UNIVERSITAT DE VALÈNCIA
(ESTUDI GENERAL)

Departament de Física Atòmica, Molecular y Nuclear

TILECAL ROD SOFTWARE LIBRARY AND RECONSTRUCTION ALGORITHMS FOR THE TMS320C6414 DSP

TREBALL D'INVESTIGACIÓ

BELÉN SALVACHÚA FERRANDO

2004

Index

Tables

Figures

Layout

This work is set on the framework of the ATLAS Hadronic Tile Calorimeter, one of the detectors which will operate at the next LHC collider at CERN. A brief introduction of CERN, the LHC accelerator and the ATLAS detector is done in **Chapter 1**.

The Tile Hadronic Calorimeter is part of the ATLAS detector. **Chapter 2** is about the calorimeter description. In this chapter we summarise the main features of the detector and we describe its optical part, front-end electronics and back-end electronics.

The Read-Out Drivers of TileCal, which belong to the back-end electronics, are described in **Chapter 3**. All the work explained in this report is mainly based on the TileCal ROD system. And the different parts of that system are described in this chapter.

Chapter 4 is about the software library developed to configure the RODs, the so-called TileRODFinal library. The TileRODFinal library was entirely programmed by the author of this report. It is a C++ based library which allows access through VME to the ROD motherboards and its components (Output Controller FPGA, Staging FPGA, Processing Units...). The library is very complete; more than 100 C++ methods were programmed. In order to do the lecture of this report more pleasant, only a general description of the library will be described in this chapter, but a detailed description of all the methods can be found in Appendix II.

As described in chapter 3, the ROD will be able to process sampled data in real time. The author studied several reconstruction algorithms which can be implemented in the Processing Units of the RODs. This work is described in **Chapter 5**. The chapter starts with a theoretical introduction of three algorithms, Flat Filtering and Fit, which are the official ones used in the beam tests of TileCal, and Optimal Filtering, which is a faster alternative to the Fit. After the theoretical introduction, the implementation of Flat Filtering and Optimal Filtering in the digital signal processors of the RODs is explained. These processors are fixed-point processors, and they are only allowed to perform multiply and accumulative operations. The chapter ends with the comparison of the online and offline reconstructions of Flat Filtering and Optimal Filtering (Energy, and Time reconstruction when possible) followed by the comparison of the calorimeter resolution when using the three algorithms with data from the beam test of TileCal for different energies and particles.

In **Chapter 6** conclusions and future plans are explained.

Appendix I shows the ROD registers addresses and a description of them.

Appendix II shows the detailed description of the C++ methods of the TileRODFinal library. The source codes for this software are not included because of space constrains.

Acronyms are in **Appendix III**.

1 INTRODUCTION

Since ever the origin of the world from the scientific point of view has been unknown. What we are and where we come from are still scientific questions without answer.

Scientists of 20 different states meet at CERN (Centre Européen pour la Recherché Nucléaire) try to give some light on some of those questions. CERN is the world's largest particle physics centre. It is placed near Geneva at the border between France and Switzerland and is currently working on the construction of the Large Hadron Collider (LHC) (Figure 1).

Figure 1: Air view of CERN and LHC.

The LHC will be the largest hadron collider in the world. In 2007 the first proton – proton (pp) collisions with a total energy in the centre of mass of 14 TeV and a full luminosity of $10^{34}cm^{-2}s^{-1}$ will start. In order to exploit the full discovery potential of the Large Hadron Collider a general-purpose pp detector named ATLAS (A Toroidal Lhc AparatuS)[1][2] is being built.

ATLAS is one of the four experiments designed for the LHC. It is 44 m long, 22 m high and weights approximately 7000 t (Figure 2). The detector is composed of several subsystems, each one with specific characteristics according to its functionality.

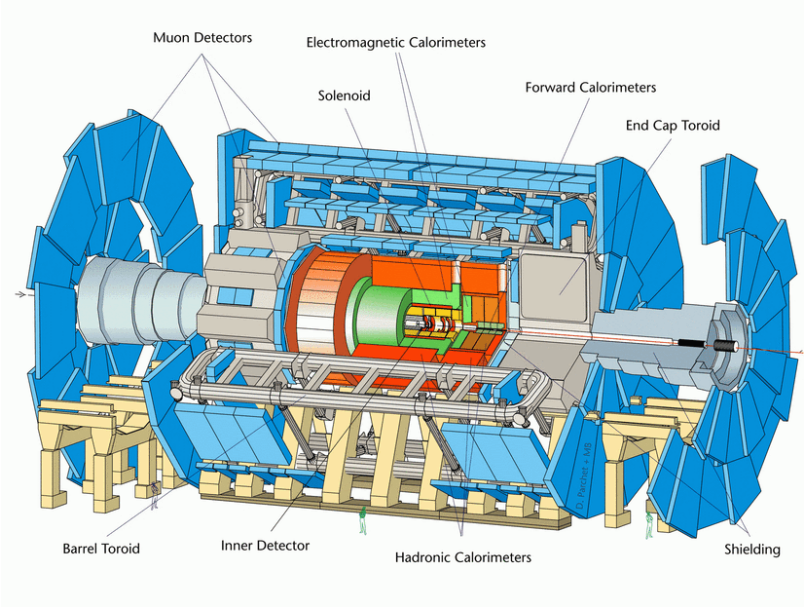

Figure 2: ATLAS detector layout.

