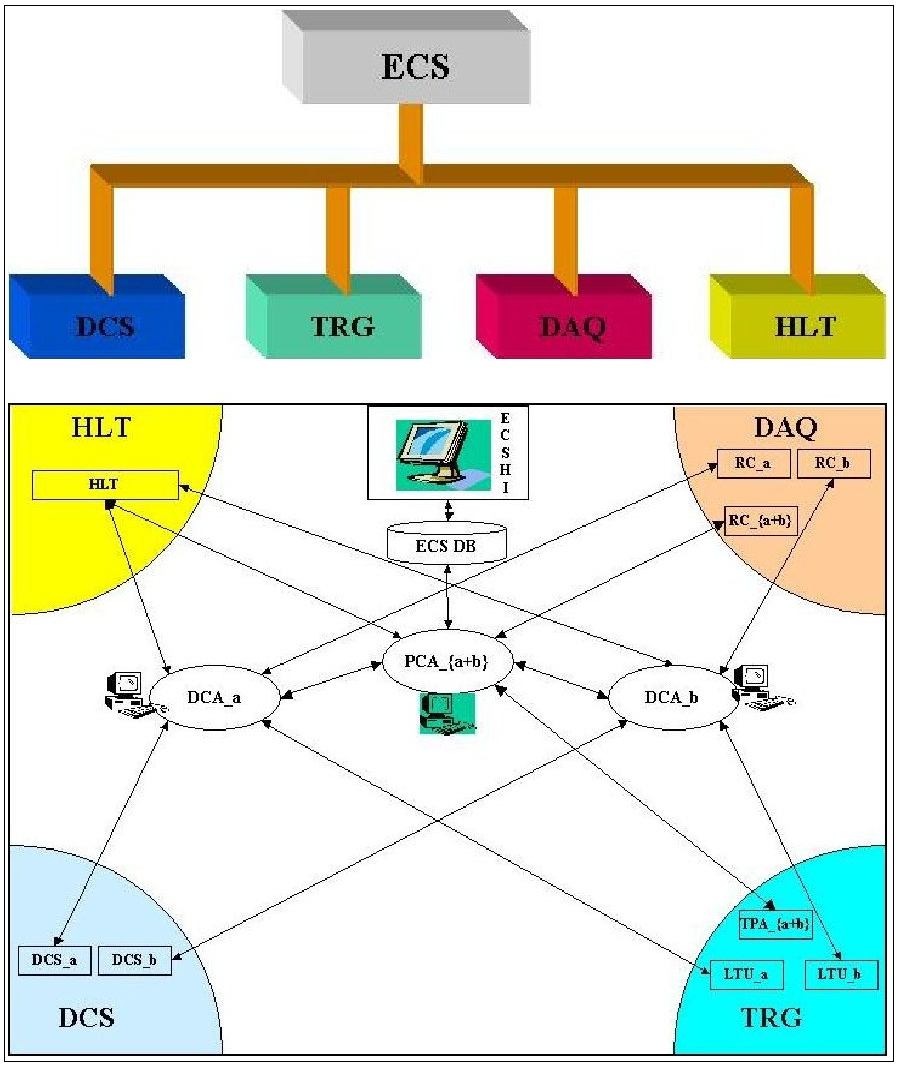


Fig. 1.19 : ECS architecture

2. CARLOSrx

The data produced by the detectors have to be saved in order to be studied, for this reason a card to send these informations to the DAQ system has been developed. CARLOSrx[20] is the SDD readout board and its main tasks are:

- to receive data from 12 SDD detectors at the same time,
- to tag and merge the 12 data streams in input into one sub-event,
- to send the informations towards the ALICE data acquisition system,
- to configure the front end electronics,
- to propagate the clock, reset and trigger signals to the FEE.

CARLOSrx is placed in the counting room **CR4** and it receives data coming from 12 CARLOS boards, installed in the **ALICE cavern.** It also provides the clock and configuration signals to all the end ladder boards and front-end electronics through optical fiber connections. Each CARLOSrx board has to deal with 36 optical fibers, plus several other interfaces like trigger system and VME bus. The first prototype of this board was based on a single board; we were planning to use parallel optical transceivers, since it was not conceivable to have 24 single optical transceivers on the same **PCB** (*Printed Circuit Board*). During the hardware development of the board no commercial parallel optical transceivers for single mode 1310 nm **OT** (*Optical Transceiver*) were available on the market, so we decided to break the design of CARLOSrx in two cards (Fig. 2.1):

- a data processing board, called **CARLOSrx**[15] **data**, that contains:
  - **12 single optical transceivers** from **Optoway**[8] for receiving data from 12 detectors and for sending configuration informations to 12 CARLOS at the same time.
  - **4 FPGA** to handle:
    - data,
    - clock,
    - reset signal,
    - busy signal,
    - trigger signals,
    - VME BUS interface.
- **a clock distribution board**, called **CARLOSrx clock**[15], that receives the clock from the CARLOSrx data board and distributes it through 12 optical fibers to the FEE.

In order to read the data produced by the full SDD barrel, in total 260 SDD detectors, twenty-four CARLOSrx boards are needed: they are hosted in 3x21 slots VME64 crates installed in CR 4.

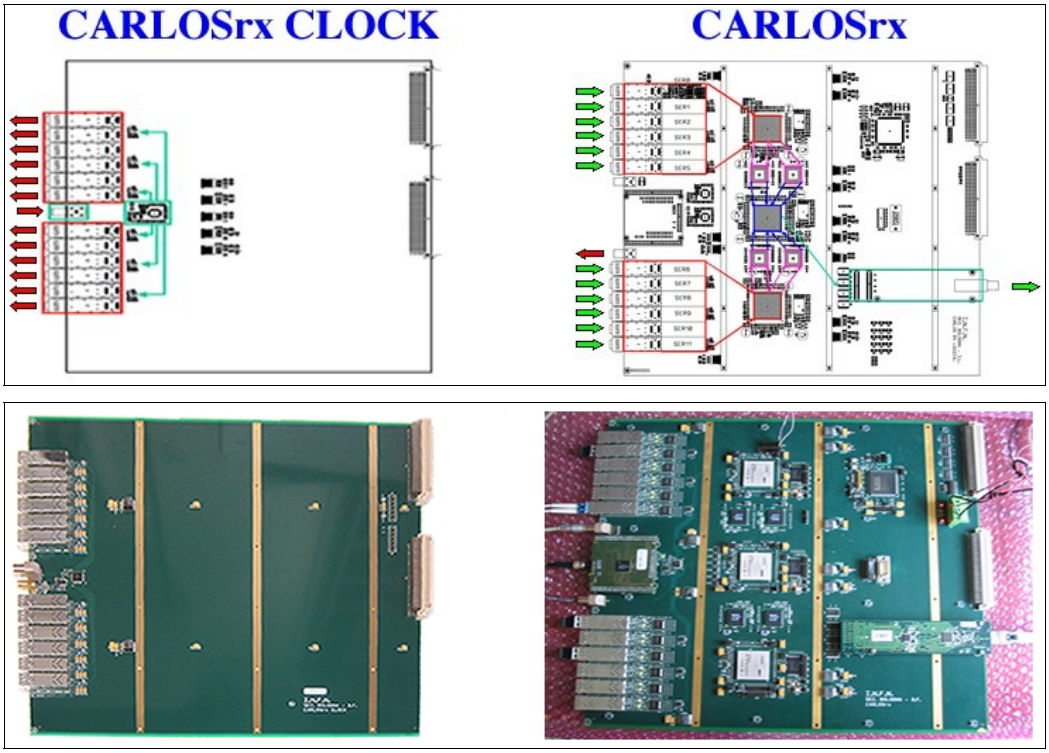


Fig. 2.1: CARLOSrx clock and data layout

2.1 CARLOSrx data processing board

This board is a ***10-layers 9U VME*** printed circuit board with the main task to control and receive data from 12 CARLOS chips, to tag and merge these informations to create a sub-event to be sent towards the data acquisition system (ALICE DAQ). It has several device interfaces:

- **3 XILINX VIRTEX II PRO FPGA (XC2VP20)**[6]: they process data coming from the detectors and handle the communications with the DAQ and the trigger system.
- **1 XILINX SPARTAN II FPGA**[7]: it handles the communication with VME bus.
- **12 optical transceivers**: to receive data from the SDD detectors and to communicate with CARLOS boards. CARLOSrx hosts ***12 x 1.25 GB/s single mode optical transceivers*** from **Optoway**[8]. They are used as ***800 Mb/s*** link for data transmission and as ***40Mb/s*** for serial control.
- **12 de-serializers TLK1501** from **Texas Instruments**[9]**,** to de-serialize the serial

input data into 16-bit words.

- **4 x 9-Mbits IDT FIFO**[10]: they store temporarily the data in order to reduce the busy time and to prevent the informations loss during the data processing.
- **TTC system**[11]: the **TTCrx chip,** installed on the **TTCrq mezzanine card** (***66 x 66 mm***) that is plugged on the CARLOSrx board, provides the **40.08MHz LHC** system clock plus the trigger signals (**L0 – L1 – L2**).
- **BUSY signal**[12]: CARLOSrx communicates with the LTU using a busy signal to stop the generation of triggers. After it has received a L0 trigger CARLOSrx asserts the busy until SDD front end electronics is ready to accept a new one.
- **ALICE DAQ**[5]: the DAQ system is connected with the detectors through the **DIU - DDL - SIU** connection. The **SIU** (*Source Interface Unit*) connects CARLOSrx with the DAQ system through a DDL link (**200 MB/s optical link**). The DAQ infrastructure receives data using a PCI based card, D-RORC[22], installed in several PCs, named LDCs. The D-RORC is connected to the DDL through the **DIU** (*Destination Interface Unit*) embedded in the RORC.
- **VME bus**: CARLOSrx is connected to the VME bus on the backplane of each VME crate, from which it gets the power supply (+5 V and +3.3 V) and the informations to re-configure the firmware of the XC2VP20 FPGAs installed on the board.
- **RS232 port**[13]: this port has been used for test/debug purposes only.

2.1.1 The 12 OPTICAL TRANSCEIVERS

CARLOSrx implements a bi-directional communication with the FEE:

- receiving data from the detectors,
- sending instructions/command to the FEE.

Each readout board has 12 single mode optical transceiver SPS 7110 1.25Gbit/s from Optoway for single mode 1310 nm optical fibers[8].

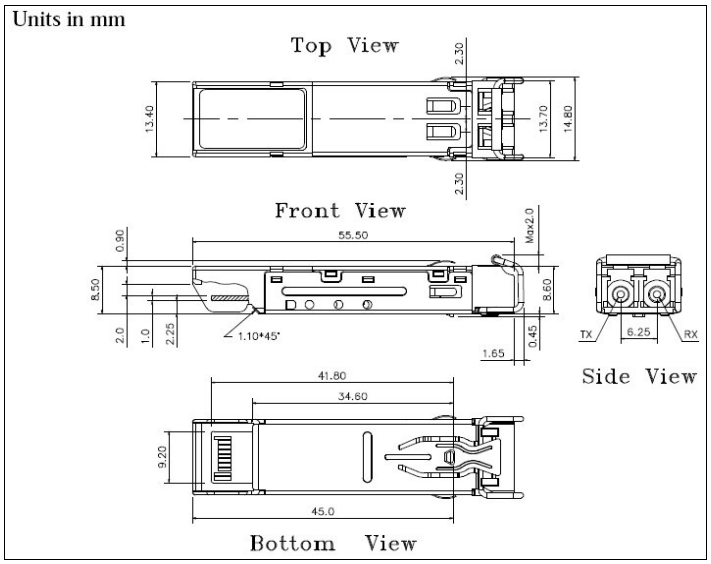


Fig. 2.2: optical transceivers installed on CARLOSrx

Each optical transceiver is **hot pluggable**, providing an easy way to substitute possible broken parts of CARLOSrx without switching off the full crate where the board is installed. Other components of the board can be replaced but in order to do that it is necessary to switch off the CARLOSrx and extract it from the VME crate where it is plugged in. The hot pluggable feature is extremely useful reducing the time needed to replace components of the board.

The data flow can be described as follow (Fig. 2.3):

1. each CARLOSrx optical transceiver receives the data coming from one SDD module;
2. CARLOSrx receives 12 data-streams and de-serializes these informations at the same time;
3. 12 data streams consisting of a 40 MHz x 16-bit buses are received from the two FPGAs connected to the transceivers;
4. after data packing from two 16-bit words to one 32-bit word the data streams are stored into one of 4 IDT FIFOs;
5. one FPGA reads data stored in these FIFOs, builds the sub-event and transmits the informations to the DAQ system through the DDL using the SIU board.

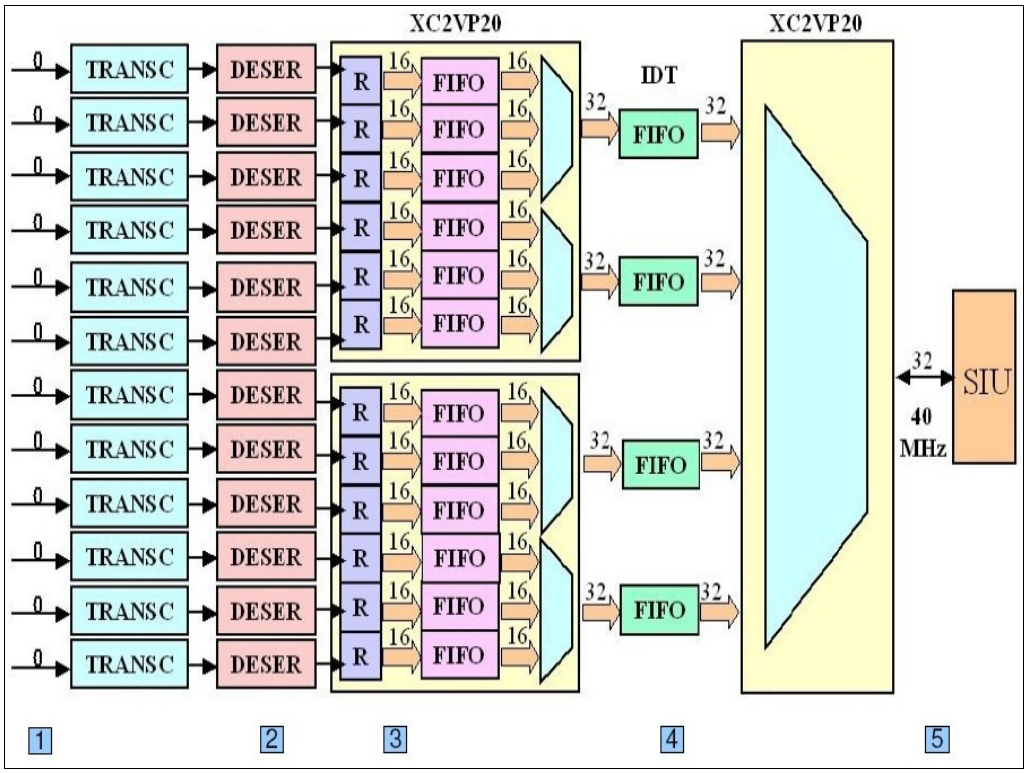


Fig. 2.3: CARLOSrx 5 working steps

The data is de-serialized by a commercial de-serializer **TLK1501** from **Texas Instruments**[9]**.** It supports serial interface speed of **0.6Gbps** to **1.5 Gbps** providing up to **1.2 Gbps** of data bandwidth. The frequency range at which the de-serializer can work is **30MHz** up to **75 MHz**. The serial to parallel conversion is done by a shift register, that is clocked on both the rising and falling edge of the internal generated bit clock which is 10 times faster than the clock in input.

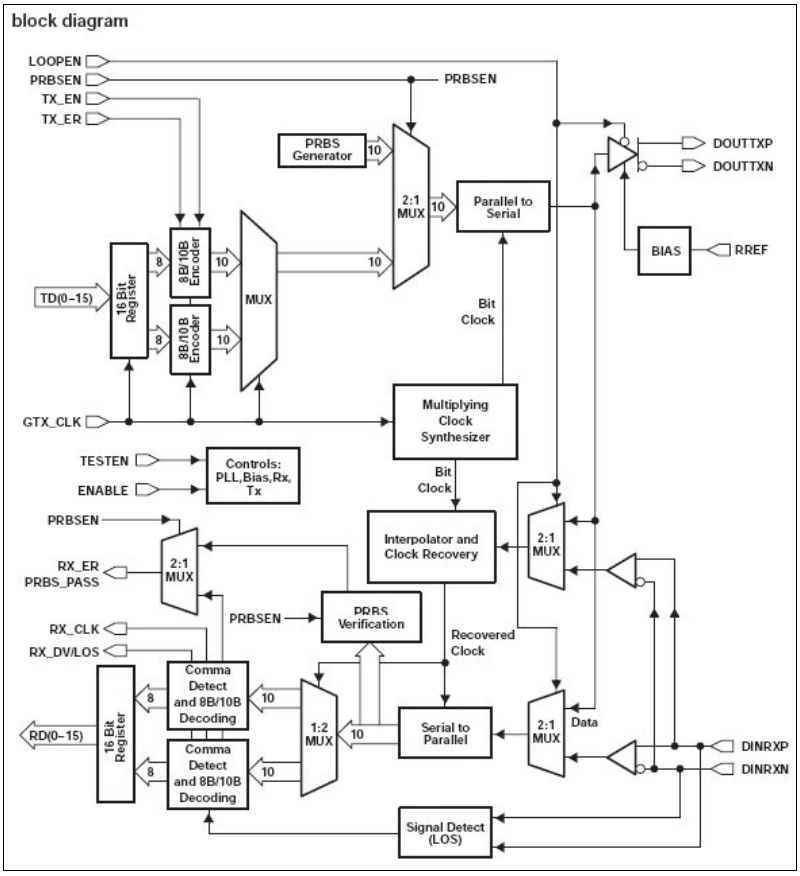


Fig. 2.4: de-serializer block diagram

The optical transceivers are used also to send command/instructions to the FEE like:

- RESET signal,
- trigger signals
- JTAG instruction,

in this case the serial control is **40 Mb/s** (Fig. 2.4).

2.1.2 IDT FIFO 4 x 9 Mbits

Each CARLOSrx receives in input 12 x (16-bit x 40 MHz) data streams and provides in output only one 32-bit x 40 MHz stream that is sent towards the DAQ (Fig. 2.4) In order to keep the acquisition as fast as possible and to prevent the data losses, a system with huge buffers has been implemented on the board. Four large hardware FIFO s temporarily store the incoming data before they are read from the main FPGA and sent to the SIU.

Each buffer is an **extremely high speed CMOS FIFO**[10] memory with clock read and write controls; it is 9 MBits large and it provides five flag pins:

- ***empty flag or output ready*,**
- ***full flag or input ready*,**
- ***half full flag*,**
- ***programmable almost empty flag (set during the initialization phase)*,**
- ***programmable almost full flag (set during the initialization phase)*.**

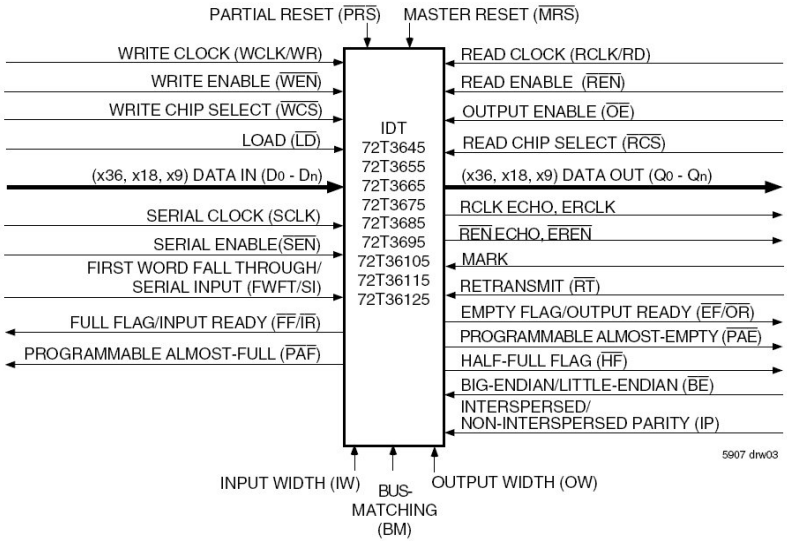


Fig. 2.5: IDT FIFO

The FIFOs receive in input the data coming from 12 CARLOS board and they contain all the informations needed to build the complete sub-event. The information stored in the FIFOs are read by one FPGA that tags and merges all the words and sends them to the DAQ system.

2.1.3 The TTC system

The **TTCrx**[11] is a custom IC designed by the ***CERN Microelectronics Group.*** It acts as an interface between the TTC for LHC experiment and its receiving electronics, in this case CARLOSrx. It has been installed on a mezzanine board **TTCrq** that can be mounted on a standard VME unit without imposing restriction on the spacing between two modules, this is very important for CARLOSrx because all the board are installed in a VME crate.

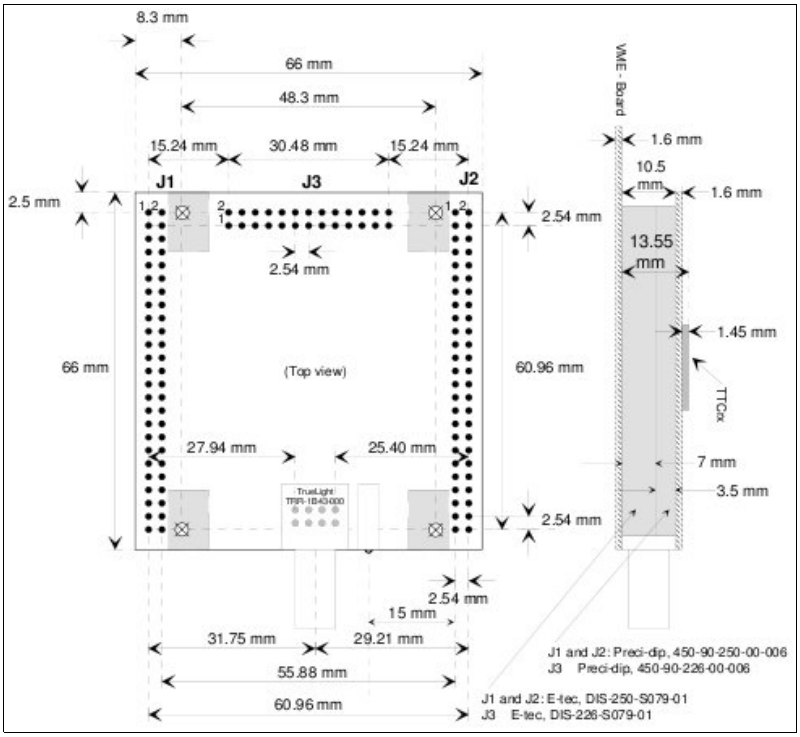


Fig. 2.6: TTCrq board

This ASIC provides the clock and control/synchronization information, such as (Fig. 2.7):

- the **40.08 MHz LHC** clock signal,
- the trigger signals (**L0 L1 L2**),
- its associated bunch and event numbers.

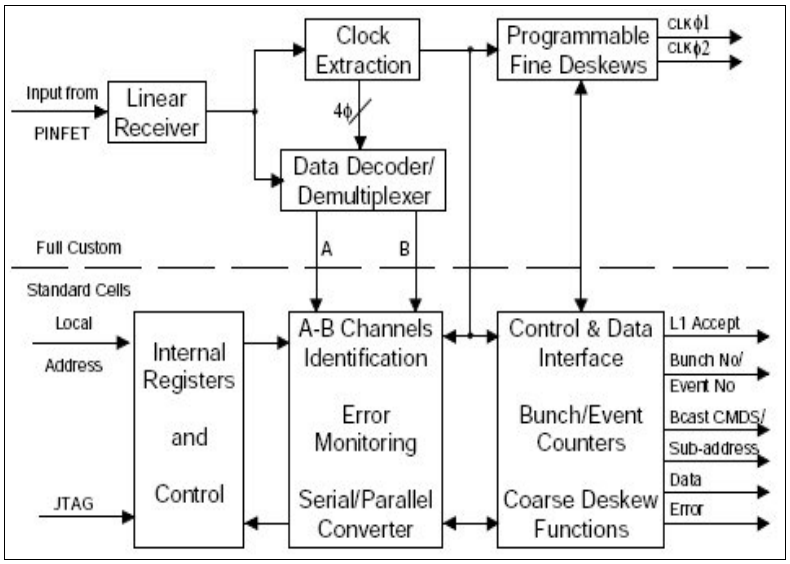


Fig. 2.7 : TTCrx architecture

The clock signal produced by TTCrq board is a **LVDS** (*Low Voltage Differential Signaling*) **signal** and it is converted by CARLOSrx into a **LVPECL** (*Low Voltage Positive Emitter Coupled Logic*) **signal**. The new converted clock enters a fan-out chip that distributes the clock to all the component installed on the board (Fig. 2.8). In the following picture we can see the top and the bottom of the TTCrq board (on the right side of the picture, the **TTCrx** chip).