From the inner part to the outer part of ATLAS the subsystems are given by:

- Inner detector.
- Solenoid magnets.
- Hadronic and Electromagnetic calorimeters.
- Toroid magnets.
- Muon chambers.

Table 1 shows the expected performance for each of the different ATLAS subdetectors together with its acceptance for physics measurements and trigger coverage.

SUBDETECTOR	MINIMAL REQUIRED RESOLUTION, CHARACTERISTICS	η COVERAGE	
		MESUREMENTS	TRIGGER
Inner Detector	30 % at p_T = 500 GeV Enhanced electron identification τ - and b - tagging Secondary vertex detection at initial luminosities	± 2.5 ± 2.5 ± 2.5 ± 2.5	
Electromagnetic calorimeter	$\frac{10\%}{\sqrt{E}} \oplus 0.7\%$	± 3	± 2.5
Hadronic calorimeter: • Barrel and End-cap • Forward	 $\frac{50\%}{\sqrt{E}} \oplus 3\%$ $\frac{100\%}{\sqrt{E}} \oplus 10\%$	 ± 3 3 < |η| < 5	 ± 3 3 < |η| < 5
Muon chambers	10 % at p_T = 1 TeV In stand-alone mode at highest luminosity	± 3	± 2.2

Table 1: General detector performance goals [1].

2 TILE HADRONIC CALORIMETER: TILECAL

2.1 Detector overview

The calorimetric system is designed to absorb the energy of the particles that cross the detectors. TileCal [3] is composed of 3 barrels, one central barrel with a length of 5.6 m and two extended barrels with a length of 2.9 m each. The inner radius of the detector is approximately 2.2 m and the outer radius approximately 4.2 m. Each barrel is divided azimuthally into 64 modules, see Figure 3 .

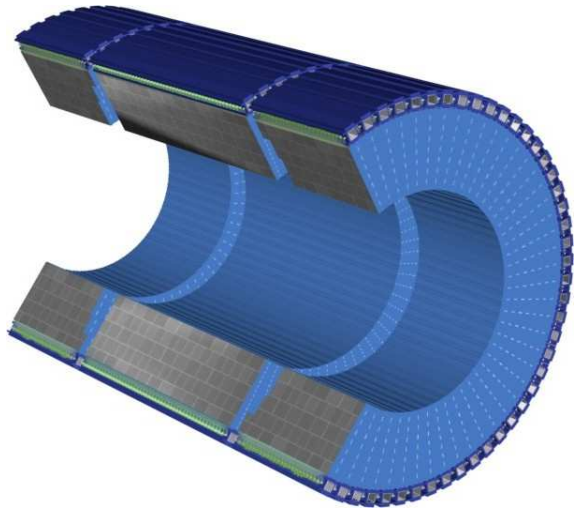

Figure 3: Tile Hadronic Calorimeter, TILECAL.

The Tile Calorimeter modules are sampling devices made out of steel and scintillating tiles, as absorber and active material respectively. The absorber structure is a laminate of steel plates of various dimensions, connected to a massive structural element referred to as a girder. The principal innovation of this detector is that the tiles are arranged perpendicularly to the beam line of the LHC. It has been shown that this characteristic provides a good homogeneity in energy resolution.

Summarising, the basic considerations of the detector design are:

- Good energy resolution over the whole η range covered.
- Good linearity in response from the few GeV to the TeV range.
- Excellent uniformity in both η and ϕ directions.
- Good hermeticity, with the presence of cracks and dead material reduced as much as possible.

2.2 Optics

Ionizing particles crossing the TileCal scintillating tiles induce the production of light in the base material of the tiles [3], with wavelengths in the Ultra Violet (UV) range and intensities

proportional to the energy deposited by the particle. This light propagates through the tile to its edges, where it is absorbed by the wave length shifting (WLS) fibres and shifted to a longer wavelength (chosen to match the sensitive region of the photomultiplier (PMT) photocathode). The fibres are coupled to the tiles in the radial direction, see Figure 4, and propagate the light via total internal reflection to the PMT where it is detected. The PMT converts the light into electric pulses that can be processed by the electronics.

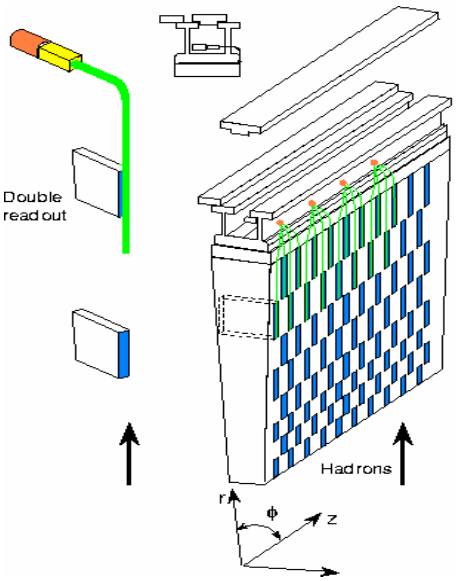

Figure 4: Tile Calorimeter sketch.