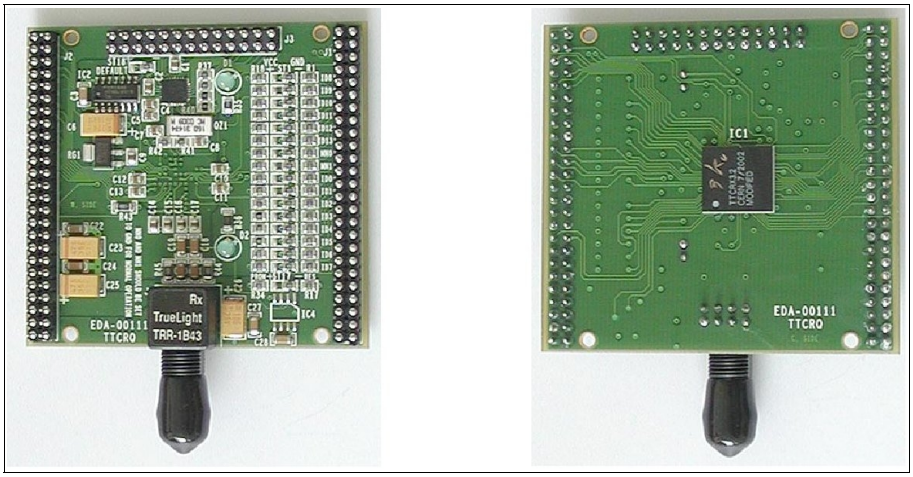


Fig. 2.8: TTCrq board (the right side of the picture shows the TTCrx chip installed on TTCrq board)

All the relevant informations about trigger and clock signals are provided by the LTU through an optical fiber directly connected to the TTCrq optical receiver (Fig. 2.8).

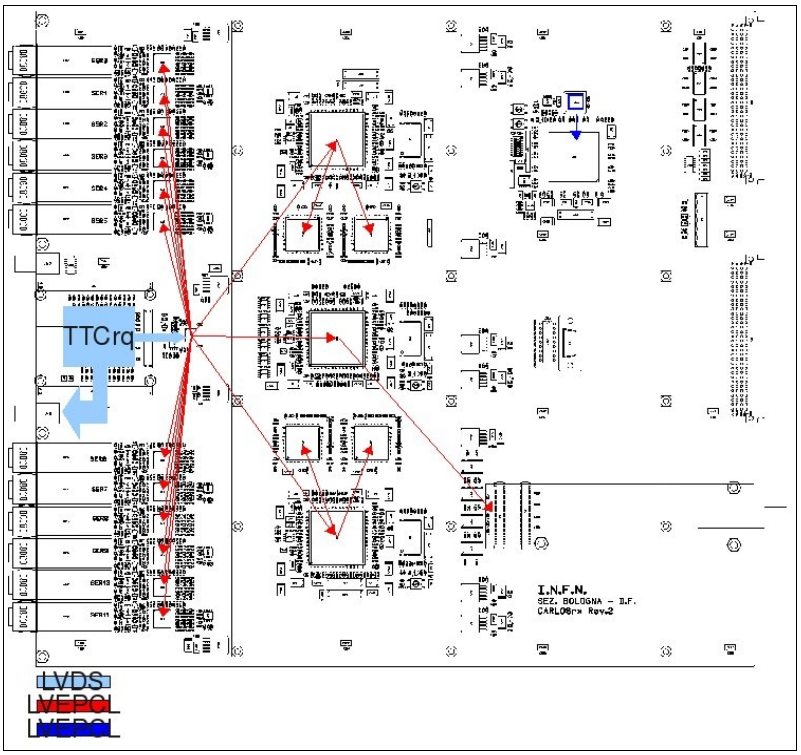


Fig. 2.9: clock distribution in CARLOSrx

The VME client does not need to be synchronized with the LHC clock and there were not any available fanout outputs for sending the TTC clock to the SPARTAN FPGA. So far we installed a quartz that provides a clock of **11 MHz** to this FPGA. It is the only component installed on the CARLOSrx board that does not receive the clock of LHC.

2.1.4 The BUSY signal

During the acquisition there are several situations in which it is necessary to stop the generation of trigger sequences, as will be described later in Chapter 3. The readout board communicates with the LTU through the busy, an LVDS signal. When CARLOSrx asserts this signal the LTU stops sending trigger sequences to the readout electronics. Since the SDD readout chain is composed by 24 CARLOSrx boards, 24 cables have to be connected to the LTU. There is one LTU for the SDD and it has only two inputs for the busy. To connect all the signals provided by all CARLOSrx boards we use a **Faninout**[24] board developed by the trigger team. It is a **6U VME64x** board, electrically connected to the LTU, TTCvi and to the FEE electronics of each sub-detector; for SDD the situation is rather different, since the Faninout, or busy, board has been installed in one of the 3 crates and all the CARLOSrx boards are connected to the busy board by mean of 24 bipolar LEMO cables. In this way there is only one cable connected to the LTU of the SDD.

The Faninout makes fan-in or fan-out of LVDS signals depending on the configuration of the board. It has been designed for :

- distribution of **ALICE L0 trigger signals** as **fan-out:** The L0 input trigger signal is coming from ALICE LTU module and the fan-out L0 signals continues to FEE electronics,
- merging together ALICE BUSY signals as **fan-in** (OR function): BUSY signals are coming from FEE electronics and the single OR function output continues to the ALICE LTU module.

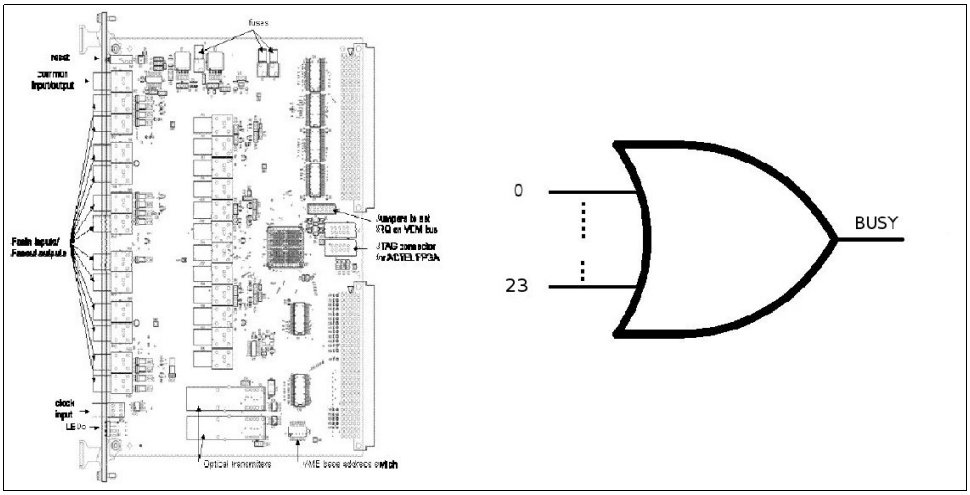


Fig. 2.10 : Faninout board for the busy signal

The Faninout board must be configured every time after a power up because in the default configuration all the inputs are disabled and it is necessary to enable them before starting a run, otherwise the busy will not work properly. In order to do that a software loaded in a CPU VME installed in the same crate of the busy card (Fig. 2.11) configures one register of this board.

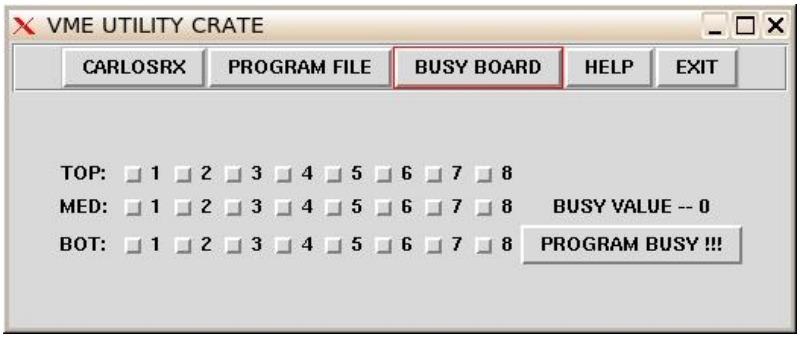


Fig. 2.11: program interface developed to configure the busy board

2.1.5 ALICE DAQ interface

The DDL[21] (Fig. 2.11) is a part of the DAQ of ALICE experiment. The DDL interfaces all the readout electronics of the detectors to the readout receiver card (RORC, a PCI card plugged inside the LDC) of the DAQ. The **SIUs** are connected to the FEE, the DIUs are connected to the RORCs, located in the counting room about 200 meters from the detector. The two DDL interface units are connected through **two multi-mode optical fibers**.

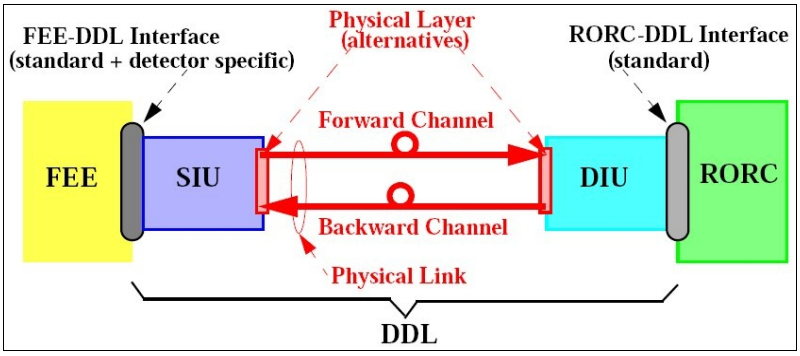


Fig. 2.12: SIU – DDL – DIU link

Each DDL is able to transmit data at a rate of **200 MB/s** with a detected **bit error rate of less than 10^{-15}**.

The FEE and the SIU are remotely controlled by the RORC through the DDL, since their placement inside the detector does not allow using any other cabling apart from the DDL medium. Therefore, commands and status information shall also be transmitted between the FEE and the RORC. So the DDL system provides a bidirectional communication:

- data blocks and commands from the FEE to the RORC:
  - sub event;
  - flow control;
  - status of the SIU board and of the FEE;
- data blocks and status information from the RORC to the FEE:
  - RESET and initialization instructions;
  - commands to the FEE (JTAG instructions - RESET command);

To keep the acquisition as fast as possible the RORC transfer data into the memory of the LDC using **DMA** (*Direct Memory Access*) providing a bandwidth larger than **400 MB/s**. The DMA transfer reduces the need of on-board memory and the software overhead. As soon as the data is provided by the detector readout electronics, the RORC card starts to write data in the PC memory. Every time the RORC is not able to write

data, it asserts a back-pressure signal through the DDL.

Fig. 2.13: RORC card (on the left side the two embedded DIUs, on the bottom side the PCI bus interface)

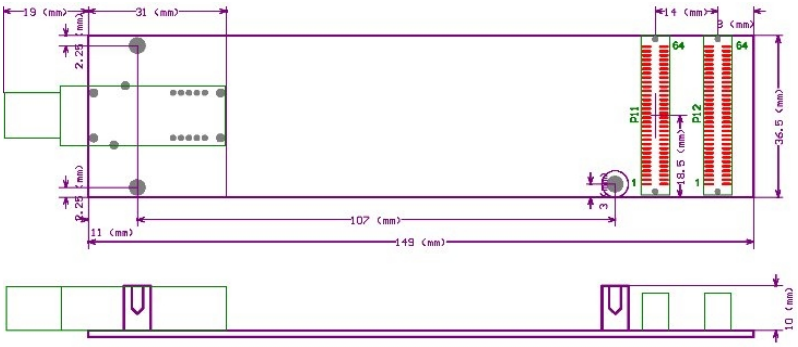


Fig. 2.14: SIU drawing

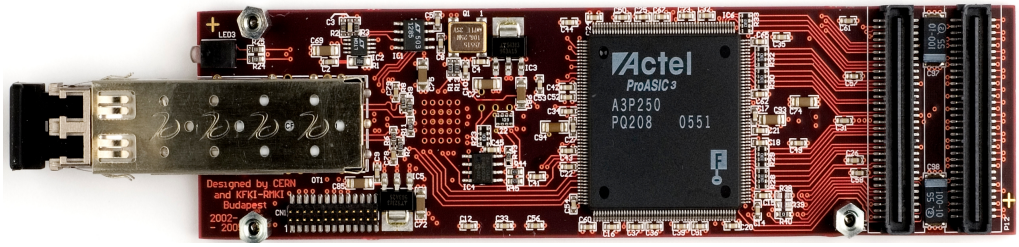


Fig. 2.15: SIU board

A 32-bit bus wise connects the main FPGA with the SIU (Fig. 2.16).

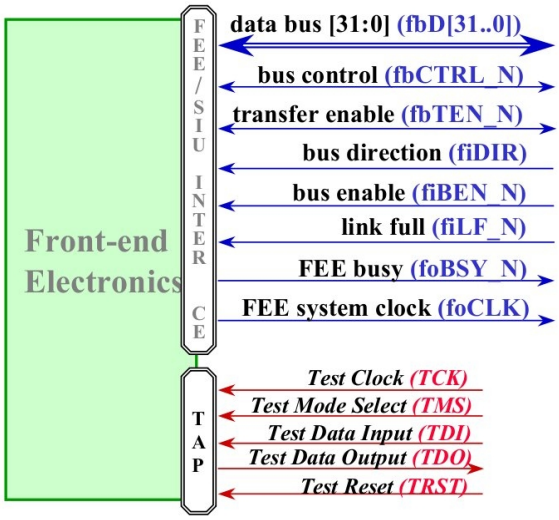


Fig. 2.16: SIU hardware interface

For a detailed description of these signals read **Appendix A.1**.

2.1.6 VME BUS

As mentioned before, all the CARLOSrx boards are installed in a 9U VME crate. From this bus they get power supply and firmware information.

The VME backplane provides:

- **+5 V**,
- **+3.3 V**.

CARLOSrx receives the input voltage and provides stabilized voltage values of:

- **2.5V**,
- **1.8 V**,
- **1.5 V**,

to all the other components of the board.

For what concerns the instructions sent through this bus the SPARTAN FPGA (Fig. 2.17) is connected to the VME bus and acts as VME client receiving the new firmware code.

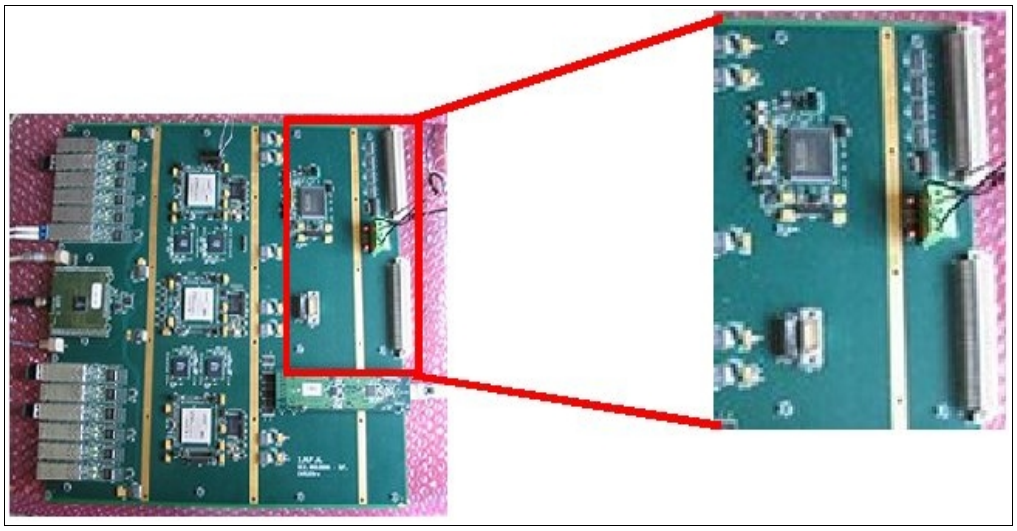


Fig. 2.17: SPARTAN and the VME bus connections

The SPARTAN uses these informations to re-configure the 3 FPGAs (XC2VP20). It is not possible to re-program the SPARTAN firmware using the VME bus so the code of this FPGA has been loaded using a standard JTAG programmer before the board has been installed in the crate and its firmware is fixed.

2.2 CARLOSrx clock distribution board

CARLOSrx clock[15] receives the 40.08MHz clock signal from the data processing board by means of bipolar LEMO cable. Once the clock signal is received it is distributed over the board using a **low jitter LVPECL clock distribution chip**[20] and sent outside towards 12 Optoway optical transceivers to (Fig. 2.18):

- all the CARLOS,
- all the front-end electronics.

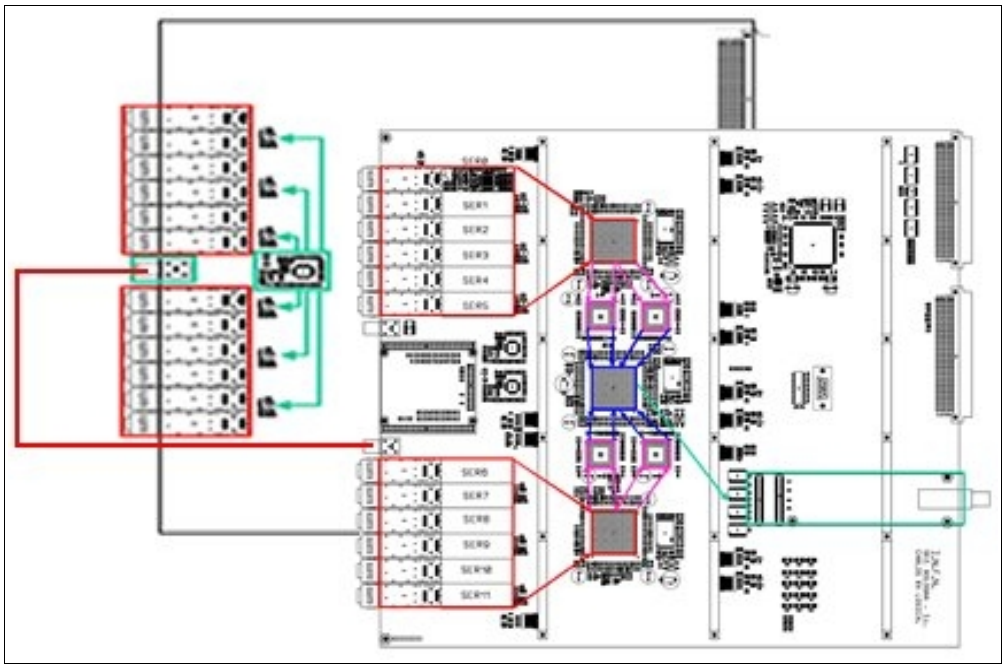


Fig. 2.18: CARLOSrx clock working principle

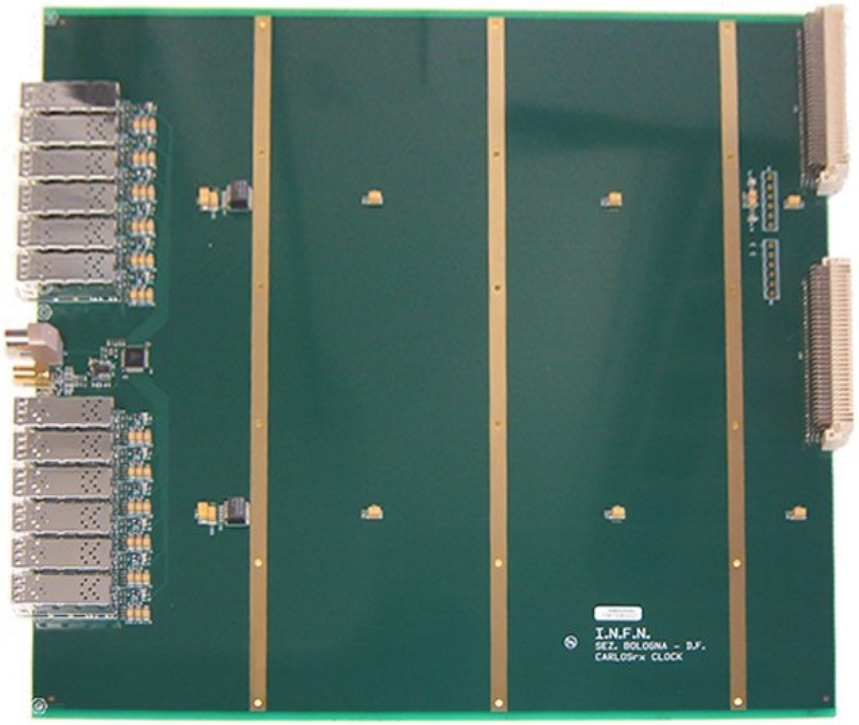


Fig.2.19: CARLOSrx clock board

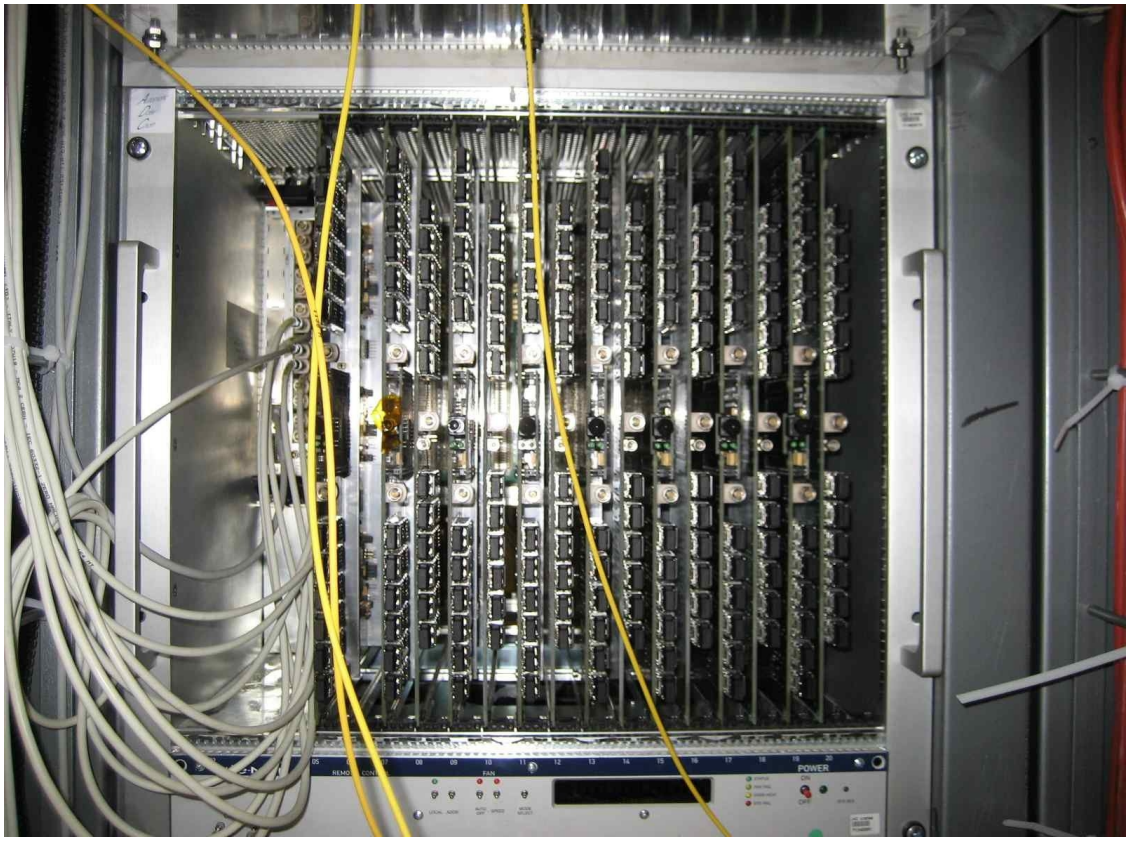


Fig. 2.20: 8 CARLOSrx data and 8 CARLOSrx clock, installed in one of the three crates situated in CR4 at Point 2

3. CARLOSrx firmware

In this chapter the algorithms and the different features of the CARLOSrx data processing board firmware will be presented . From now on the name CARLOSrx will be used to refer to CARLOSrx data processing board, because it does not make sense to speak of firmware for the CARLOSrx clock board, since there is not any FPGA installed on this board and its behavior is fixed.

In order to distinguish among the 3 identical FPGAs, **XILINX XC2VP20**, they will be referred to with the following terms:

- ***input FPGA***, the one connected to the optical transceivers and receiving data from the modules (there are 2 devices of this type),
- ***main FPGA***, the one that takes the data stored in the 4 IDT FIFOs and sends these informations to the DAQ system. This FPGA is the one that builds the sub-event merging 4 data streams coming from the FIFOs in a single one.

The other FPGA, a **XILINX SPARTAN II**, interfaces CARLOSrx with the VME bus to retrieve the firmware information and to re-program the other XILINX XC2VP20 FPGAs.

In the next section the algorithm and the logical blocks implemented will be presented.

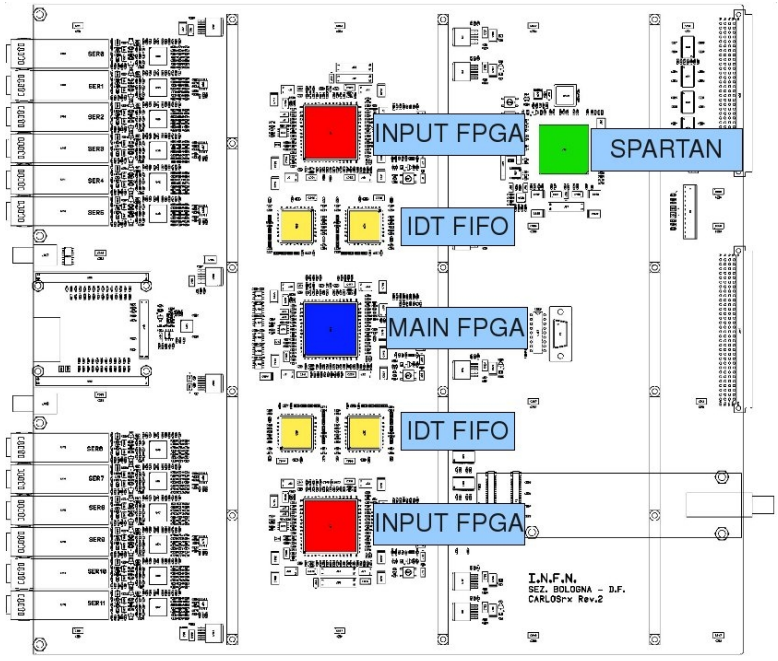


Fig. 3.1: 4 FPGAs installed on CARLOSrx

3.1 Input FPGA firmware

The two input FPGAs, figure 3.2, are connected directly to 6 optical transceivers each one and their main tasks are:

- to concentrate the 16-bit data streams, coming from one CARLOS chip, into one 32-bit data stream,
- to store temporarily these informations into 4 external big FIFOs; each FPGA is connected to two of them,
- to include one or more transceivers in the acquisition, in this way CARLOSrx board can decide to exclude one or more modules from the run.

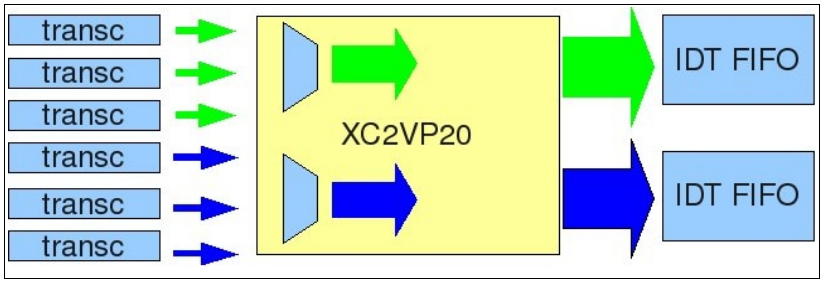


Fig. 3.2: CARLOSrx input FPGA

3.1.1 The algorithm

The task of this FPGA is to receive 6 data streams, 16-bit each one, of data coming from the detectors and merge them into two data streams of 32-bit each one, figure 3.3. After tagging and merging operations data is stored into 2 external FIFOs waiting to be read in order to create the sub-event , this operation is described in the section related to the main FPGA.

Each FPGA operates in this way (Fig. 3.3):

- the data coming from the optical transceiver is de-serialized and stored in a dual clock input FIFO, for synchronizing de-serialized data with the system clock,
- a **DP** (*Data Packing*) **block** encodes the 16-bit input data into words of 32-bit,
- a 4Kwords 32-bit FIFO stores these informations while other channels are being written,
- a **scheduler block** acts as queue manager between 3 inputs and 1 output avoiding FIFO overflow, rising the back-pressure signal to stop the flow of data coming from the SDD modules, in case it is necessary.

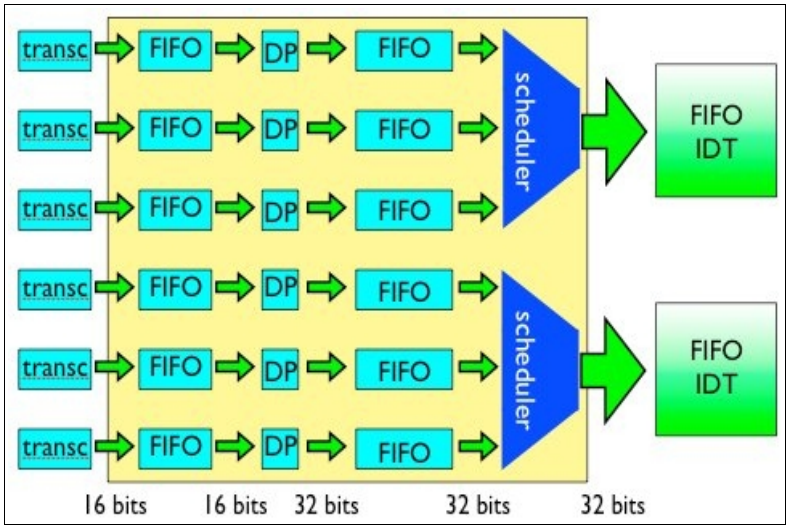


Fig. 3.3: input FPGA algorithm

3.1.2 Data Packing

All the streams coming from SDD detectors must be converted from 16-bit to 32-bit, because the DAQ system expects words of 32-bit wide words. There is a **data-packing block** (Fig. 3.5) for each optical transceiver that fills an internal FIFOs waiting to be read from the scheduler block. It works in this way:

- it receives the 16-bit data stream from the CARLOS board, in which different types of information like data, header, footer and error flag words are encoded (Fig. 3.4),
- it takes 2 x 16-bit words of the same type to build a 32-bit word, for example it puts together two words of channel0 or two words of channel1.

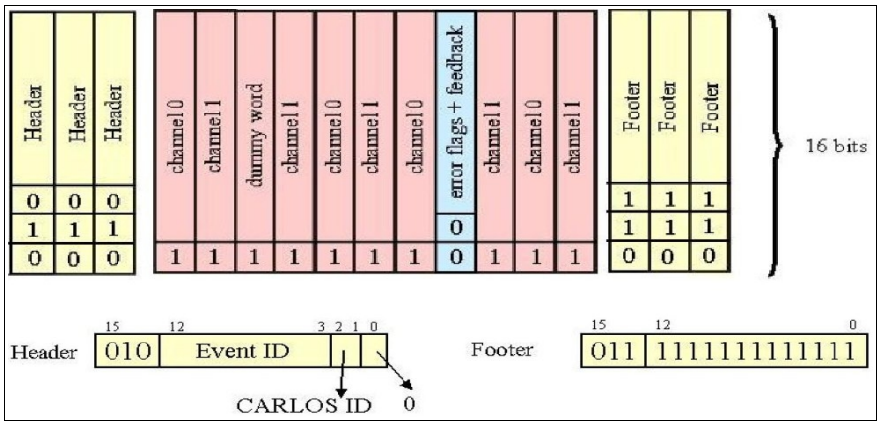


Fig. 3.4: CARLOS output data format

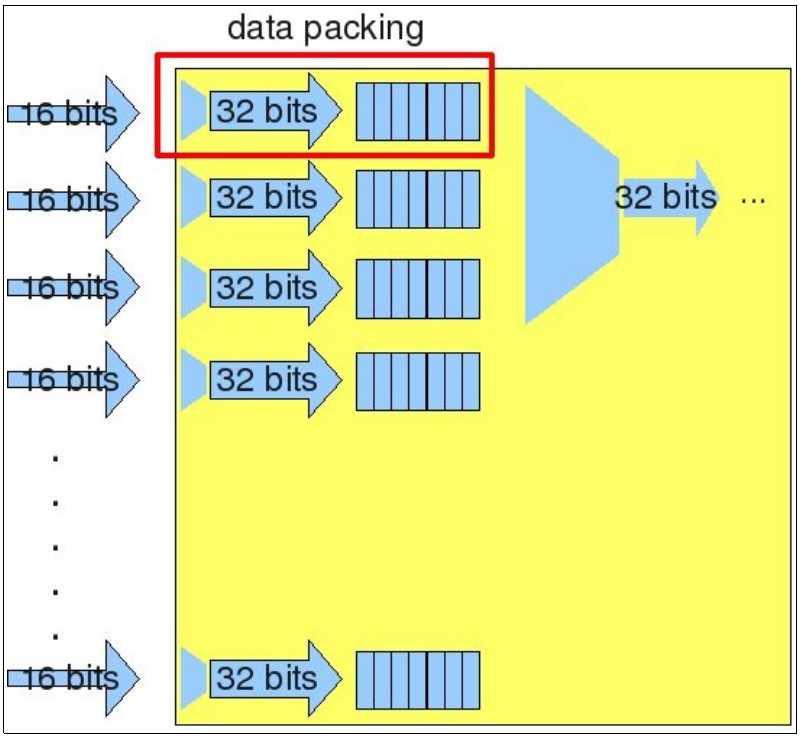


Fig. 3.5: data packing

The 32-bit data stream resulting from this operation has the format shown in Tab. 3.1.

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

Tab. 3.1: data packing format.

When a 32-bit word is ready it is stored into a 4K words 32-bit FIFO, then the scheduler block decides to read the words from these FIFOs.

3.1.2 The scheduler block

This block is particularly important, since it tries to keep the number of words stored in the FIFOs at the same level avoiding that one of them becomes almost full and an other one empty. It also unifies 3 inputs data streams into a single one, tagging and merging all the informations.

It implements a **Round Robin** (***RR***) **queue management** algorithm (it is one of the simplest scheduler algorithms, which assigns time slices to each FIFO in equal portions and in order, handling them without priority):

- if the FIFO IDT is not FULL, the scheduler block writes the 32-bit words read out from the first internal FIFO that is not empty.
- It starts to read out the words from the first FIFO with some data inside (Fig. 3.6).
- After it has read out **256 words**, it jumps to an other FIFO only if it is not empty and starts to read words from this one. If the FIFO is empty it waits for the next not empty one.
- When it jumps from a FIFO to an another before starting to send data out the scheduler block puts a **CARLOS header word** to tag all the data coming from the corresponding module, in this way it is possible to recognize the different words coming from different detectors when they are mixed in the full sub-event.

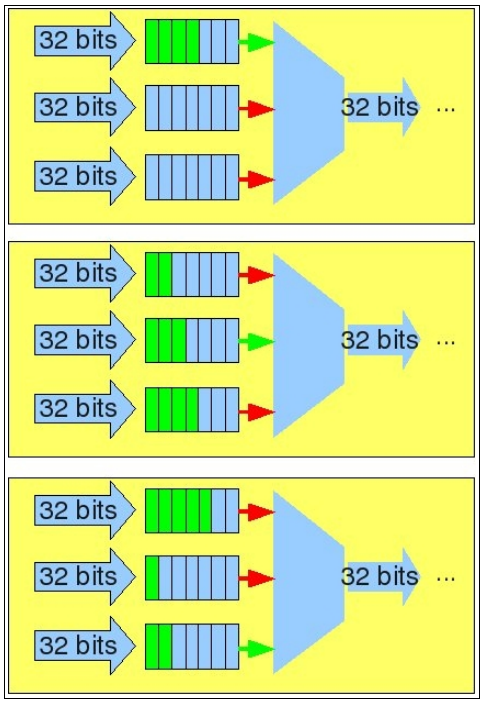


Fig. 3.6: scheduler block in operation, as showed in the figure it tries to keep all the FIFO at the same level (the green arrow indicates the FIFO reads).

When all the words of this event are stored in the external FIFOs the complete sub-event is ready to be built and sent out to the DAQ. The stream in output from the scheduler block contains informations from one to a maximum of three detectors. All the words coming from the same input stream are tagged with a header word, otherwise it would be impossible to reconstruct the informations coming from the same detector when all the words are mixed in the sub-event. The data stored in the IDT FIFO have this format (Fig. 3.7):

- CARLOS0 header word,
- data produced by CARLOS0
- CARLOS1 header word
- data produced by CARLOS1, and so on.

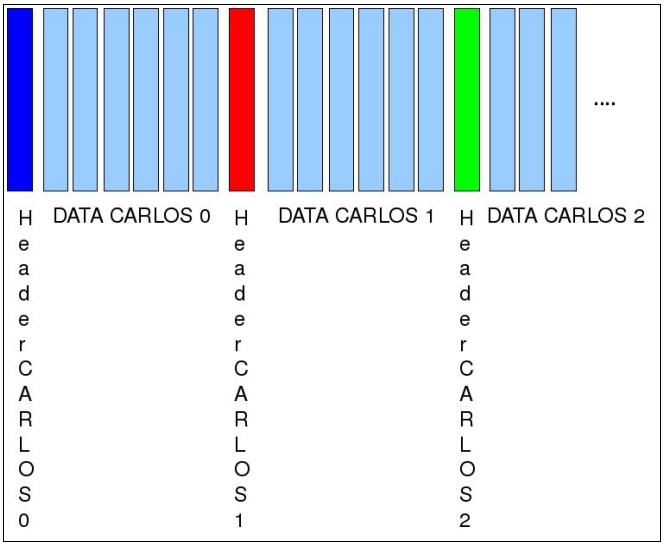


Fig. 3.7: data format

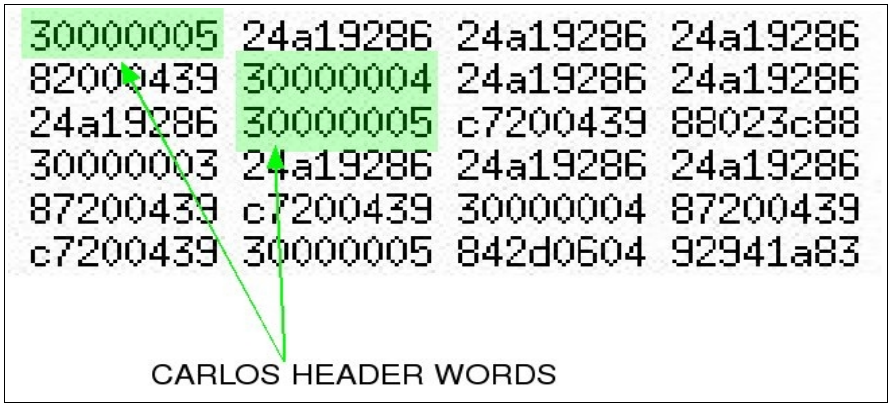


Fig. 3.8: data format stored in the IDT FIFO.

This algorithm provides a unique way to reconstruct all the informations. In order to prevent FIFOs from getting full the scheduler block asserts the **back-pressure signal** that stops CARLOS sending data to the readout board. When the data in the FIFO reach a threshold the back-pressure is asserted.

Back-pressure is very important:

- when transmitting large events,
- when no compression is operated by CARLOS,
- when the trigger rate is high.

3.1.2 INPUTS selection

The input FPGA can decide to exclude one or more modules from the acquisition. This is very useful feature in case some detectors or the front end electronics connected to them show problems that would affect in some way the data acquisition. So by excluding only the faulty module the run can continues without problems

During the JTAG configuration the input FPGA receives the list of inputs to be read during the acquisition:

- a direct bus connects the main FPGA with the two inputs FPGAs, the main one activates the lines of this bus accordingly to the inputs that must be read,
- the input FPGA discards the control signals and the data coming from the modules that must be excluded from the acquisition, in this way the excluded modules will never assert the busy and its data will never appear in the sub-event.

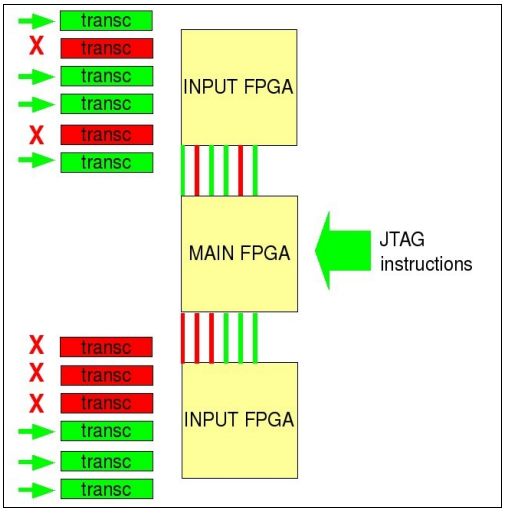


Fig. 3.9: inputs selection

3.1.3 RESET signal

The input FPGAs receive the reset signal just before the JTAG configuration. After receiving the RESET command, the main FPGA forwards this command to the input FPGAs through a direct connection. As soon as this signal is acknowledged they perform the following operations:

- to send the reset to the internal FIFOs,
- to send the RESET to the IDT FIFOs.

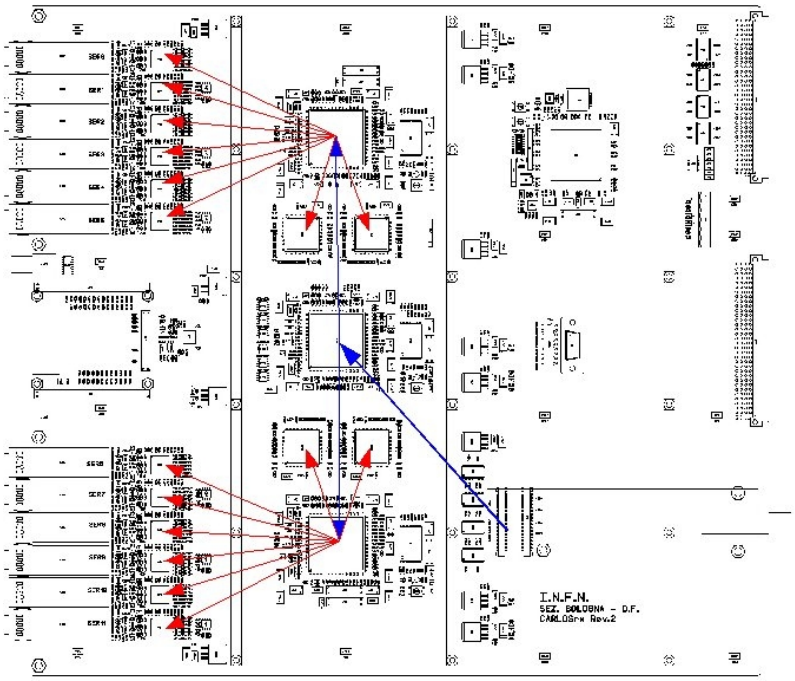


Fig. 3.10: reset distribution, in blue the reset sent to the 2 input FPGAs, in red the reset propagated to the other components from the input FPGAs

After the RESET signal has been sent, the input FPGAs wait for the JTAG instructions and then from data coming from the detectors.

3.2 Main FPGA firmware

This FPGA is the core of CARLOSrx board, whose main tasks are:

- to concentrate all the data stored into the 4 IDT FIFOs,
- to build the sub-event,
- to send it towards the DAQ system through the DDL link.

It has also several interfaces to communicate with the other external components of the experiment:

- **JTAG interface**: the main XC2VP20 is connected to the other 2 FPGAs through a bus and to the the FEE through the serial link. It uses these 2 interfaces to send JTAG instructions and to send the reset signal. This is the only way to configure and initialize the front end electronics of the SDD readout chain.
- **FIFO interface**: it is connected to the 4 IDT FIFOs, that contain all the informations needed to build the sub-event.
- **Optical Transceivers interface**: they receive data coming from the detectors and they also send instructions to the CARLOS boards and the FEE through the serial back link.
- **DAQ interface**: it is connected via a 32-bit bus with the SIU board sending data through the DDL link and receiving from it:
  - the back-pressure signal (when the DAQ system is not able to process data anymore),
  - commands like JTAG instructions or the RESET.
- **TTCrq interface**: this FPGA is directly connected to the TTCrq board, it interfaces the readout board with the trigger system. Through this board CARLOSrx receives the trigger signals plus the 40.08 MHz clock signal.
- **busy signal**: it asserts this signal when the FEE is not able to process data anymore. When the trigger system detects a busy, it stops sending triggers until the busy signal is put to 0 again.
- **RS232 interface**: this port has been used for test purposes. The main FPGA connected to this port is able to interpret commands sent by a PC directly connected to the serial port of the board. Using this system we were able to communicate with CARLOSrx during the acquisition and to change some parameters on the fly or to view the status of the board.

3.2.1 JTAG interface

As described before the DDL link provides a bidirectional communication between the DAQ and the electronics of the SDD readout chain. This features has been developed with the purpose of re-programming the FEE internal registers and the CARLOSrx board. The block of the firmware that handles the JTAG configuration is called **FE2C block** and it works in this way:

- CARLOSrx receives the JTAG words from the SIU,
- it stores them in different buffers (one for each module),
- as soon as it has received all the JTAG words, it starts to encode them using the JTAG protocol,
- it sends to CARLOS chip the instructions serialized through the OTs, to configure the FEE.

Table 3.2 shows the format of the JTAG words sent to CARLOSrx.

	number of words to be read from the RAM			
CARLOSrx	*10xxxxxx*	FF	anode length	load trigger
CARLOS	*11xxxxxx*	*address*	*address*	*address*
	TH ch1	TH ch0	TL ch1	TL ch0
	stop if error	enable 2D	anode length ch1	anode length ch0
	FF	FF	FF	read back
AMBRA ch0 x4	*0000xx10*	*address*	*address*	*address*
	tx address	read counter	write counter	SOP delay
	FF	FF	baseline present	read back
	baseline 3	*baseline 2*	*baseline 1*	*baseline 0*
	...	...	...	...
	baseline 63	*baseline 62*	*baseline 61*	*baseline 60*
PASCAL ch0 x4	*0000xx00*	*address*	*address*	*address*
	calibration DAC		VREF control DAC	
	ADC full	AM full	gain control	sel cal channels
	FF	FF	FF	read back
AMBRA ch1 x4	*0010xx10*	*address*	*address*	*address*
	tx address	read counter	write counter	SOP delay
	FF	FF	baseline present	read back
	baseline 3	*baseline 2*	*baseline 1*	*baseline 0*
	...	...	...	...
	baseline 63	*baseline 62*	*baseline 61*	*baseline 60*
PASCAL ch1 x4	*0010xx00*	*address*	*address*	*address*
	calibration DAC		VREF control DAC	
	ADC full	AM full	gain control	sel cal channels
	FF	FF	FF	read back

Tab. 3.2: JTAG word format.

CARLOSrx receives the JTAG words through the SIU and sends the JTAG informations using the **serial back-link,** towards CARLOS.

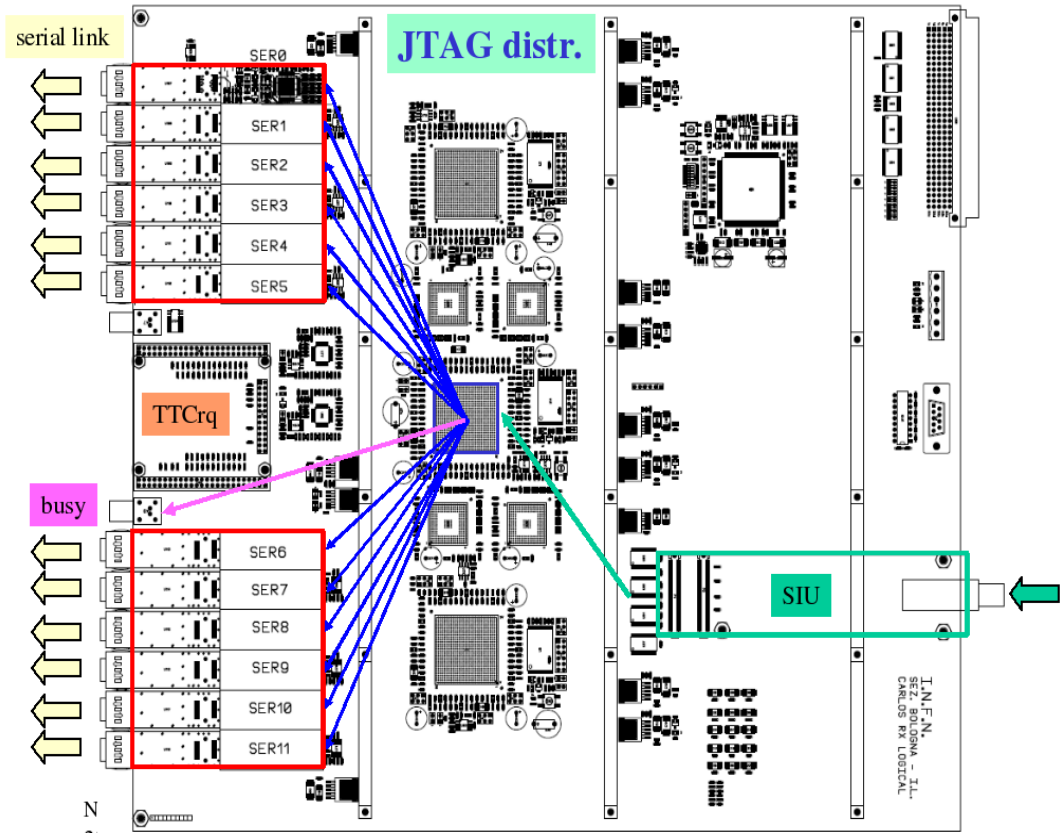


Fig. 3.11: JTAG instructions sent through the SIU to CARLOSrx

Each time we want to configure the front end electronics, the JTAG words are sent to CARLOSrx:

- after decoding the instruction "***Put CARLOS in JTAG mode***", CARLOSrx sends to CARLOS the instruction "***Enter JTAG mode***". CARLOS enters in this mode also after it has received a **RESET signal**,
- now CARLOSrx can program all the module and front-end electronics,
- when all the JTAG words have been sent through the serial back-link the instruction "***Put CARLOS in run mode***" is sent and from now on the chain is ready to acquire data.

3.2.2 Optical transceivers interface (serial back-link block)

The optical transceivers also send data as 40 Mb/s link for serial control, this is the only way that CARLOSrx has to communicate with CARLOS and the FEE.

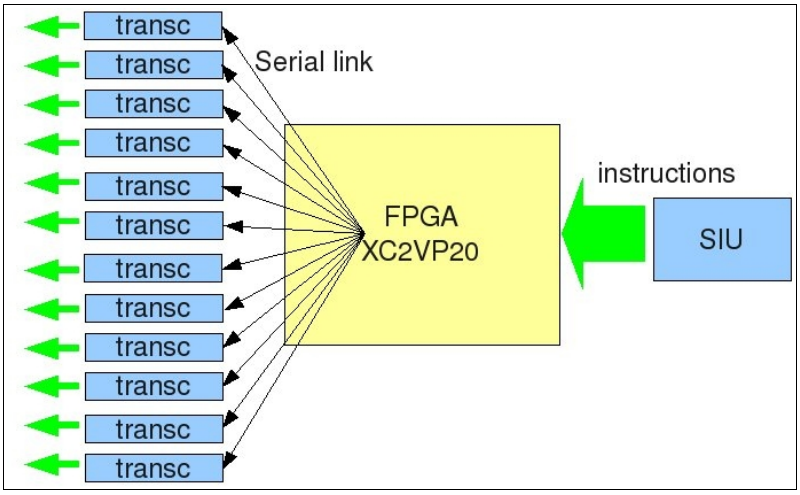


Fig. 3.12: serial link

The **serial back-link block** is the part of the firmware that handles this communication. This is the set of commands that is possible to send to CARLOS:

- ***enter JTAG mode***;
- ***enter RUN mode***;
- ***trigger signal***;
  - *prepulse25 50 75 10 125 150 175 200*;
  - *test-pulse*;
  - *L1 reject*;
  - *L2 reject*;
- ***JTAG information***;
- ***stop acquisition***;
- ***restart acquisition***.

These commands are sent on a 8-bit packets synchronous serial link, so there is a variable jitter of a few clock periods, starting from when a signal is received to the moment the related command is sent to CARLOS. A 3-bit counter continuously runs from 0 to 7, every time the counter value is equal to 7 a command is sent.

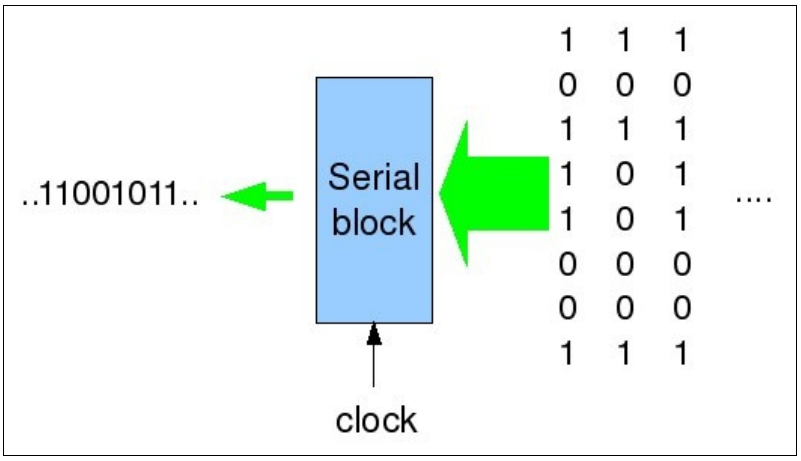


Fig. 3.13: serial link block

This block is extremely important not only for the configuration of the chain, but also for the trigger signals, since each time CARLOSrx receives a trigger signal, it propagates this information back towards the FEE using the serial link (for more details on the serial back-link code used by CARLOS see **Appendix C.2**).

3.2.3 FIFO interface (scheduler block)

It works in the same way like the one installed in the input FPGAs.

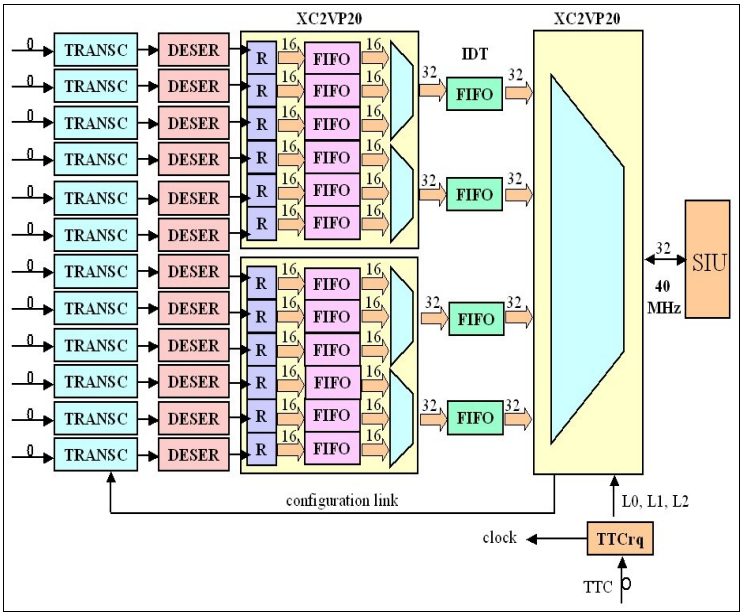


Fig. 3.14: scheduler block in the main FPGA

It reads the informations stored in the 4 IDT FIFOs building the complete sub-events. The working principle is the same used for the scheduler block in the input FPGA, that is a Round Robin queue manager:

- the first FIFO that is not empty is a good candidate to be read,
- when **256 words** have been read or the FIFO is empty, it jumps to another FIFO,
- before starting to read data from one FIFO it writes an header FIFO word,
- this loop goes on until the data belonging to each sub-event are completely read.
- Once all the informations are sent to the SIU CARLOSrx writes 3 MACROFOOTER words indicating the end of the sub-event.

The format of the sub-event is the following:

- FIFO1 HEADER,
- words coming from that FIFO1,
- FIFO2 HEADER,
- words coming from that FIFO2
- ...

The FIFO header words, like the CARLOS header words, are important to identify words coming from the same FIFO when the complete sub-event has been built.

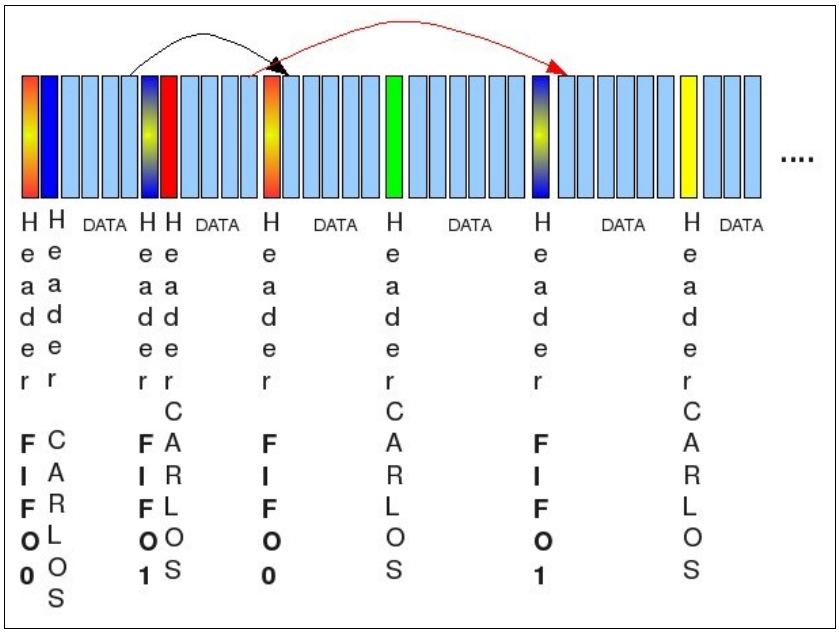


Fig. 3.15: final data format

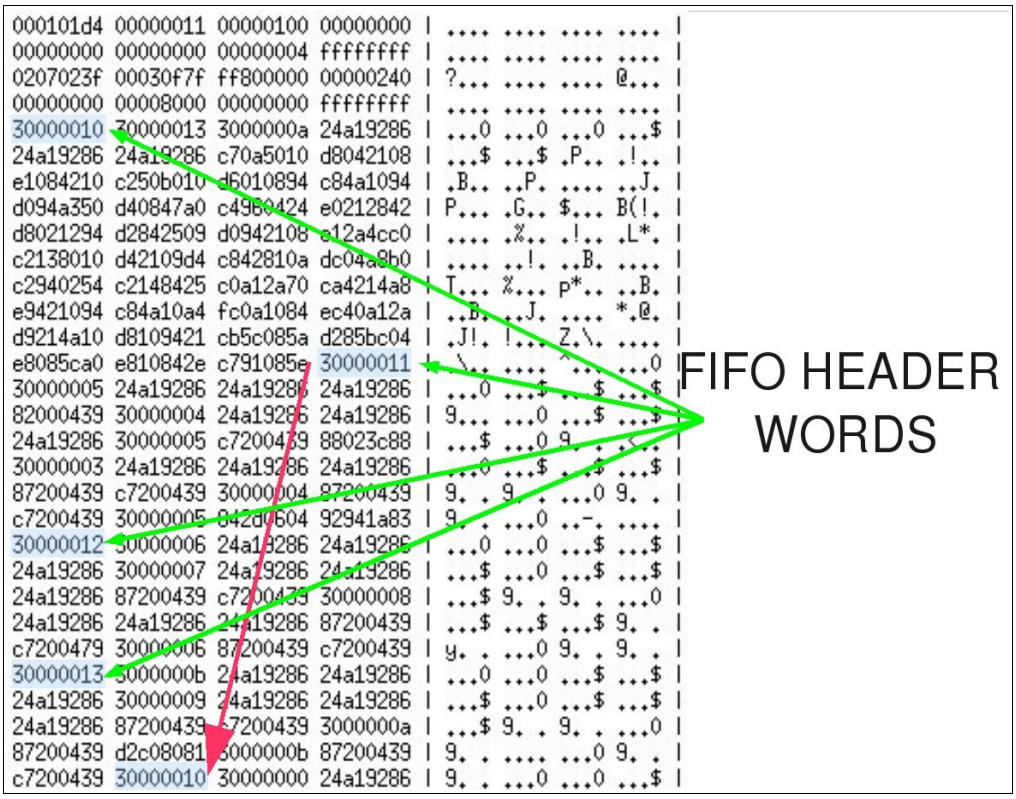


Fig. 3.16: data format in the event.

When the scheduler block has read all the words for a specific sub-event coming from one FIFO it writes 3 MACROFOOTER words. When the decoding program reads these words it knows that it has reached the end of the sub-event and it can starts the initialization procedures to decode a new one. All the words of several events are stored one after another, so **header** and **footer** words are very useful to highlight the beginning and the end of a specific event.

3.2.4 DAQ interface (SIU interface block)

This block communicates directly with the SIU board (Fig. 3.19).

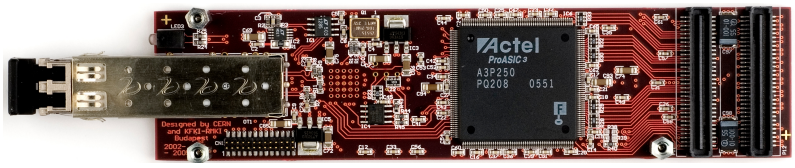


Fig. 3.17: SIU board installed on CARLOSrx

This board provides a bidirectional communication bus with the DAQ framework:

- **INPUT mode**: the SIU acts as master, while CARLOSrx as a slave. In this configuration the SIU board sends commands to CARLOSrx that will decode them and sends the corresponding instructions to the FEE.
- **OUTPUT mode**: CARLOSrx acts as master and the SIU as a slave, in this case CARLOSrx takes the control of the bus and sends the data coming from the detector over the DDL using the SIU board as interface.

When the SIU is in MASTER mode, it sends instructions or RESET command to CARLOSrx. In this configuration if CARLOSrx detects the reset command, this signal will be sent to all the rest of the electronics. This provides an easy way to reset all the electronics in an automated way, for instance this reset command can be sent at the beginning of each run from the DAQ system. When in INPUT mode CARLOSrx receives the JTAG instructions to reprogram the front end electronics. In this case a software installed on the LDC sends through the SIU all the JTAG words. When the JTAG words are received from CARLOSrx they are stored in internal RAMs (one for each CARLOS). Once it has received all the words it starts to program the FEE sending them all the JTAG instructions towards the serial link connected to each CARLOS.

After sending each event CARLOSrx waits for a decision from the SIU in order to decide if to release the control over the bidirectional bus or to send a new event when available, in this case:

- CARLOSrx waits for the SIU to be ready. When it is possible to send data CARLOSrx recognizes the command RDYRX (Ready To Receive).
- Each data packet begins with the DDL header and ends with a front end status

word, which confirms that all the sub-event has been sent.

- After this words CARLOSrx waits for a new decision from the SIU, that it can take back the control or leave the situation like it is.

3.2.5 TTCrq interface

The main FPGA is directly connected with the mezzanine board TTCrq and this block has the important job to decode all the information provided by the trigger system. This board is connected with the SDD LTU through an optical fiber.

The main tasks of this block are the following:

- to fill the **CDH** (*Common Data Header*),
- to decode the **L0 L1 L2 trigger signals**,
- to provide the related informations in case of erroneous trigger sequences.

The CDH[18] is really important for the DAQ system to be able to identify all the data blocks transferred by every detector over the DDL. All the events associated to the same trigger are sent over the DDL within the same block and preceded by one header with the same trigger identification informations.

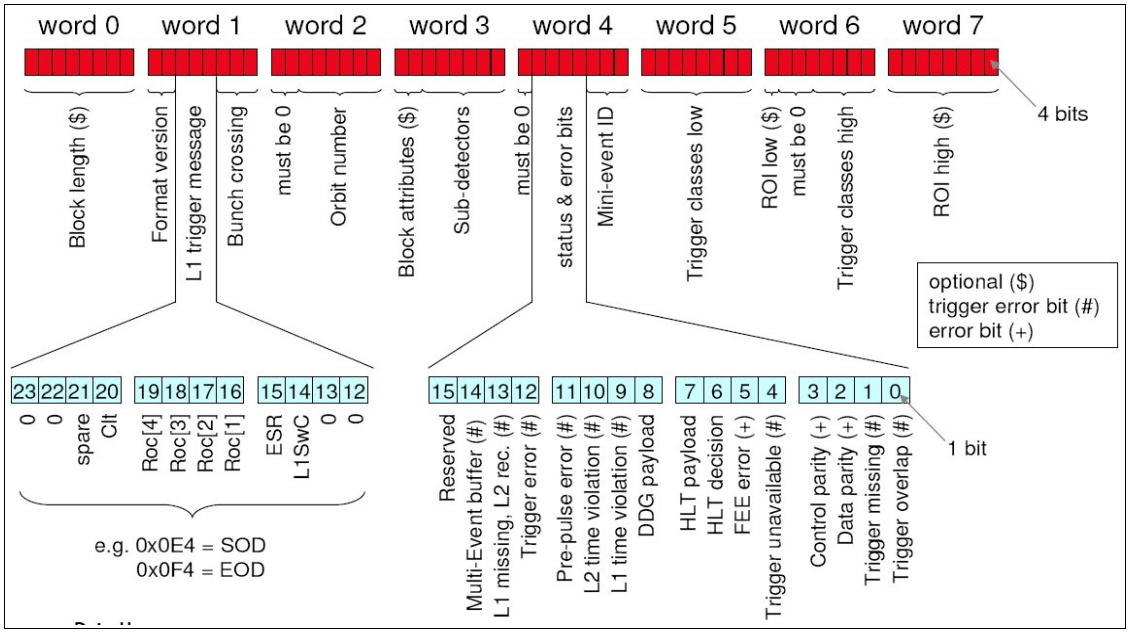
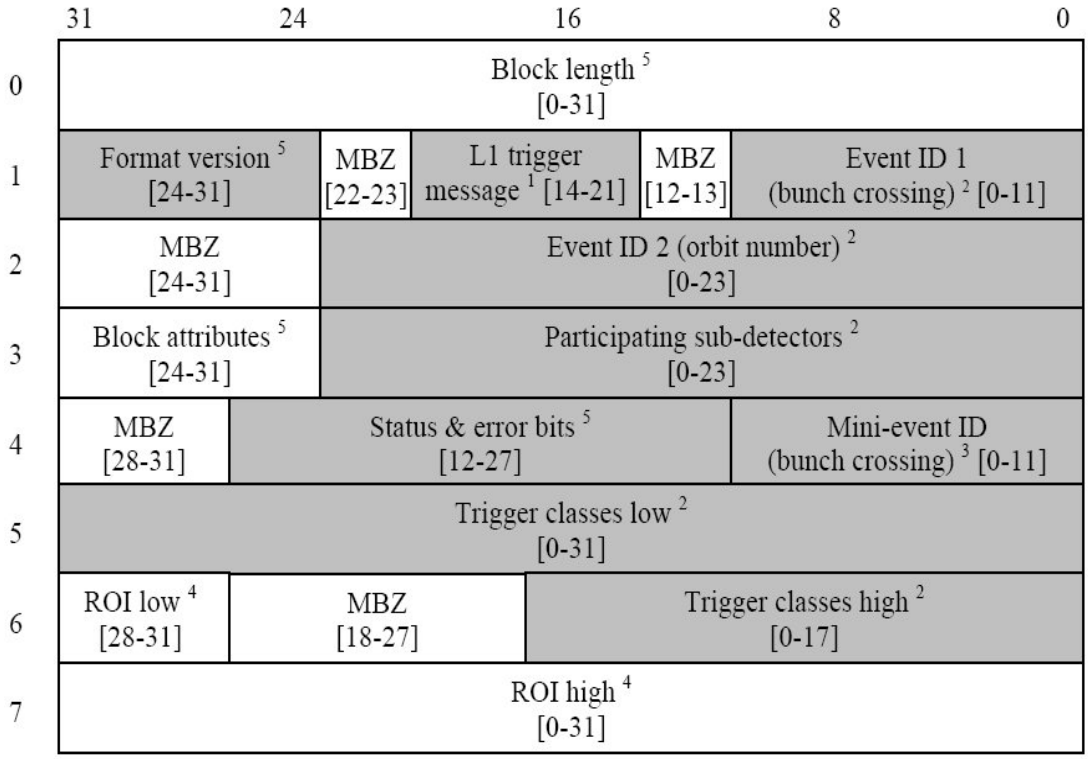


Fig. 3.18: Common DATA header explanation



Tab. 3.3: Common Data Header structure

All the fields of the CDH must be filled by the readout electronics of the detector using informations either fixed in the firmware (like CDH version) or provided by the Trigger system during the run (if the ALICE Trigger system is not available, like in the standalone runs done in our lab setup, these informations were handled directly by the electronics, for more details on the CDH fields refer to **Appendix A.2**).

The CDH has not only been developed to recognize different data blocks, but it is also used to perform preliminary checks of the data. In fact during the run the readout software of the DAQ controls if specific rules are respected, and in case of errors, it stops the run reporting the cause of the problem. CARLOSrx fills the fields of the CDH once it has decoded the trigger informations received by the TTCrq board and, in case of erroneous trigger sequences, it provides a CDH with the related trigger error bits asserted.

CARLOSrx is able to identify the following list of erroneous trigger sequences:

- Spurious L0 Error.
- Spurious L1 Error.
- L1 Message Data Error.
- Incomplete L1 Message Error.
- Spurious L1 Message Error.
- Missing L1 Message Error.
- L2a Message Data Error.
- Incomplete L2a Message Error.
- Spurious L2aMessage/L2r Word Error.
- Missing L2a Message /L2r Word Error.
- BC Identifier Error.
- Prepulse Error.
- Calibration Trigger Error.
- L1 Message Content Error.
- L2a Message Content Error.

For a detailed description of these erroneous trigger sequences refer to the **Appendix B.1**.

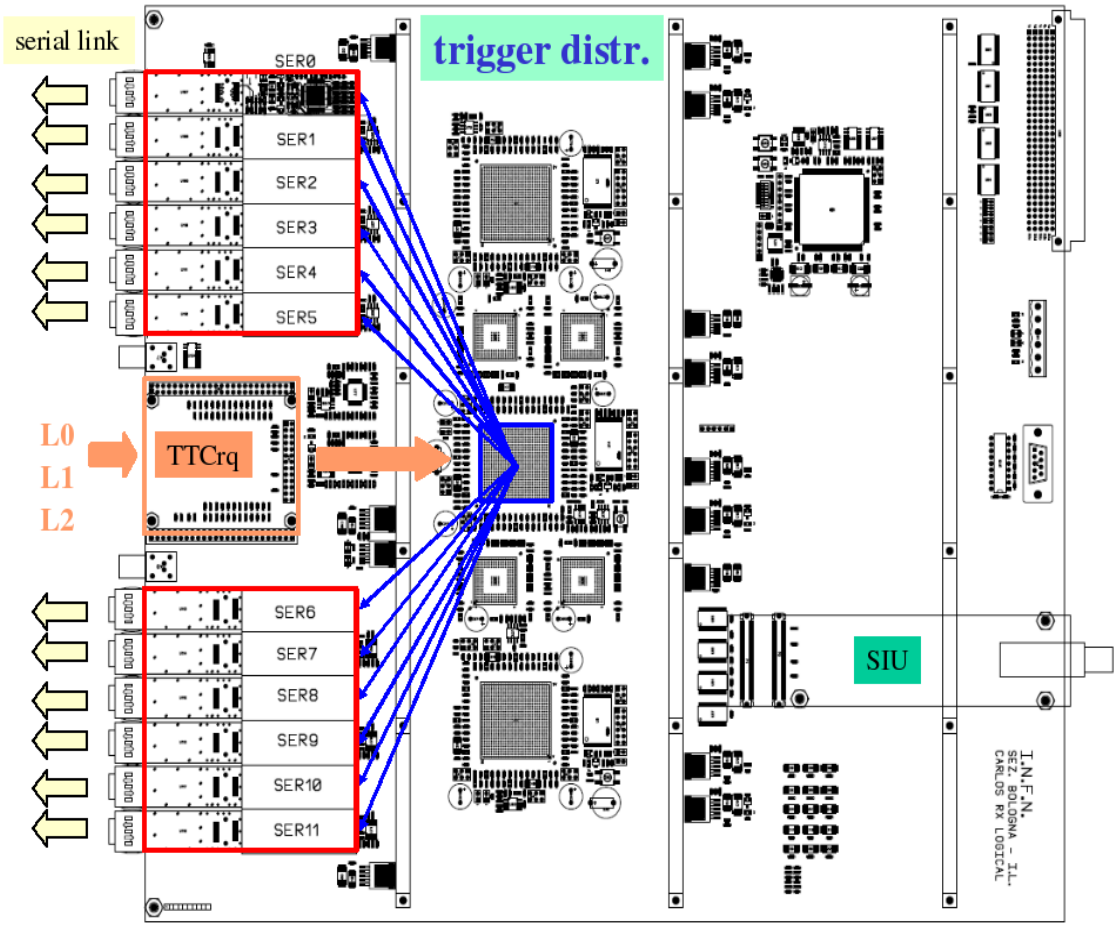


Fig. 3.19: CARLOSrx trigger handling

Info	aldaqpc084	Readout has received START_OF_DATA after 0 seconds, eventId: Period counter = 0, Orbit counter= 2557396, Bunch crossing = 1750
Info	aldaqpc003	Waiting 10 seconds for START_OF_DATA
Info	aldaqpc097	AsynchReadRorcData RorcData: eqId=1334 has seen SOD
ERROR	aldaqpc097	ReadEvent RorcData: eqId=1335, (ERROR 357) CDH trigger error bit(s) present in CDH, status/error bits=0x1400= + CDH_TRIGGER_L2_TIME_VIOLATION_ERROR_
ERROR	aldaqpc097	--- dump CDH --- # 1 --- eqId 1334 --- 0x0000013c 0x020e46d6 0x002705d4 0x3601027f 0x00000736 0x7f000000 0x00000102 0x00000000
ERROR	aldaqpc097	--- dump CDH --- # 2 --- eqId 1335 --- 0x0000012c 0x02000000 0x00000000 0x37000000 0x01400f1c 0x00000000 0x00000000 0x00000000
ERROR	aldaqpc097	Error 357 from above is TOLERATED: this equipment has seen so far 1 *trigger* errors, maxErrors=10
ERROR	aldaqpc097	ReadEvent RorcData: eqId=1335, (ERROR 359) CDH mismatch with the L1 trigger message, current=0x0, former=0x39
ERROR	aldaqpc097	--- dump CDH --- # 1 --- eqId 1334 --- 0x0000013c 0x020e46d6 0x002705d4 0x3601027f 0x00000736 0x7f000000 0x00000102 0x00000000
ERROR	aldaqpc097	--- dump CDH --- # 2 --- eqId 1335 --- 0x0000012c 0x02000000 0x00000000 0x37000000 0x01400f1c 0x00000000 0x00000000 0x00000000
ERROR	aldaqpc097	Error 359 from above is TOLERATED: this equipment has seen so far 1 errors, maxErrors=10
ERROR	aldaqpc097	ReadEvent RorcData: eqId=1335, (ERROR 364) CDH event identification cannot be zero
ERROR	aldaqpc097	--- dump CDH --- # 1 --- eqId 1334 --- 0x0000013c 0x020e46d6 0x002705d4 0x3601027f 0x00000736 0x7f000000 0x00000102 0x00000000
ERROR	aldaqpc097	--- dump CDH --- # 2 --- eqId 1335 --- 0x0000012c 0x02000000 0x00000000 0x37000000 0x01400f1c 0x00000000 0x00000000 0x00000000
FATAL	aldaqpc097	Readout asks to stop the run: Error 364 in routine ReadEvent active equipment 3
Info	aldaqpc007	Waiting 10 seconds for START_OF_DATA

Fig. 3.20: sample of CDH errors recognized by the DAQ system, generated by readout electronics

3.2.6 The BUSY block

The main FPGA has the important task to assert the busy signal when the FEE is no longer able to receive triggers. When a L0 trigger is received, the SDD busy signal is immediately set and, after a programmable delay which accounts for the L0 latency (1.2 μs) and the maximum detector drift time (~5 μs), the analogue memories are frozen. The busy signal being still set, their contents are then digitized by a set of 10-bit linear successive-approximation ADCs which write the data into one of the free AMBRA buffers. The digitization takes about 230 μs (120 μs when the half frequency mode is programmed) and can be aborted by the absence of the L1 trigger or by the arrival of an L2-reject signal; in both cases, the front-end electronics reset the SDD busy signal and returns to the idle state within 100 ns. On the successful completion of the analogue-to-digital conversion the SDD BUSY is reset if at least one buffer is still available in the AMBRAs. The busy asserted by the FEE is always present after each trigger, but there are situations that can increase the busy time of the detector: for example when the trigger rate is too high or the event size is too big. In these cases the DAQ or CARLOSrx can take too long to handle all the informations by asserting the back-pressure signal. If this situation goes on for a long time, AMBRA buffers will all get full and the busy will be asserted.

3.2.7 The RESET block

CARLOSrx receives 3 RESET signals, making it possible to initialize different components of the board one at a time. These signals are sent to the board using the DDL link. Once the SIU has received the RESET command this informations are put on the 32-bit bus and they are decoded by a logic whose task is to identify one of the RESET commands:

- **XXXXAAAA**: after receiving this command this block generates the internal reset signal of CARLOSrx, that is also sent to all the logic on CARLOSrx. The reset is also sent to the CARLOS boards using the serial link.
- **XXXXBBBB**: after receiving this command, the reset block initializes all the internal **DLLs** (*Digital Locked Loop*)
- **XXXXDDDD**: this command is used to reset the TTCrx chips, which need to be initialized in order to provide correct informations for each event to be stored in the CDH.

3.2.8 The UART block

This block, whose task is to interface CARLOSrx to the RS232 port of a PC, is no longer part of the final release of the firmware because it has been used only during the test phase of the board.

It reads the data coming from the serial port and interprets the instructions executing different commands like:

- ***to generate RESET signal***,
- ***to start the internal data generator***,
- ***to stop the internal data generator***,
- ***to change the trigger rate***,
- ***to change the event size***,
- ***to print the FIFOs and register status***.

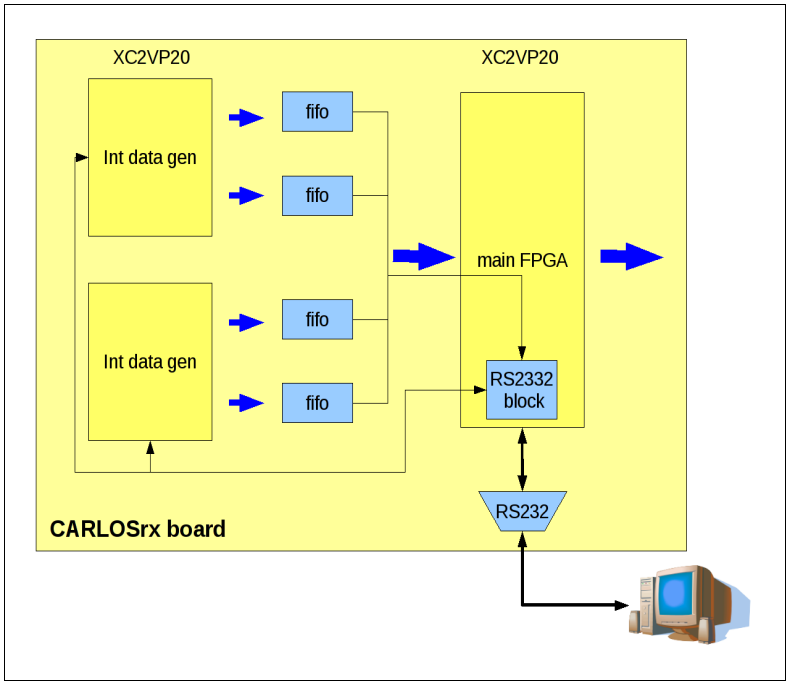


Fig. 3.21: RS232 block

The only possible way to communicate with CARLOSrx is using the DDL link, so the RS232 block has been removed as soon as the CARLOSrx board have been installed in CR4; besides it is better to avoid using a cable longer than 5 meters to prevent errors in the signal transmission.

3.2.9 The VME interface

The VME bus is interfaced by the **SPARTAN FPGA** that is connected to this bus through the two VME connectors installed on CARLOSrx. We have implemented a SERVER-CLIENT strategy to communicate with all the boards installed in the crate. The VME CPU (server) sends the firmware information through the VME bus, that is read by the SPARTAN FPGA acting as VME client and used for configuring the 3 XILINX XC2VP20 FPGAs. In each crate the following boards have been installed:

- 1 CPU VME,
- 8 CARLOSrx boards.

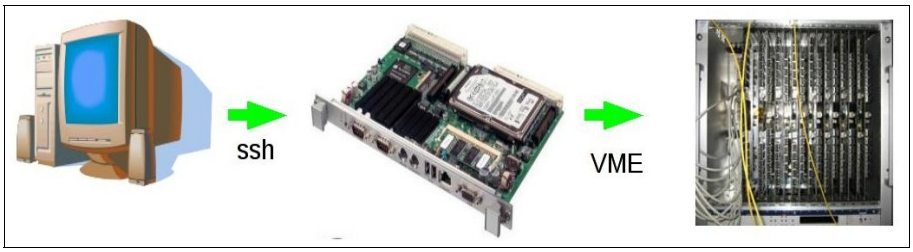


Fig. 3.22: JTAG configuration with the CPU VME

The VME bus adopts a SERVER CLIENT strategy:

- **SERVER**: the CPU VME sends all the informations in broadcast mode,
- **CLIENTS**: all the SPARTAN FPGAs see the firmware code that is passing in the bus, but only the one addressed reads these instructions.

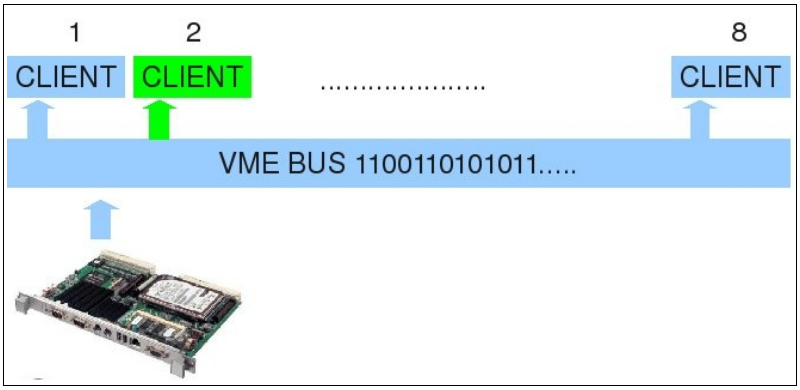


Fig. 3.23: broadcast communication

The operating system installed in the VME CPU is a version of GNU/LINUX, SLC 4 and it provides standard communication protocol like ssh; in this way it is possible to control the VME bus and to send the new firmware code remotely.

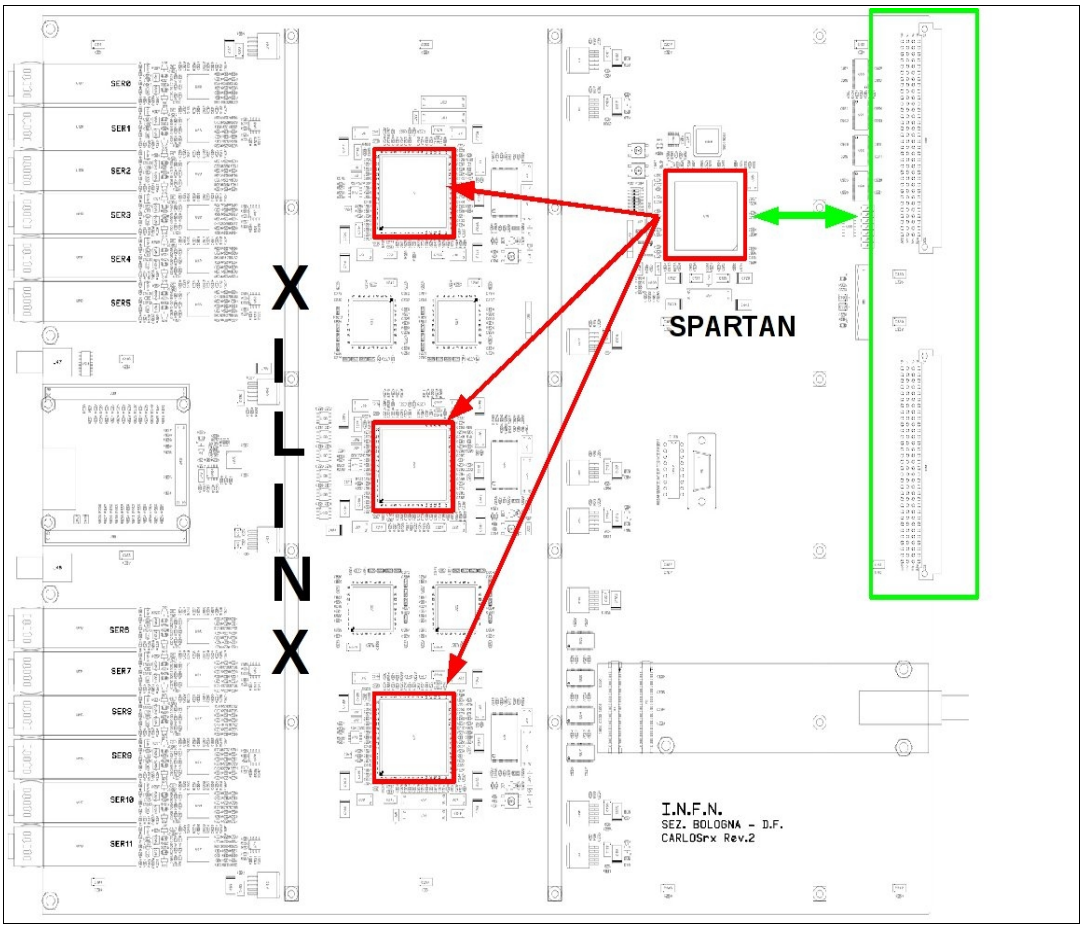


Fig. 3.24:SPARTAN interfacing the VME bus

4. The software developed for CARLOSrx

The development of software tools is an important part of my research work. Sometimes it is easier and faster to build proper debugging tools instead of using the standard ones available on the market. During the development of the electronics in our lab in Bologna we had several debugging instruments to test electronics, like:

- oscilloscope,
- pattern generator,
- logic analyzer.

We discovered in short time that the available instrumentations were not the best choice to debug the electronics of our readout chain. In fact not all the boards had the necessary test pins to connect the probes of these instruments, and the number of signals provided were not enough to give a complete picture of the system. Since buying new and powerful instruments would require money and the time needed to learn how to use them, we decided to build software tools for the debugging of our readout chain. So far the developed programs were specifically optimized for our electronics; we had all the hardware needed to develop them, we spent only time for the development.

In order to test, configure and control the FEE and readout electronics several programs have been developed:

- **RS232 program**: it allows to read the status of the board and to send commands from a standard PC.
- **CONFIG. program**: it configures all the registers of the readout chain chips,
- **monitor program**: it monitors the events being acquired,
- **decoding program**: it verifies and checks the data stored,
- **VME program:** it is used to upload the new firmware of CARLOSrx using the VME bus.

All the programs have been written using C/C++, bash scripting and TCL/TK for the GUI, they have been compiled and developed in this environment:

- SLC 4 (Scientific Linux CERN the LINUX distribution developed by CERN team, installed on all the machine used at Point 2)
- kernel 2.6.9-55.0.2.EL.cernsmp
- gcc version 3.4.6 20060404 (Red Hat 3.4.6-8)

4.1 The RS232 program

This program has been developed in order to manage the communication between a PC and CARLOSrx board using the RS232 port.

It is not an easy job to debug a readout chain of this complexity. Trying to find the source of possible problems can be a difficult task because a lot of chips are playing a role during the data acquisition. Not all of the boards provide test pins to connect an external analyzer instrument to study their status or, when the pins are available, it is not obvious how to connect the instrument due to the limitation of the space. CARLOSrx is the most accessible board in the chain and it has several "windows" where it is possible to monitor the status of the system. We used the following instruments as logic analyzers:

- **CHIPSCOPE software** from **XILINX**,
- **RS232 program** developed by us.

CHIPSCOPE uses the JTAG programmer provided by XILINX to interface itself with the FPGA. If the firmware installed on the FPGA does not use all the available resources, it is possible to store the history of some signals and to check them during the run using **CHIPSCOPE program**. Since the firmware being debugged was already using most of the available resources of the FPGA, only a few were available for debugging giving the possibility to monitor only a few signals, not enough to have a complete picture of the situation. Moreover since the FPGA programmer can be used with one FPGA at the a time, we decided to build an other tool using the RS232 port installed on the CARLOSrx. The RS232 program made it possible to query CARLOSrx and receive in the PC monitor the status of the FIFOs and to check the values of some other registers; obviously the firmware and the program needed to be prepared accordingly in order to provide the correct instructions/values.

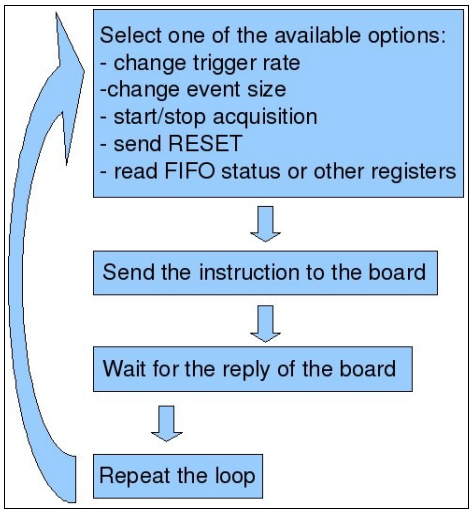


Fig. 4.1: state machine of the RS232 program.

All the instructions sent by this program were analyzed and executed by the **RS232 block** in the firmware of CARLOSrx.

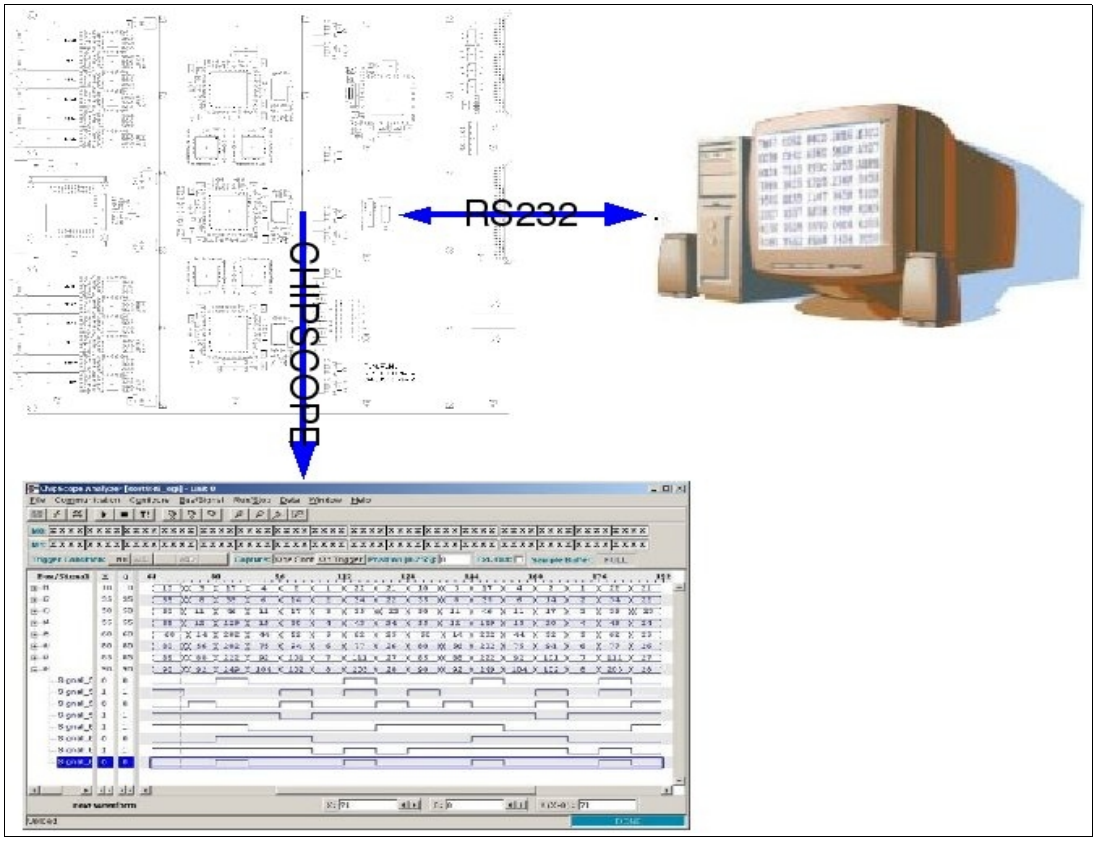


Fig. 4.2: CHIPSCOPE and RS232 connected to the board

An intensive use of this program has been carried out during the development of the internal data generator installed on CARLOSrx with the purpose of emulating the SDD modules and FEE. In fact in order to test the hardware components of the CARLOSrx board and the firmware behavior without the full readout chain, we decided to install an internal data generator per each SDD module on the CARLOSrx FPGA.

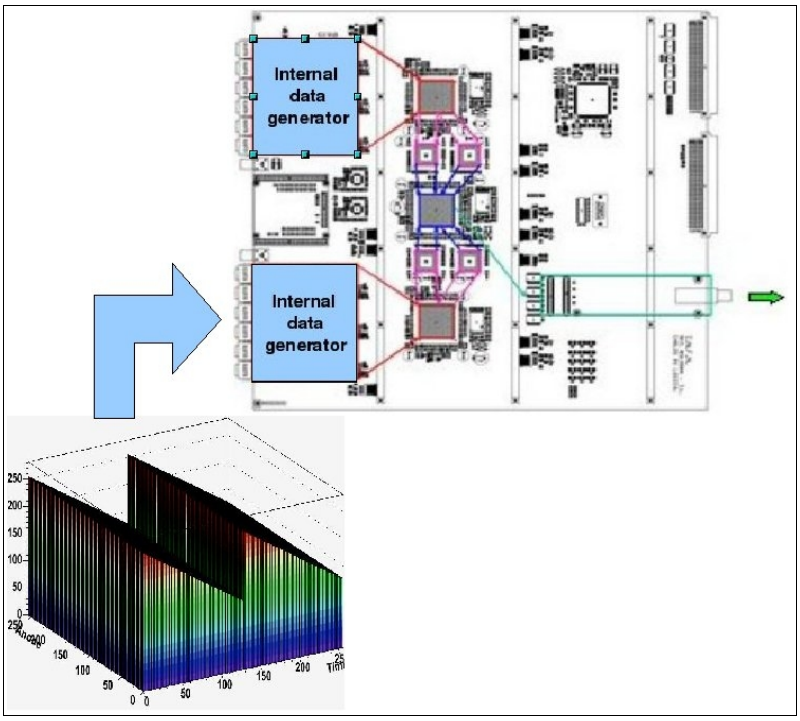


Fig. 4.3: internal data generator concept

We designed a special block in the two input FPGAs to use known data as input to the firmware under test. In this way we were able to test different features of the firmware without having the other components of the chain.

Using the RS232 program we were able to send configuration parameters to CARLOSrx during the acquisition:

- to change the event size produced by the internal data generator,
- to change the trigger rate,
- to stop and start the trigger.

The same program has been upgraded adding several instructions in order to be used during the commissioning of the board. We added the possibility of sending the RESET signal and reading the status of different buffers installed on CARLOSrx or the value of the registers during the run.

4.2 The configuration program

The time needed to configure the electronics is a challenge that is important to not underestimate, especially for a project like this one where the readout chain is composed by a lot of registers that must be configured in a fast way before the start of each run. The configuration of the different chips is done via JTAG: CARLOSrx receives all the necessary informations and sends them to the CARLOS board that configures through a JTAG chain the FEE.

The simplest configuration is:

- 1 SDD module, 8 pairs of P-A chip,
- 1 CARLOS,
- 1 CARLOSrx.

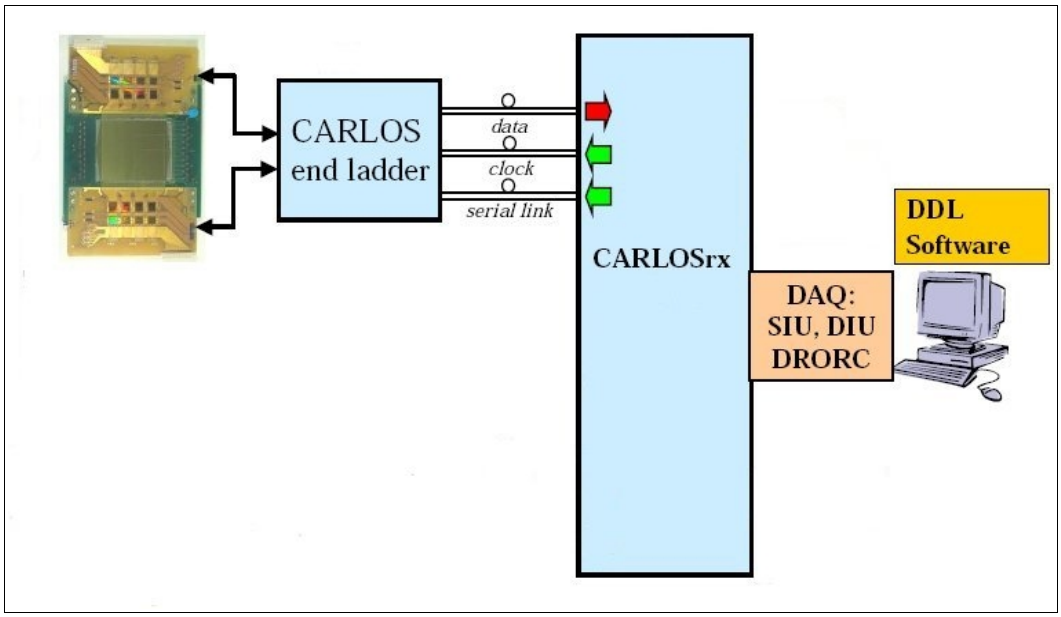


Fig. 4.4: 1 SDD module, 1 CARLOS chip, 1 CARLOSrx board

Each chip has several registers that must be configured.

AMBRA.config:		PASCAL.config:	
READ_REG	Y	READ_REG	Y
VALUE SOP DELAY	160	VALUE VREF CONTROL DAC	75
VALUE WCNT STOP	255	VALUE CALIBR DAC	511
VALUE RCNT STOP	255	VALUE SELECT CALIBRATION	21
WRITE BASELINE	N	VALUE GAIN CONTROL	3
		SET AM FULL	Y
		SET AM HALF	N
		SET ADC FULL	Y
		SET ADC HALF	N
		READ AM ADC FREQ	Y
CARLOS.config:		**CARLOSRX.config:**	
AL RIGHT	255	VALUE AL CARLOSRX	255
AL LEFT	255	VALUE TRIGGER	209
TL RIGHT	0		
TL LEFT	0		
TH RIGHT	0		
TH LEFT	0		
ENABLE 2D	Y		
READ REG	Y		
STOP IF ERROR	0		

Fig. 4.5: registers for each chip to be configured

The complete readout chain is composed by:

- **2080 pairs of P-A chips**,
- **260 CARLOS**,
- **24 CARLOSrx**.

This means **more than 30'000 registers** to be configured: it is clear that it is impossible to do this operation by hand. An other issue is that all the configuration files containing the values for several registers are written in hexadecimal format, that is not the most readable format for a human being.

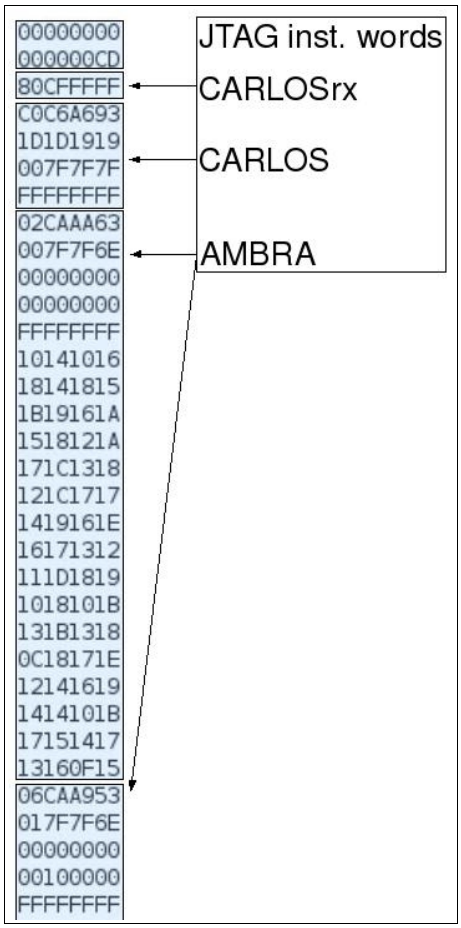


Fig. 4.6: example of file used by the program to configure the electronics

From the time of the first prototype of the readout board, the development of a configuration with the following features has been addressed:

- to provide a graphic interface to read easily the configuration values,
- to generate all the necessary files;
- to start the configuration;
- to complete all these steps in a short time;

We developed a program to provide configuration files for different types of runs (pedestal, injectors, testpulse, physics) and to use them to configure the electronics. When possible the configuration of the front end electronics is performed in parallel to reduce the time needed for this operation: in this way several boards can be configured at the same time. Since the software is installed in 4 different LDCs connected to the CARLOSrx boards, it is possible to configure more than one board at the same time.

These are the main operations for the user:

1. to decide which board configure,
2. to select the registers and the value,
3. to generate the configuration files,
4. to start the configuration process.

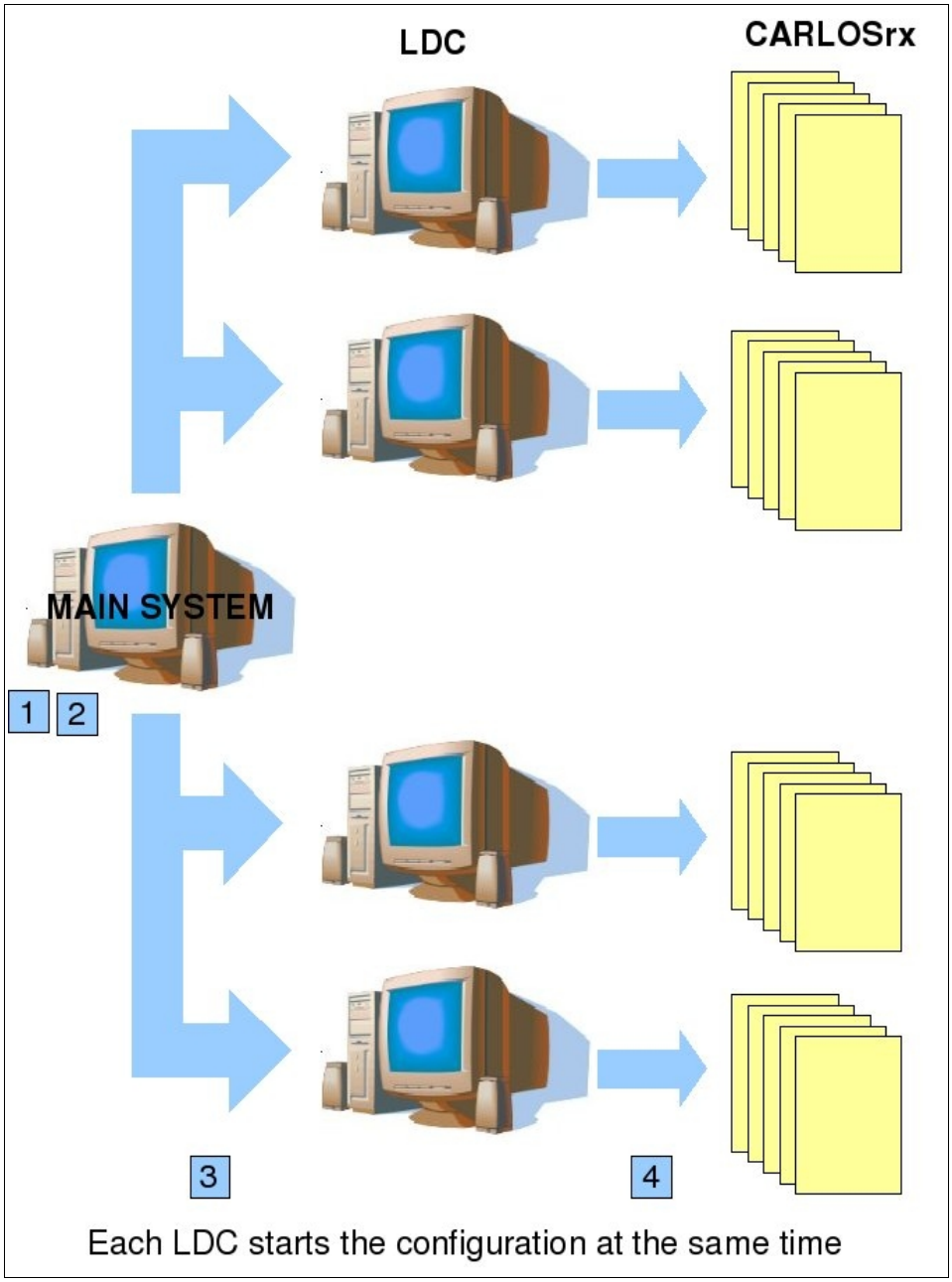


Fig. 4.7: JTAG configuration

The configuration of all the front end electronics takes a few seconds and, at the end of the process, a report is printed on the screen to show if each card has been correctly configured.

During the **GLOBAL run**, when the SDD readout chain is included in the final DAQ infrastructure, the configuration of the electronics is done automatically at each start of run by the **ECS**, using the files provided by our program. In he following picture it is possible to note the report regarding different boards being configured at the same time; the labels "**OK TOP2**", "**OK MED5**" ... refer to the name of the CARLOSrx boards configured (the name contains a clear indication of its position in the 3 different VME crates). The time stamp on the left it is the same for all the boards, showing that different boards can be configured at the same time.

```
19:30:29  aldaqpc083  ReadoutShell  **** **** **** **** OK TOP2 **** **** **** ****
19:30:29  aldaqpc099  ReadoutShell  NUMBER WORDS:\t 150240
19:30:29  aldaqpc099  ReadoutShell  **** **** **** **** OK MED5 **** **** **** ****
19:30:29  aldaqpc098  ReadoutShell  NUMBER WORDS:\t 75120
19:30:29  aldaqpc098  ReadoutShell  **** **** **** **** OK BOT4 **** **** **** ****
19:30:29  aldaqpc083  ReadoutShell  Starting /date/rorc/Linux/FeC2_0 -m0 -c0 -T 200000 -f /local/home/sdd/JTAG/res.txt -v
19:30:29  aldaqpc083  ReadoutShell  FeC2 ended. Processing time = 0.002061 seconds. Exit code = 0
19:30:29  aldaqpc028  equipmentLis  Arming RorcData: eqId=1313, source = detector electronics, revision number=4, serial number=5019, channel number=1
19:30:29  aldaqpc084  ReadoutShell  NUMBER WORDS:\t 137720
19:30:29  aldaqpc084  ReadoutShell  **** **** **** **** OK MED1 **** **** **** ****
```

Fig. 4.8: JTAG configuration of different boards.

A GUI has been added to facilitate the configuration operations, in the following pictures some screen shot of the GUI are shown.

Fig. 4.9: image of the GUI program to configure the electronics

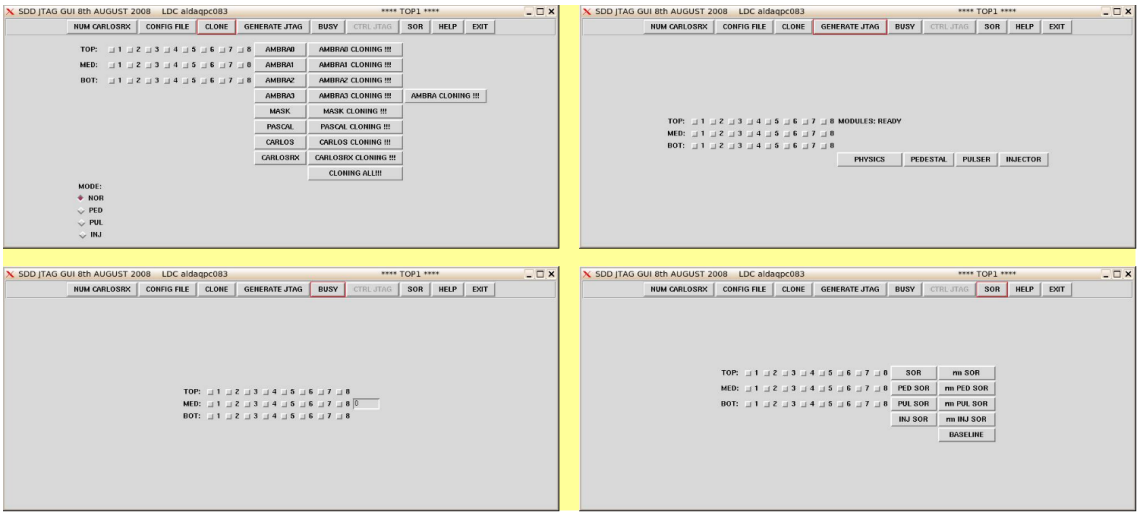


Fig. 4.10: image of the GUI program to configure the electronics

4.3 The monitor program

Normally in experiment like ALICE there are several programs/systems to control in real time the status of different parts of the electronics or mechanics. The same principle holds also for the data acquisition: there is a program that constantly checks the quality of the data.

Once the run is started it is important to have a way to understand if the electronics have kept the right configuration, or if the configuration is the one we think we have loaded. It is not sufficient to look only at the trigger rate and at the event size, so a tool is needed to control the data during the run. A decoding program is useful to analyze the data stored in the disk, but this kind of approach can be time and space consuming. There is the risk of storing GBytes of wrong informations if we do not check the data quality during the run. A monitoring on-line program gives the user the opportunity to control the quality of the data during the acquisition, in this way the user can constantly check the status of the readout chain without waiting until the end of the run

The working principle is rather simple:

1. the acquisition is started,
2. the monitor program takes one event and gives a graphic representation of it,
3. it waits a timeout and then it takes an other event, until the program is stopped.

We developed a program that uses ROOT framework for the graphic part. It works following the just described steps.

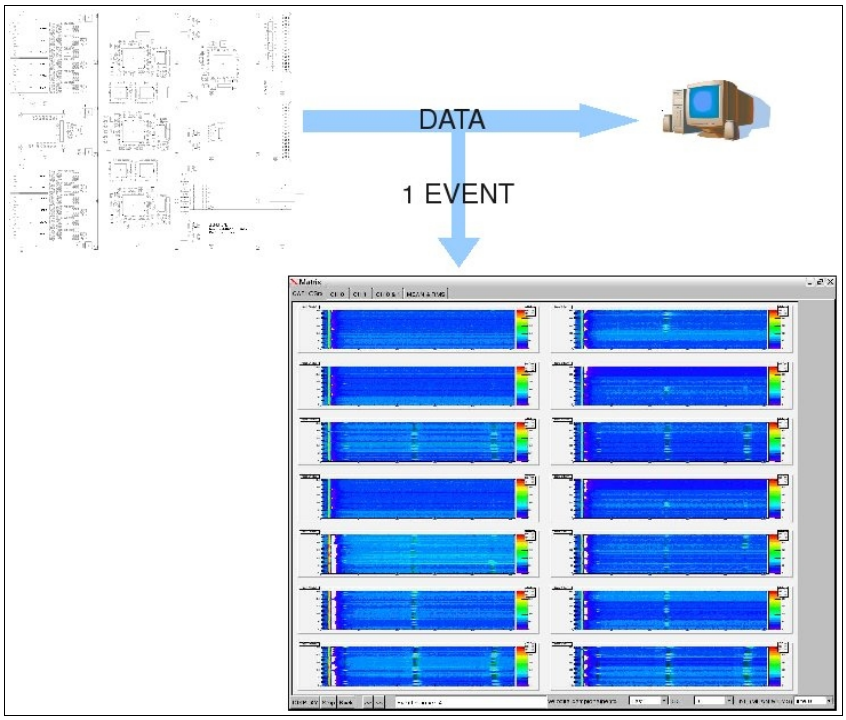


Fig. 4.11 : on-line monitor program

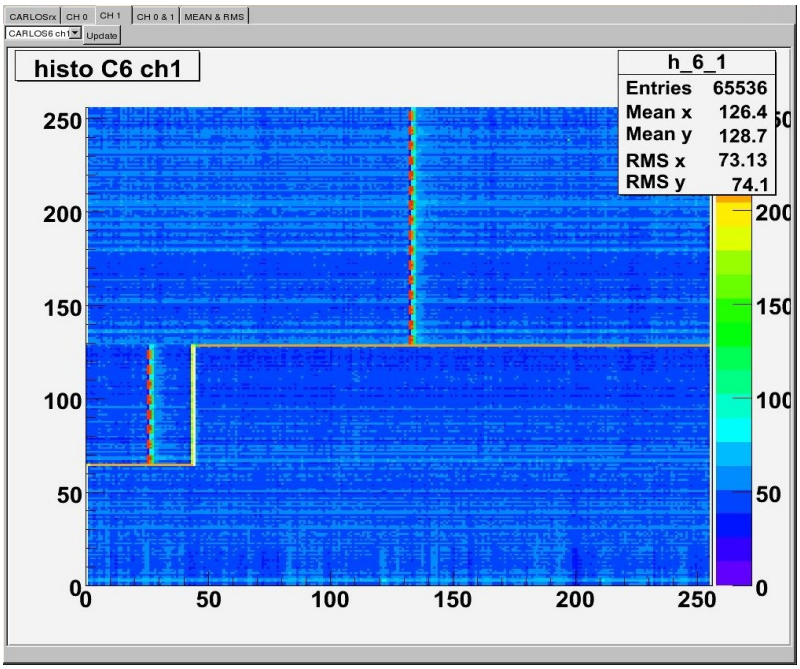


Fig. 4.12: example of bad module configuration discovered after few events with the monitor program. It is possible to see in this picture that one P-A chips pair does not display the testpulse signal, and another pair has lost the synchronization with the other two pairs.

During the first scan phase of the SDD barrel this software has been very useful searching for bad modules. Indeed we discovered that some of the chip pairs P-A were not able to keep the JTAG configuration. We discovered this problem in short time sending testpulse and prepulse signals to all the chips and using the monitor program to see if these signals were correctly generated.

4.4 Decoding program

The decoding program played an important role during the entire development of the firmware, starting from the first release of the readout board.

This software decodes the words coming from CARLOSrx in order to reconstruct the event and operates several checks on the data:

- it controls the number of header and the footer words in each event,
- it controls the number of end of row summaries (256 per event), special words stored at he end of each anode row containing summary informations of the data,
- it checks the words of the CDH,
- it checks the status words,
- obviously, it decodes the data.

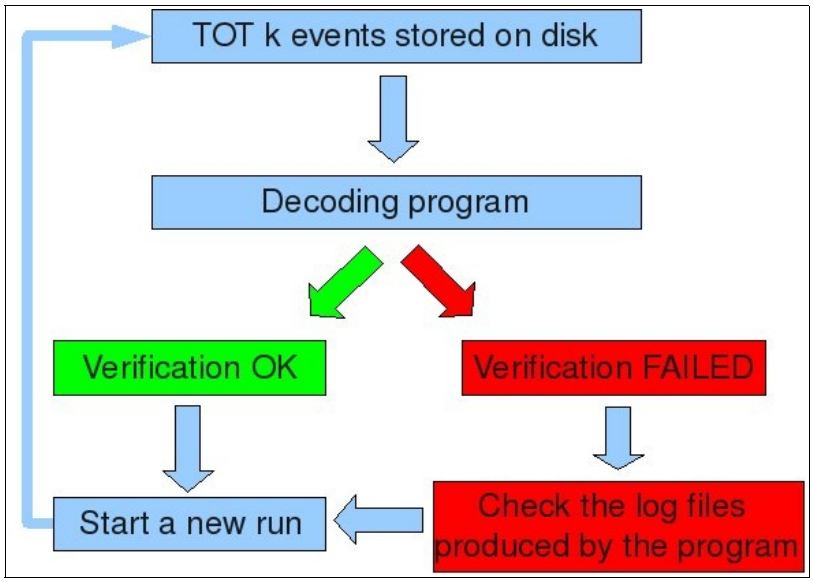


Fig. 4.13: decoding program working principle

4.5 The VME program

This program has been developed in C using the ***CCT VME driver*** for the VME part.

A 6U VME CPU is installed in each VME crate acting as a server and controlling the VME bus. The software to upload the firmware has been installed on each CPU. Its working principle is rather simple:

- it takes as input the address of the CARLOSrx board and the file with the firmware,
- it starts to upload the information on the VME bus, then the board with the selected address starts to read the data.

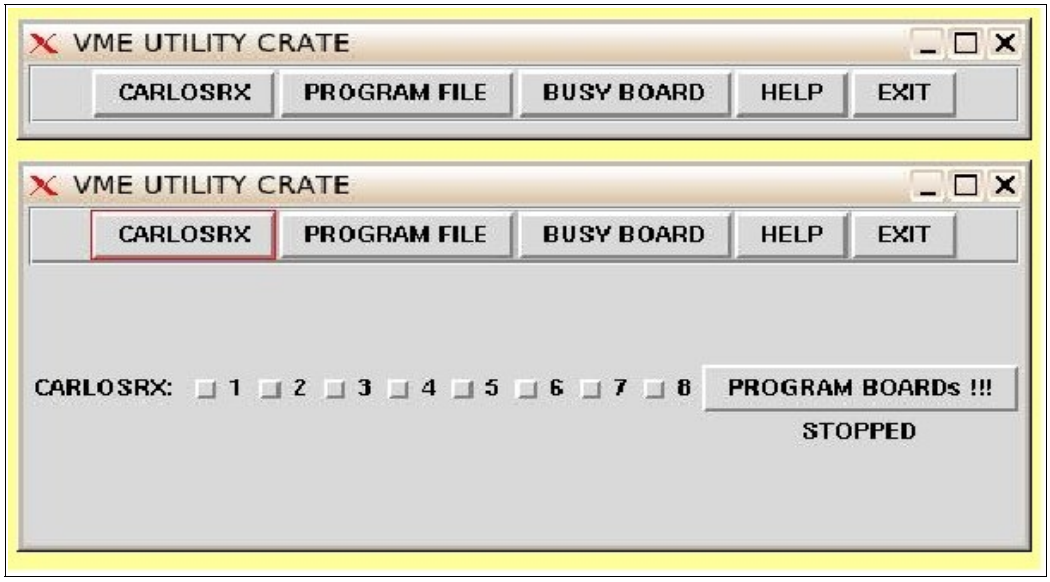


Fig. 4.14: program developed to load the different firmware in the FPGAs

For the time being we are able to program only the FPGAs, not the PROMs: this means that each time the crate is powered off the FPGAs lose their configuration and need to be re-programmed at the next power on. For this reason a script has been developed to program all the boards inside the crate.

5. CARLOSrx at CERN

At the beginning of 2007 the entire SDD barrel has been installed at CERN to start the integration of the complete readout chain in the ALICE experiment. In order to do that we prepared a test setup at CERN installing all the necessary electronics to read out the complete SDD barrel. This was the first time we tested all the 260 SDD detectors assembled together.

During this period we worked in contact with the DAQ and TRIGGER teams. After almost 2 years of work we have integrated successfully, the electronics and the detectors in the experiment, as described in the following sections.

5.1 Commissioning of the SDD barrel

The SDD barrel has been transported from Torino, where it has been assembled, to CERN site, Point 2 (see Fig. 5.1), where the ALICE cavern is placed.

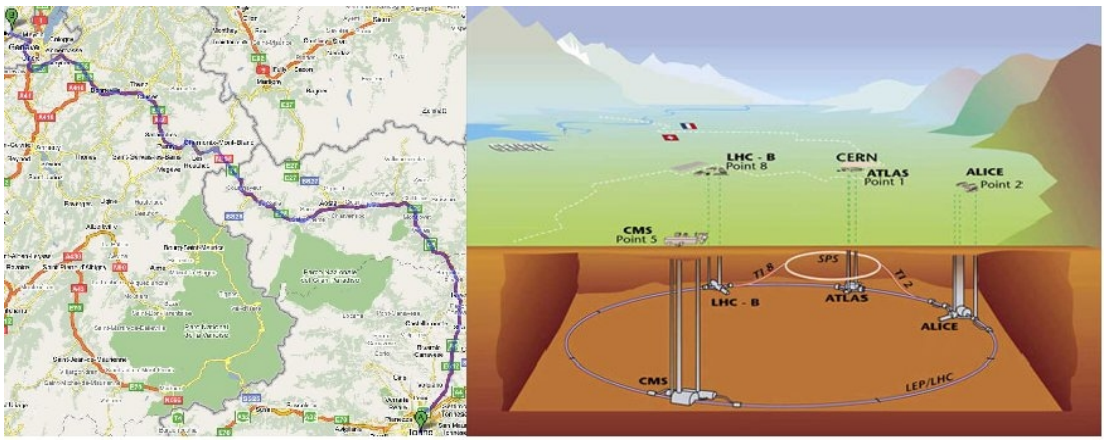


Fig. 5.1: Point 2 ALICE.

First of all, before starting the integration of the SDD in the **ALICE** detector, we needed to be sure of the good status of the barrel after this long trip, so we prepared a test setup composed by:

- **SDD barrel:** 260 SDD detectors.
- **2 CARLOSrx board pairs**.
- **Trigger system**: 1 LTU installed in a standalone VME crate, to generate trigger sequences in order to debug the electronics in different situations.
- **DAQ framework**: 2 LDCs, one for each CARLOSrx.

We decided to start using only two CARLOSrx boards to readout the full SDD

barrel instead installing the complete read out chain, in this way it has been easier to identify possible problems and to study each single module. We acquired one ladder at a time checking the behavior of all the detectors and the related electronics attached to it.

In order to scan the full barrel we followed this scheme:

1. to connect one ladder to the low voltage and to one CARLOSrx board,
2. to start the acquisition of 100 Kevents,
3. to check the data with the monitoring program to control the behavior of the modules,
4. to complete the scan of the full barrel, repeating point 1, 2 and 3.

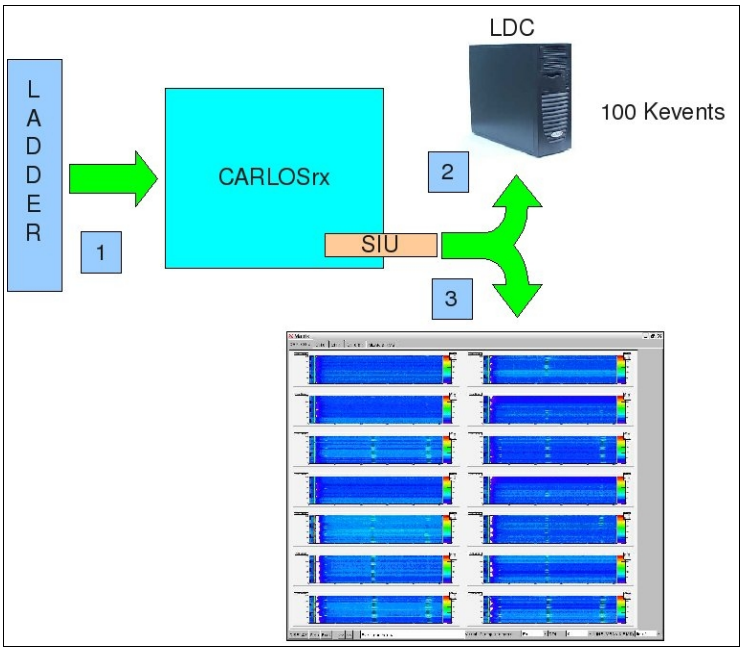


Fig. 5.2: test setup

We scanned all the modules of the SDD barrel. The first scan gave us good results: all the modules were working properly. The second step of the test was to connect the high voltage and to control the modules using the testpulse signals. We used the same scheme as before, but this time we discovered some problems. We noticed that the testpulse was not correctly generated in some detectors and we also noticed that some modules were not able to keep the JTAG configuration when running at high frequencies. After completing the test we had the 97.6% of the full SDD barrel working correctly. For the rest of the modules we are still trying to find a solution to solve this loss of parameters configuration.

The next step was to start the acquisition of more than one ladder at the same time in order to check if there was any interference between adjacent ladders. To do so we had to upgrade the first hardware setup adding another CARLOSrx board and to configure

the DAQ system to receive data from two DDLs connected to a LDC each one, the configuration of this setup is described in the following paragraph. At the end of this test we concluded that there were no interferences between close ladders.

5.1.1 DAQ/ECS integration

Once we finished the scan test on the SDD barrel and we reached a stable configuration we started to integrate the SDD system with the different infrastructure of ALICE. We started integrating the readout electronics of SDD with the DAQ system. The first step was to acquire data coming from two ladders using 2 LDCs at the same time, when the acquisition is controlled by the ECS. The test setup was composed by:

- 2 SDD ladders,
- 2 CARLOSrx,
- 2 LDCs,
- ECS, installed in another PC.

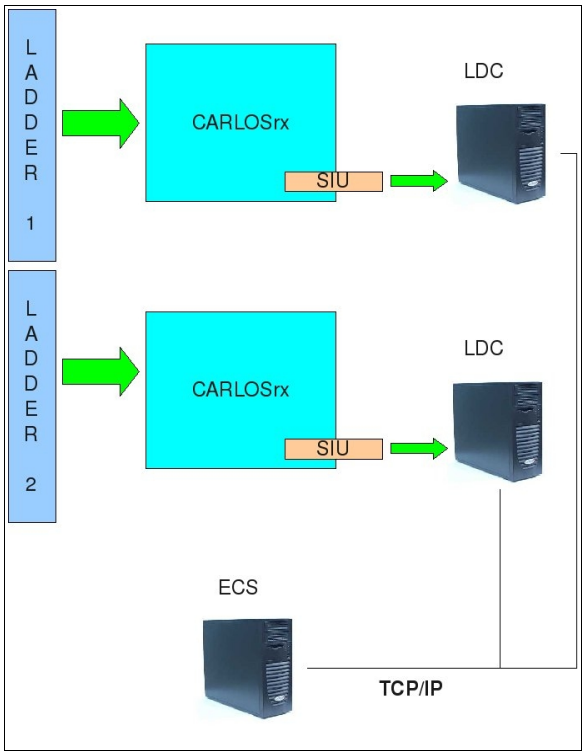


Fig. 5.3: test setup built to acquire data from two LDCs, controlled by ECS.

At CERN the SDD was the first detector to use two LDCs at the same time to acquire data, so it was a good test session also for the DAQ team. After configuring the DAQ software and the front end electronics the acquisition ran smoothly and we acquired million of events in a stable configuration. The next step was to use the ECS.

We had to provide scripts to configure the electronics from the ECS: in fact, as described in Chapter 1, the ECS has to automatize all the commands at start and during the run in order to avoid any user operation. Once we have provided all the programs and the scripts requested by ECS to work properly and populated the DAQ database, where all the SDD informations are stored, we started the run without problems.

5.1.2 Trigger test

The SDD system has to behave correctly also when receiving erroneous trigger sequences (for more details on the different errors look at the **AppendixB.1**). To verify its behavior in this situation, the trigger team tested the readout electronics with different trigger sequences. As you can see at http://epweb2.ph.bham.ac.uk/user/krivda/ALICE/ we positively replied to all the questions. Indeed CARLOSrx behaves correctly when receiving bad trigger sequences, providing the required operation (for example discarding the event and filling properly the fields in the CDH to notify to the DAQ and to the OFFLINE team the type of error occurred).

5.2 Integrations of the SDD barrel in the ITS

After completing all these tests, the barrel has been integrated into SSD barrel (a video of this can be found at http://www.youtube.com/watch?v=NHOvDs70Yiw), this operation took more than 4 hours.

5.3 CARLOSrx in position

During the installation of the complete ITS in the ALICE detector, we started the installation of the readout electronics in CR4 where one rack was reserved for our electronics. This rack has been installed with:

- 2 patch panels for TRIGGER and DAQ optical fibers,
- 3 VME crates to install all the CARLOSrx boards, the CPU VME and the busy board.

From this room CARLOSrx communicates with the FEE installed in the ALICE cavern through 780 optical fibers (data, instructions clock) and it sends the information towards the DAQ system using 24 DDL links.

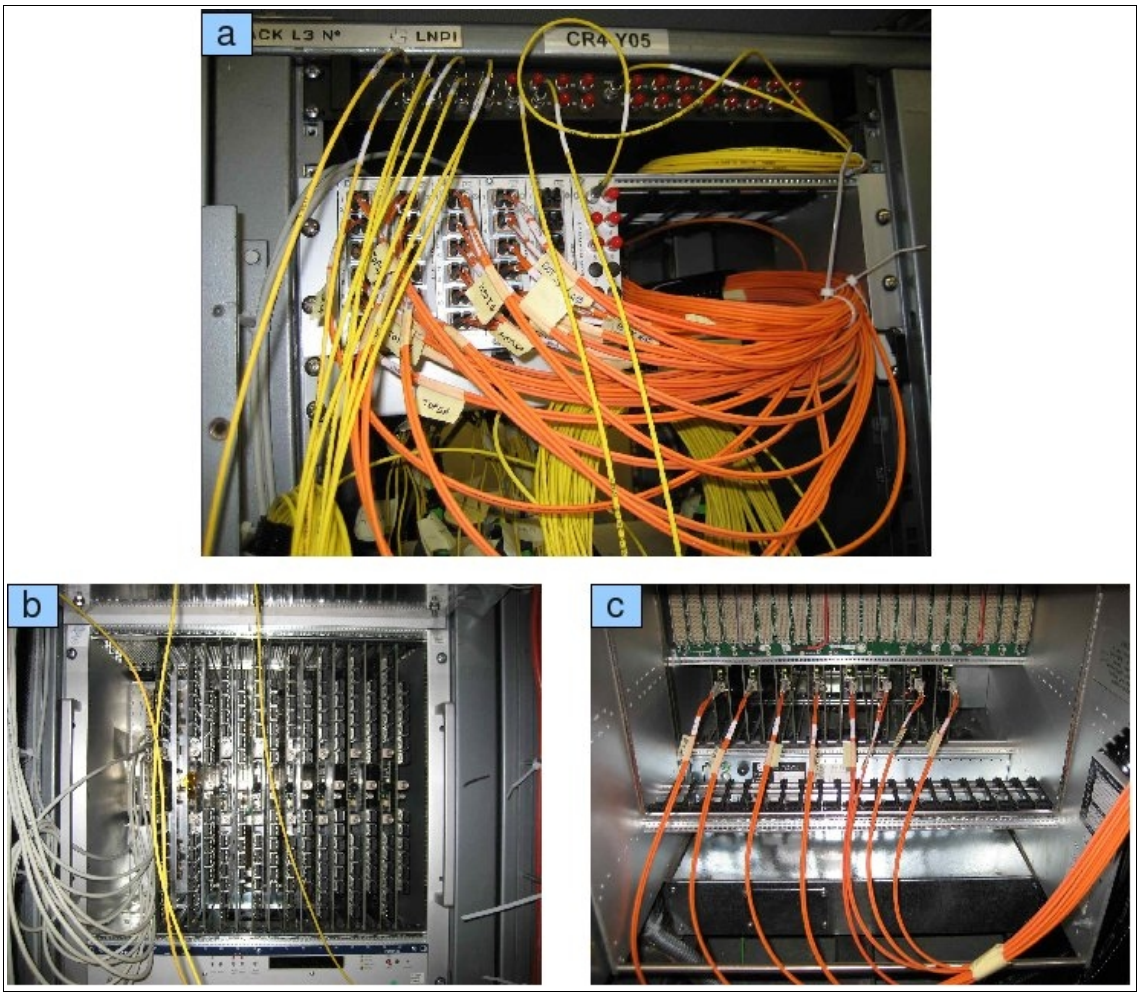


Fig. 5.4: a) patch panel DAQ and trigger optical fibers, b) CARLOSrx data and clock installed in the VME crate, c) back of CARLOSrx data, DDL link optical fibers.

Conclusions

The SDD barrel and its electronics have been successfully installed allowing us to participate in the data acquisition during the ALICE cosmic rays tests performed at CERN in the last year 2008.

The goals of this PhD work were:

- to develop a stable version of the CARLOSrx firmware in order to read data coming from 12 SDD detectors at the same time,
- to install all the components of the SDD readout chain at CERN to start the commissioning of ALICE,
- to develop software tools to configure the front end electronics and to monitor the system.

After several tests performed in our laboratory at Bologna and later at CERN a stable version of the CARLOSrx firmware has been delivered in time with the schedule assigned to the SDD detector.

In order to obtain such a result we followed several steps:

- First of all we developed a data emulator in order to test all the hardware components and several features of the firmware in absence of the SDD detectors.
- Once we produced a first release of the code we tested it using the real readout chain electronics (SDD modules plus CARLOSrx).
- We brought the SDD barrel at CERN and we built a test setup to verify its good status, discovering problems in some modules.
- Subsequently we developed new features of the firmware to satisfy the requests of the DAQ and TRIGGER teams to integrate our readout chain with the several infrastructures of the experiment.
- At the same time the changes in the software followed step by step the firmware upgrades, in order to constantly monitor the system and providing fast alert in case of errors.

Almost all the SDD barrel works as expected, indeed we had **97,6%** of all the detectors in acquisition at the end of 2008.

Starting from middle of October 2007 up to now the SDD detectors have been performed more than **10 Kruns,** running in total for **2178 hours** in STANDALONE and GLOBAL runs, (more details in table 5.1).

In STANDALONE the detector can acquire data with software triggers and test readout without interfering with the other detectors. So if for example one detector becomes busy, the other runs will be unaffected. On the contrary when one or more detectors enter the GLOBAL run, the situation is completely different. In this case they share all the run conditions and if one or more detectors get busy the run is no longer able to accumulate events. When in GLOBAL runs all the sub-events produced by all

the participating detectors are used to reconstruct the complete event.

Using ACORDE trigger (cosmic rays), ALICE stored *PETAbytes* of data used by different teams to study the performance of their detectors. During these runs all the different SDD detectors calibration parameters were stable and stored automatically after each calibration run in ALICE database:

- noise,
- gain,
- drift speed.

Analyzing all the data fine tuning of the charge conversion factor has been applied and the charge dependence on the drift length has been corrected and the complete ITS and TPC have been correctly aligned.

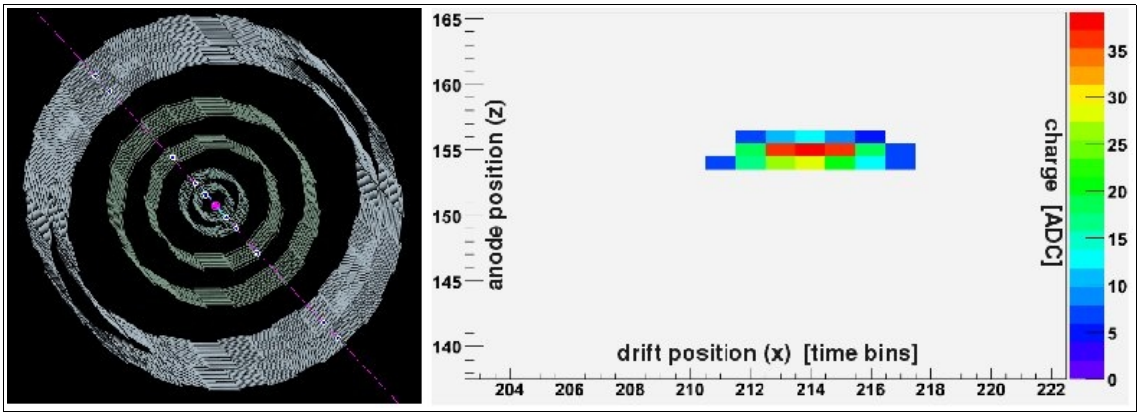


Fig. 5.5: detail of a particle detected by the ITS

In the night between ***11/12 September 2008*** one of the two beam was circulating in the LHC, and the ITS detected the interaction.

The tracks detected by the ITS are displayed in the figure 5.6.

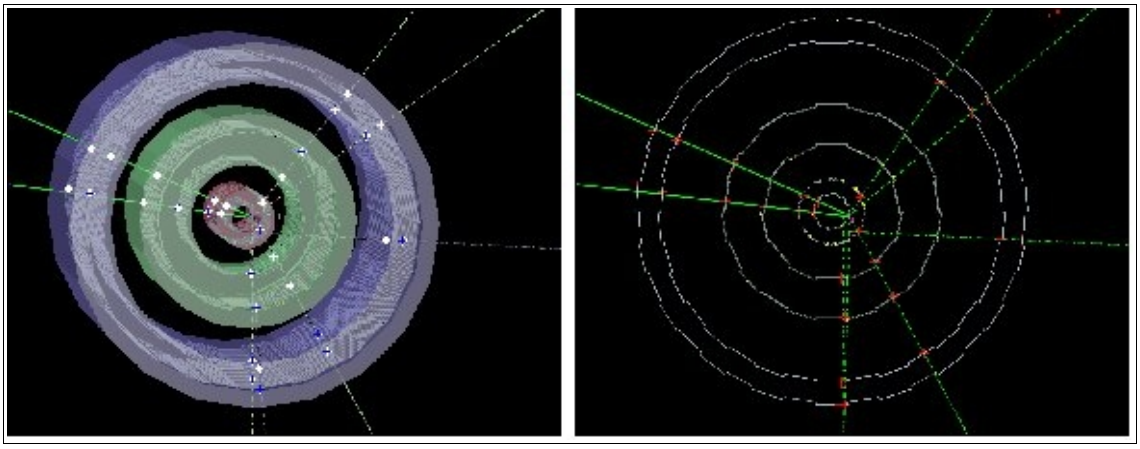


Fig. 5.6: ITS tracks, generated by particle interaction

Runs	Duration	SubEvents	Bytes	Bytes (recorded)
10938	2178 h	1226305082	763.1 TB	97.6 TB

Tab. 5.1: SDD run statistics.

Partition	Runs	Duration	SubEvents	Bytes	Bytes (recorded)
Standalone	8712	931 h	909875157	619.7 TB	3.2 TB
ALICE	1453	561 h	53300682	83.1 TB	65.1 TB
ALICE_TEST	103	131 h	13211095	15.7 TB	15.2 TB
PHYSICS	5445	482 h	67756722	27.5 TB	13.6 TB
TEST_1	57	41 h	28935076	14.6 TB	161.2 GB
TEST_2	65	33 h	153226350	2.5 TB	443.0 GB

Tab. 5.2: STANDALONE and GLOBAL run statistics.

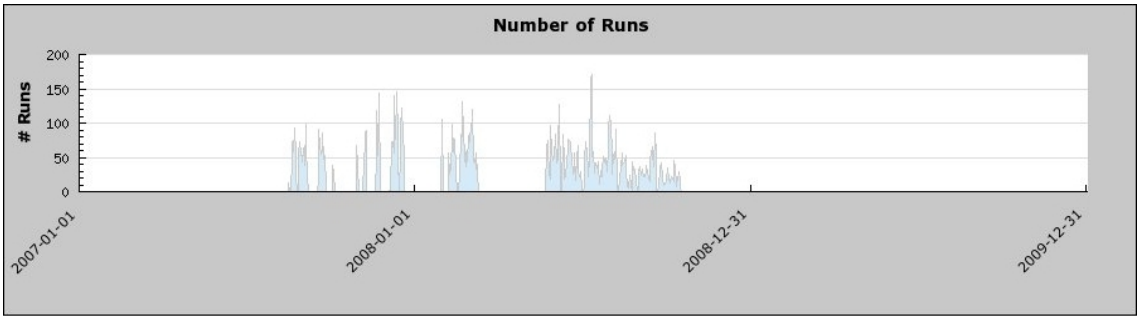


Fig. 5.7: number of run performed by SDD

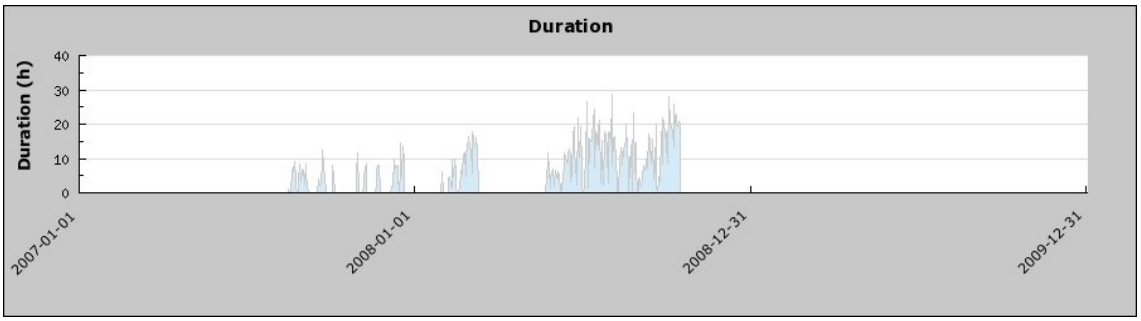
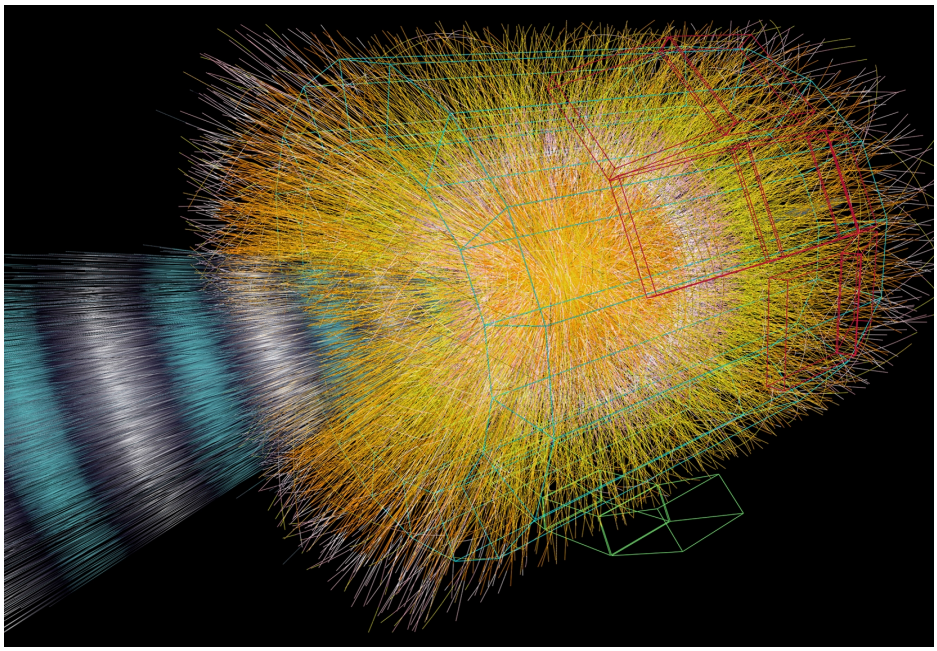


Fig. 5.8: hours of run performed by SDD

Currently we are working on new features of CARLOSrx firmware in order to recover the 2.4% remaining modules that are excluded from the acquisition. An upgrade of the configuration software is work in progress because we are expecting an upgrade of the DAQ infrastructure and the SDD readout chain will have 2 more LDCs, so in total 6, to distribute the total bandwidth.

For the time being we can conclude that SDD is ready for the first **p-p beam collisions**.



Appendix A

A.1 SIU signals

All of the signals of the FEE-SIU interface are synchronized to the foCLK signal.

- ***FbD[31..0] DATA[31..0] bi-directional***: the fbD[31..0] is a 32-bit wide, tri-state data bus. If the level of the fiDIR line is high, the data on these bus lines are transferred from the FEE to the SIU on a low-to-high transition of the foCLK when the fbTEN_N is active. The fbD[31..0] contains status words when the fbCTRL_N is active, otherwise it contains normal data words. If the level of the fiDIR line is low, the data on these bus lines are transferred from the SIU to the FEE on a low-to-high transition of the foCLK, when the fbTEN_N is active. The fbD[31..0] contains commands, when the fbCTRL_N is active, otherwise it contains normal data words.
- ***fbCTRL_N CONTROL bi-directional active low***: the fbCTRL_N is a tri-state management line for the data bus. The active level of this line indicates that the data word to be transferred between the FEE and the SIU is a command or a status word, depending on the direction of the information transfer. It is a status word, when the information is transferred from the FEE to the SIU, otherwise it is a command.
- ***fbTEN_N TRANSFER ENABLE bi-directional active low***: the fbTEN_N is a tri-state management line for the data bus. The active level of this line enables data to be transferred between the FEE and the SIU on the low-to-high transition of the foCLK. The direction of the information transfer depends on the level of the fiDIR line.
- ***FiDIR DIRECTION input high: FEE to SIU transfer***: the fiDIR is a management line for the data bus. The high level of this line indicates that the information is transferred from the FEE to the SIU on the fbD[31..0] lines, while the low level indicates the opposite direction of the information transfer. When the level of this line is high, all of the bi-directional lines are driven by the FEE, otherwise they are driven by the SIU.
- ***fiBEN_N BUS ENABLE input active low***: the fiBEN is a management line for the data bus. The inactive level of this line will put in high-impedance state the tri-state drivers of all the bi-directional lines of the FEE-SIU interface in the SIU and in the FEE.
- ***fiLF_N LINK FULL input active low***: the fiLF_N is a flow control line. The active level of this line indicates that the FEE shall stop the data block transfer and the FESTW(EOB=1) status word transfer to the SIU, because the SIU and/or the DIU and/or the RORC are busy. After this line becomes active, only one more data word or status word may be transferred from the FEE to the SIU.
- ***foBSY_N BUSY output active low***: the foBSY_N is a flow control line. The

active level of this line indicates that the FEE is not able to receive data block from the DDL. After this line become active, only one more data word may be transferred from the SIU to the FEE.

- ***FoCLK FEE CLOCK output***: the foCLK is a clock line. This free running clock is generated by the FEE for the synchronization of the information transfer between the FEE and the SIU. Please note, that for improved signal quality, the foCLK signal is terminated by a resistive load on the SIU card.

A.2 CDH fields explanation

- ***Block length*****: t**he block length is an optional field. It can be filled in by the detector readout electronics to indicate the total length of the data block including header and payload(s). The length must be expressed in bytes being transferred over the DDL. If not handled, the field must be loaded with hexadecimal FFFFFFFF to distinguish it from an erroneous zero value.
- ***Format version*****: t**he format version indicates which version of the present data format is used. The presence of this field provides the backward compatibility in case of change or upgrades. The content of the field must be compared with the current format version number (zero-extended to 8 bits).
- ***Trigger message*****: t**he L1 Trigger message consists of selected parts of the trigger L1 Message. This information is distributed over the TTC to the detector readout. When the ALICE Trigger system is not available, this field can contain any value.
- ***Event ID (1 & 2)*****: t**he LHC clock will supply the event identification in ALICE. This clock is distributed to all the detectors readout units by the TTC system used as trigger distribution network. The current LHC design foresees 3564 bunches in one orbit. The LHC clock identifies each bunch crossing within an orbit and signals the beginning of a new orbit. Currently the TTC foresees 12 bits for the bunch crossing number. The Trigger system shall include a cyclic counter of 24 bits to count the orbit. This scheme uniquely identifies every bunch crossing in a period of more than 20 minutes (224 x 88 μs = 1476 s = 24 minutes), which is sufficient for this purpose. Further identification will be added by the DAQ to uniquely identify one event in a run. The information stored in the Event ID fields (1 & 2) is transmitted by the CTP. It is distributed over the TTC in a dedicated part of the L2a Message and received via the TTCrx chips. When running without the ALICE Trigger system, the Event ID 1 field must be set to zero and the Event ID 2 must contain an incremental, unsigned binary number, to be reset at FEE reset.
- ***Block Attributes*****: t**he block attributes is an optional field that can be used freely by the detector groups to encode specific information such as the event type. If unused, this field should be set to zero.
- ***Participating Sub-Detectors*****: t**he mask of participating detectors is a mandatory field. Its information is produced by the CTP only while handling software triggers (Test Classes). It is distributed over the TTC in a dedicated part of the L2a Message and received via the TTCrx chips. The received value must be copied as-is in the “Participating Sub-Detectors” field. When running without the ALICE Trigger system, the “Participating Sub-Detectors” field can be loaded with any value.
- ***Status and error bits*****: t**his is a mandatory field, to be loaded by the readout electronics under all running conditions. An error or status condition that occurred before, during or right after FEE readout must be signaled by setting to

one the corresponding bit(s) of this field.

- ***Mini-Event ID*: l**ocal event identification must also be included in the common data format for cross-verification with the global event identification. This local event identification is the value of the local BC counter at the time the detector has received the trigger L1 signal. The counter is a part of the TTCrx chip. The local bunch-crossing counters of all the TTCrx chips of the experiment must be synchronous. A key issue is to resynchronize them at regular intervals to ensure that this synchronism is maintained. The solution chosen is to use the mechanism foreseen by the TTC system. The local bunch-crossing counter in the TTCrx chip is automatically reset by a fast signal synchronous with the LHC orbit. The LHC orbit signal is delivered by the TTCmi module. This signal is then sent over the TTC as a short-format broadcast signal. Proper usage and setting of the TTCvi module will guarantee that the TTC Rx chip receives this reset command by the end of the LHC extractor gap. The TTCvi provides four priority levels for data transmission. The bunch counter reset command uses the highest priority (level 0). The Mini-Event ID is a mandatory field. When running without the **ALICE** Trigger system, the Mini-Event ID field must be set to zero.
- ***Trigger classes (Low & High)*: f**or physics triggers, the bits encoded in the Trigger Classes Low and Trigger Classes High fields are taken as-is from the trigger L2a Message. When running without the **ALICE** Trigger system, these two fields can contain any value.
- ***Region Of Interest (ROI) (Low & High)*: t**he ROI data is distributed over the TTC system. The value - if available - should be stored in the ROI Low and ROI High fields. When running without the ALICE Trigger system, the ROI Low and ROI High fields can contain any value.

Appendix B

B.1 Erroneous trigger sequences

1. **Spurious L0 Error**: a spurious L0 trigger received during the L1 Decision Time - the interval that follows a correctly received L0. The state diagram of the state machine is shown in Figure B.1; the L0 BUSY Timer defines the L1 Decision Time; each received L0 input, correct or spurious, triggers the L1 Window Timer that defines the arrival time of the corresponding L1 signal.

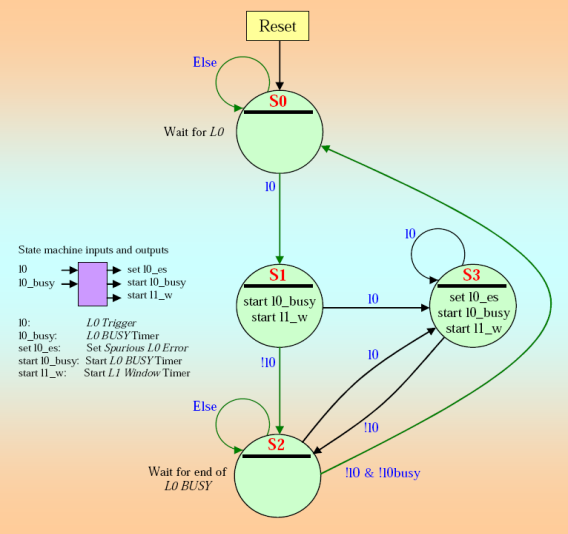


Fig. B.1

2. **Spurious L1 Error**: a spurious L1 trigger received outside of the L1 Time interval - the interval that follows a correctly received L0 signal. The state diagram of the state machine is shown in Figure B.2; the L1 Window Timer is triggered by the previous L0 input (see B.1); each received L1 input, correct or spurious, triggers the L1 Message Window Timer and the L2 Message Window Timer that define, respectively, the arrival time of the corresponding L1 Message and the L2a Message/ L2r Word.

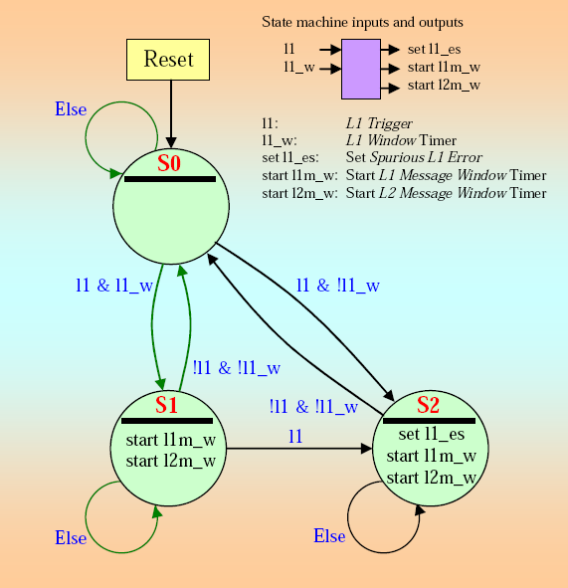


Fig. B.2

3. **L1 Message Data Error**: an L1 Message data word not preceded by the L1 Message header (word 1). The state diagram of the state machine is shown in Figure B.3; the logic generates the L1 Message Strobe signal - required by other error-detecting circuits - whenever the L1 Message Header (word 1) is received.

4. **Incomplete L1 Message Error**: less than 4 L1 Message data words received following the L1 Message header (word 1). The state diagram of the state machine is shown in Figure B.3. Note 2: Just a reminder that the format of sub-detector L1 Message is a programmable option: the message could be completely suspended; only the first word could be transmitted; or the transmission of the full message - all five words - could be enabled.

5. **Spurious L1 Message Error**: the L1 Message Strobe signal (see B.3) received outside of the L1 Message Window interval (see B.2). The state diagram of the state machine is shown in Figure B.5; the identical logic circuit is used for both the L1 and the L2 Messages.

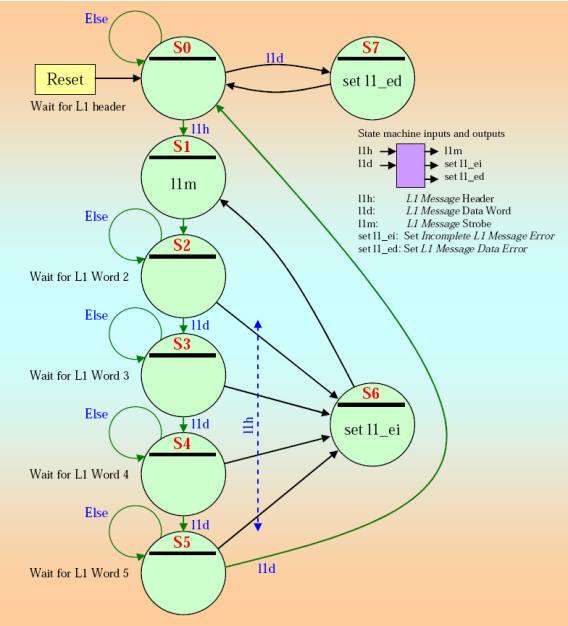


Fig. B.3

6. **Missing L1 Message Error**: the L1 Message Strobe signal (see B.3) has not been received during the current L1 Message Window interval (see B.2). The state diagram of the state machine is shown in Figure B.5.

7. **L2a Message Data Error**: an L2a Message data word not preceded by the L2a Message header (word 1). The state diagram of the state machine is shown in Figure B.4; the logic generates the L2 Message Strobe signal - required by other error-detecting circuits - whenever the L2a Message Header (word 1) or the L2r Word is received.

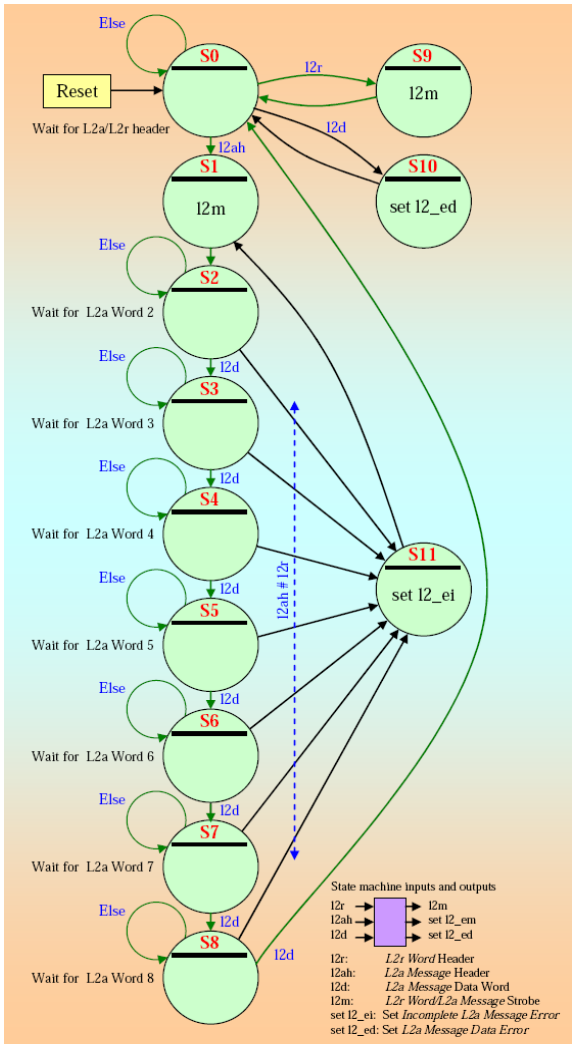

Else
S0
Reset
Wait for L2a/L2r header
l2r
S9
l2m
l2d
S10
set l2_ed
l2ah
S1
l2m
Else
S2
Wait for L2a Word 2
l2d
Else
S3
Wait for L2a Word 3
l2d
Else
S4
Wait for L2a Word 4
l2d
Else
S5
Wait for L2a Word 5
l2ah ≠ l2r
S11
set l2_ei
l2d
Else
S6
Wait for L2a Word 6
l2d
Else
S7
Wait for L2a Word 7
l2d
Else
S8
Wait for L2a Word 8
l2d
State machine inputs and outputs
l2r
l2ah
l2d
l2m
set l2_em
set l2_ed
l2r: L2r Word Header
l2ah: L2a Message Header
l2d: L2a Message Data Word
l2m: L2r Word/L2a Message Strobe
set l2_ei: Set Incomplete L2a Message Error
set l2_ed: Set L2a Message Data Error

Fig. B.4

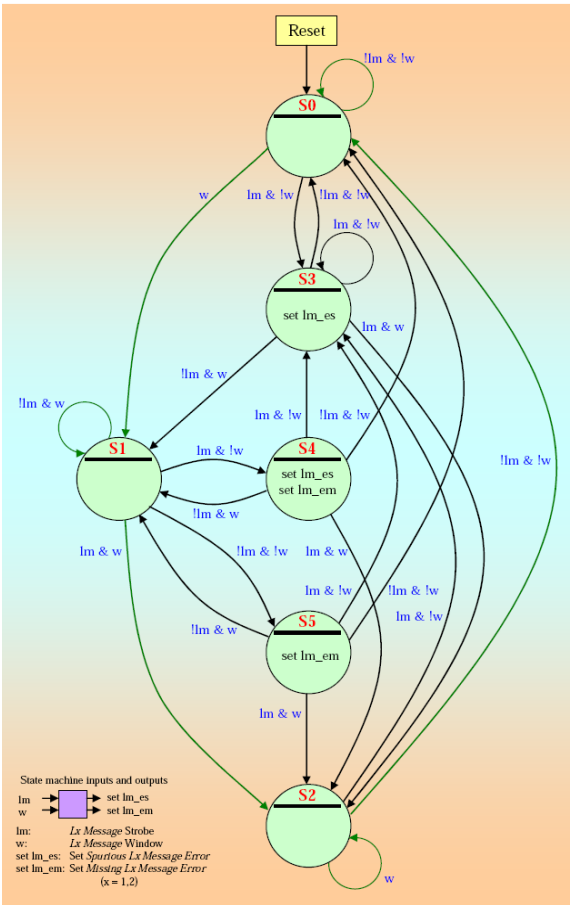


Fig. B.5

8. **Incomplete L2a Message Error**: less than 7 L2a Message data words received following the L2a Message header (word 1). The state diagram of the state machine is shown in Figure B.4.

9. **Spurious L2a Message/L2r Word Error**: the L2 Message Strobe signal received outside of the L2 Message Window interval (see B.2). The state diagram of the state machine is shown in Figure B.5; the identical logic circuit is used for both the L1 and the L2 Messages.

10. **Missing L2a Message/L2r Word Error**: the L2 Message Strobe signal has not been received during the current L2 Message Window interval (see B.2). The state diagram of the state machine is shown in Figure B.5.

11. **BC Identifier**: error The BC Identifier in the L2a Message/L2r Word different from the content of the TTCrx BC Counter at the time of transmission of the corresponding L1 signal.

12. **Prepulse Error**: reception of the PREPULSE signal in a bunch-crossing interval different from the Pre-pulse BC Interval; the interval is predefined and fixed for each

sub-detector. Note 3: Another reminder: the BC Counter in the TTCrx chip counts "shifted" time, set to give a correct bunch-crossing number at the arrival of the L1 signal; at the arrival of the PREPULSE signal, the counter shall read, approximately, (Pre-pulse BC Interval - L1 Time).

13. **Calibration Trigger Error**: reception of the Calibration Trigger signal in a bunch-crossing interval different from the Calibration Trigger BC Interval; the interval is predefined, fixed and the same for all sub-detector.
14. **L1 Message Content Error**: a number of possible inconsistencies/contradicting bit settings in the current L1 Message, most likely caused by the hardware or transmission errors. Examples: inconsistent/unexpected setting of the RoC[4..1] bits; the inconsistent setting of the ClT and the L1SwC bits; somewhat more ambitious checks could include data comparisons between the contents of the L1 Message and the corresponding L2a Message. Note 4: A reminder and a warning: the final formats of the L1 Message and the L2a Message are presented in [3]; some not fully up-to-date versions exist in earlier documents.
15. **L2a Message Content Error**: a number of possible inconsistencies/contradicting bit settings in the current L2a Message, most likely caused by the hardware or transmission errors. Examples: all the examples given in 14.; the inconsistent setting of the L1SwC and the L2Cluster[6..1] bits; inconsistent/unexpected setting of the Detector[24..1] bits; the BCID[12..1] content out of range - larger than 3563; irregular counting of the OrbitID[24..1]; etc. .

Appendix C

C.1 2D algorithm

The 2D algorithm makes use of two threshold values:

- a high threshold T_H for cluster selection;
- a low threshold T_L so to collect information around the selected cluster.

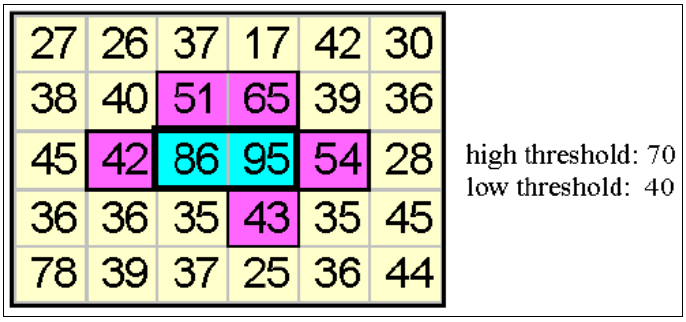


Fig. C.1

The algorithm retains data belonging to a cluster and around a cluster in the following way :

- the pixel matrix is scanned searching for values higher than the T_H value (70 in Fig. C.1);
- the pixels positioned around the previously selected ones are accepted if higher than the low threshold value T_L (40 in Fig. C.1), otherwise they are rejected;
- thus a cluster is defined and cluster values are saved exactly as they are: other pixels, not belonging to clusters, are discarded;
- if a pixel value higher than the T_H value is found but it has not pixel values higher than T_L around its value is rejected. This is the case of the 78 value on the bottom-left corner in Fig. C.1 which is discarded, even it its value is greater than the high threshold value.
- pixel values belonging to a cluster are encoded using a simple lookup table method, assigning long codes to non-frequent values and short codes to frequent symbols.

So far in Fig. C.1, after applying the 2D compression algorithm, only the shadowed values are stored, while the other value ares erased. The 2D algorithm is conceptually very simple to understand, but it is quite more complex than the 1D for what concerns hardware implementation. In fact having to perform a bi-dimensional analysis of the pixel array implies the need of storing all the information on a digital buffer on CARLOS, thus requiring a larger silicon surface and a higher cost.

C.2 Serial back-link

CARLOS is remotely controlled using a serial back-link coming from CARLOSrx board. 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. 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 links is good and transitions back to the SYNC state for normal operation. If, in the CHECK state, CARLOS sees 3 invalid cods (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 C.1 shows the list of command that CARLOS can receive through the serial back-link from CARLOSrx: they are all DC-balanced (same number of 1s and 0s).

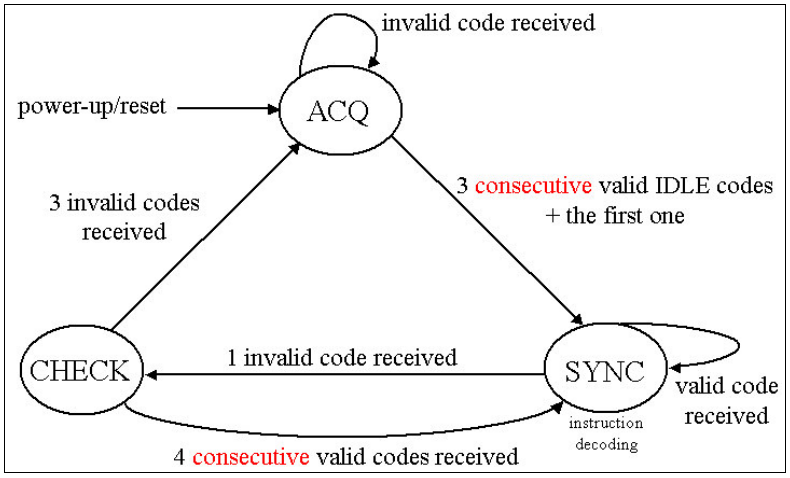


Fig. C.2: synchronous serial back-link state machine decoding

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
L2 reject	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 he charge injectors
Prepulse50	1001|0101	Prepulse output is asserted for 2 clock cycle towards he charge injectors
Prepulse75	1001|0110	Prepulse output is asserted for 3 clock cycle towards he charge injectors
Prepulse100	1001|1001	Prepulse output is asserted for 4 clock cycle towards he charge injectors
Prepulse125	1001|1010	Prepulse output is asserted for 5 clock cycle towards he charge injectors
Prepulse150	1001|1100	Prepulse output is asserted for 6 clock cycle towards he charge injectors
Prepulse175	1110|0001	Prepulse output is asserted for 7 clock cycle towards he charge injectors
Prepulse200	1110|1000	Prepulse output is asserted for 8 clock cycle towards he charge injectors
Idle	0011|1001	Used for link synchronization only and when there is no command to send
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_stop 1 to 0
JTAG	01|**trst**| not(**trst**)|**tms**| not(**tms**)|**tdi**| not(**tdi**)	JTAG values for one tck period

Tab. C.1

ACRONYMS

- **ACORDE -** *ALICE Cosmic Ray Detector*.
- **ADC** - *Analogue-to-Digital Converter*
- **ALICE** - *A Large Ion Collider Experiment*
- **ASIC** - *Application Specific Integrated Circuit*
- **ATLAS** - *A Toroidal LHC ApparatuS*
- **CDH** - *Common Data Header*
- **CERN** - *European Organization for Nuclear Research*
- **CFRP** - *Carbon-Fibre Reinforced Plastic*
- **CMS** - *Compact Muon Solenoid*
- **CTP -** *Central Trigger Processor*
- **DAQ** - *Data Acquisition*
- **DATE** - *ALICE Data Acquisition and Test Environment*
- **DCS** - *Detector Control System*
- **DDL -** *Detector Data Link*
- **DIU** - *Destination Interface Unit*
- **DLL** - *Digital Locked Loop*
- **DMA** - *Direct Memory Access*
- **DP** - *Data Packing*
- **D-RORC -** *DAQ Read-Out Receiver Cards*
- **ECS** - *Experiment Control System*
- **EMCAL** - *ElectroMagnetic CALorimeter*
- **FPGA** - *Field Programmable Gate Array*
- **FMD** - *Forward Multiplicity Detector*
- **GDC** - *Global Data Collectors*
- **GOL** - *Gigabit Optical Link*
- **HLT** - *High Level Trigger*
- **HMPID** - *High-Momentum Particle Identification Detector*
- **ITS** - *Inner Tracking System*
- **LDC** - *Local Data Concentrator*

- **LHC** - *Large Hadron Collider*
- **LHCb** - *Large Hadron Collider beauty experiment*
- **LHCf** - *Large Hadron Collider forward*
- **LTU -** *Local Trigger Unit*
- **LVDS** - *Low Voltage Differential Signaling*
- **LVPECL** - *Low Voltage Positive Emitter Coupled Logic*
- **NTD** - *Neutron Transmutation Doped*
- **OT** - *OpticalTransceiver*
- **PCB** - *Printed Circuit Board*
- **PDS** - *Permanent Data Storage*
- **PHOS** - *PHOton Spectrometer*
- **PMD** - *Photon Multiplicity Detector*
- **RR** - *Round Robin*
- **SDD** - *Silicon Drift Detector*
- **SIU** - *Source Interface Unit*
- **SPD** - *Silicon Pixel Detector*
- **SSD** - *Silicon Strip Detector*
- **TDS** - *Transient Data Storage*
- **TOF -** *High resolution array Time Of Flight*
- **TOTEM** - *Total Cross Section Elastic Scattering and Diffraction Dissociation*
- **TPC** - *Time-Projection Chamber*
- **TRD** - *High-granularity Transition-Radiation Detector*
- **TTC -** *Timing, Trigger and Control*
- **ZDC** - *Zero-Degree Calorimeters*

BIBLIOGRAPHY

[1] http://public.web.cern.ch/public/en/LHC/ALICE-en.html

[2] **http://public.web.cern.ch/public/en/LHC/LHCExperimets-en.html**

[3] ALICE Collaboration: F Carminati , P Foka , P Giubellino , A Morsch , G Paic , J-P Revol K ˇSafaˇr´ık , Y Schutz and U Awiedemann
ALICE: Physics Performance Report, Volume I
Published 19 October 2004
Online at stacks.iop.org/JPhysG/30/1517

[4] Davide Falchieri
Hardware implementation of data compression algorithms in the ALICE experiment.
PhD thesis 2000/20001
available: **http://www.bo.infn.it/~falchier/phd.html**

[5] **ALICE data acquisition**
http://ph-dep-aid.web.cern.ch/ph-dep-aid/

[6] **Xilinx, Inc. Xilinx Virtex II PRO FPGA,**
datasheet [online] available: **http://www.xilinx.com**, 2006.

[7] **Xilinx, Inc. Xilinx Spartan II FPGA,**
datasheet [online] available: **http://www.xilinx.com**, 2006.

[8] **Optoway SPS7110,1.25 Gbit/s SFP transceiver,**
datasheet [online]available: **http://www.optoway.com**, 2006.

[9] **Texas Instruments TLK1501, 0.6 to 1.5 Gbps Transceiver,**
datasheet [online] available: **http://www.ti.com**, 2006.

[10] **IDT 72T36125L44BBG 2.5 V TeraSync high speed FIFO**,
datasheet [online] available: **http://www.idt.com**, 2006.

[11] **CERN Microelectronics Group TTCrq mezzanine**,
datasheet [online] available: **http:// ttc.web.cern.ch/TTC**.

[12] **LTU Local Trigger Unit**,
datasheet [online] available: **http://www.ep.ph.bham.ac.uk/user/pedja/ALICE**, 2006.

[13] **RS232 protocol spec.**
http://en.wikipedia.org/wiki/RS-232

[14] ON Semiconductor NB100LVEP221, 2.5V/3.3V 1:20 Differential HSTL/ECL/PECL Clock Driver, datasheet [online] available: **http://www.onsemi.com**, 2006.

[15] S. Antinori, F. Costa, D. Falchieri, A. Gabrielli, E. Gandolfi, M. Masetti
Design and Test of the ALICE SDD Data Concentrator Card CARLOSrx
2006 IEEE Nuclear Science Symposium and Medical Imaging Conference , 29 October - 4 November, 2006, San Diego, CA USA.

[16] S. Antinori, Filippo Costa, D. Falchieri, E. Gandolfi, M. Masetti
CARLOSrx: an online Data Acquisition system for ALICE ITS SDD
(TWEPP2007) 12th Workshop on Electronics for LHC and future Experiments
3 - 7 September 2007, Prague, CZ

[17] Dr. Antinori Samuele, Dr. BRUNO Alessandro, Dr. BEOLè Stefania, Dr. COLI Silvia, Dr. COSTA FILIPPO, Dr. DE REMIGIS Paolo, Dr. FALCHIERI Davide, Prof. GANDOLFI Enzo, Dr. GIRAUDO Giuseppe, Dr. GIUBELLINO Paolo, Mr. KRàL Jiri, Prof. MASETTI Massimo, Dr. MAZZA Gianni, Dr. RASHEVSKY Alexandre, Dr. RICCATI Lodovico, Dr. RIVETTI Angelo, Dr. SENYUKOV Serhiy, Dr. TOSCANO Luca, Dr. TOSELLO Flavio, Dr. WHEADON Richard.
Test and commissioning of the CARLOS control boards for the ALICE Silicon Drift Detector
(TWEPP2007) 12th Workshop on Electronics for LHC and future Experiments
3 - 7 September 2007, Prague, CZ

[18] ALICE DAQ project
ALICE DAQ and ECS User's Guide
January 2006

[19] carlos v5 datashhet
http://www.bo.infn.it/~falchier/carlosv5_datasheet.pdf

[20] carlosrx datasheet
http://www.bo.infn.it/~falchier/carlosrx_v4_rel1.2_datasheet.pdf

[21] **http://ALICE-proj-ddl.web.cern.ch/ALICE-proj-ddl/ddl_intro.html**

[22] **http://ALICE-proj-ddl.web.cern.ch/ALICE-proj-ddl/rorc_intro.html**

[23] Trigger infos
http://epweb2.ph.bham.ac.uk/user/krivda/ALICE/

[24] Faninout board user manual
http://mkrivda.web.cern.ch/mkrivda/faninout.htm