Each tile is read-out by two photomultipliers to achieve signal redundancy. Read-out cells are defined by the routing of the fibres from different tiles to one PMT. Each cell is read-out by 2 PMTs. In Figure 5 the read-out cell distribution is shown.

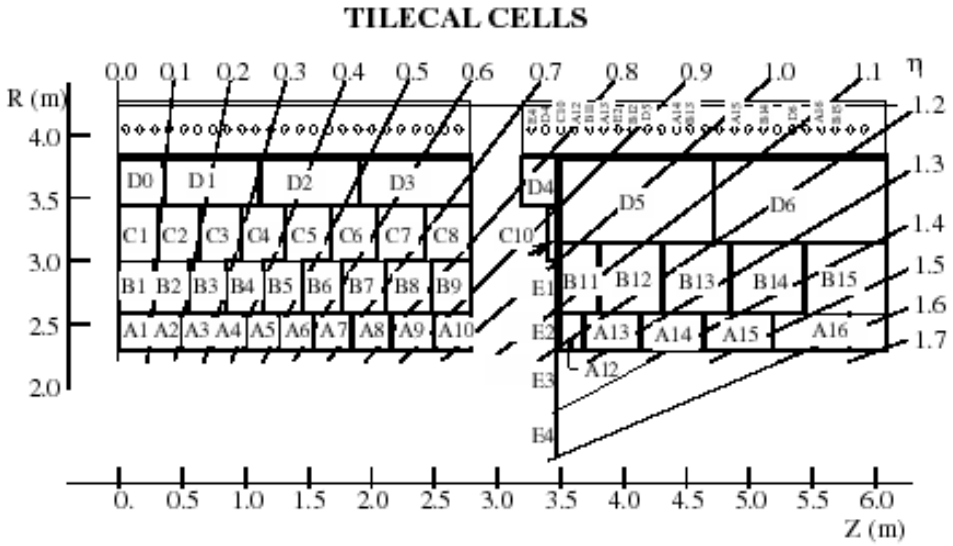

Figure 5: TileCal cell distribution for a central and an extended barrel module.

2.3 Front - End Electronics

The overall design is simple and compact; all front-end and digitizing electronics are situated in the back-beam region of the calorimeters on so-called drawers. The drawers are movable and can be inserted into a girder (Figure 6). Always two drawers are combined to a 3 m long unit, called super-drawer. The super-drawer of a barrel module contains 45 PMTs and the super-drawer of an extended barrel module 32 PMTs. The high voltages (HV) of the PMTs are regulated by special boards in the drawer. Pipelines and digitizers are locally installed. One super-drawer is needed to read-out an extended barrel module, while barrel modules are equipped with two super-drawers.

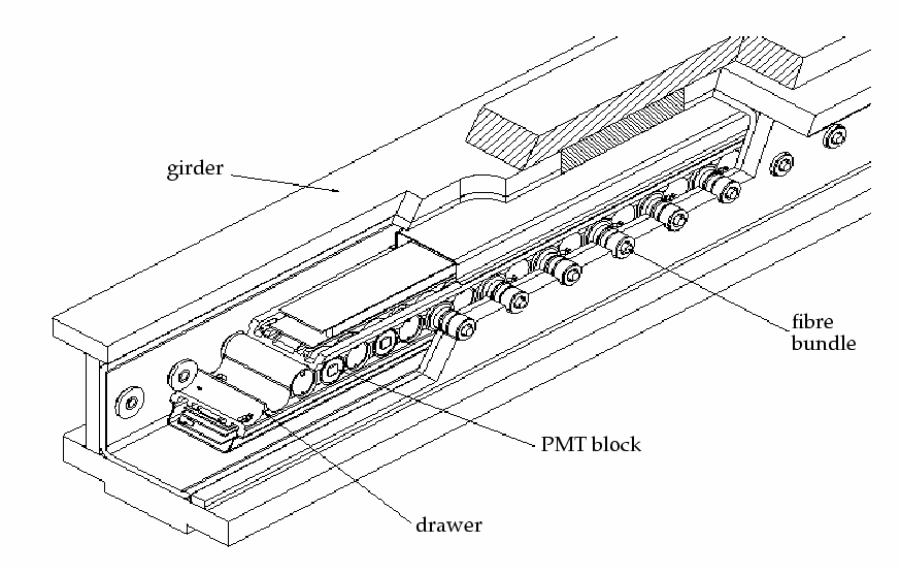

Figure 6: View of a drawer inside a girder of a module.

The electronics boards are connected to motherboards which are situated on the top and on the bottom sides of the drawers. Figure 7 shows the layout of the electronics in a drawer.

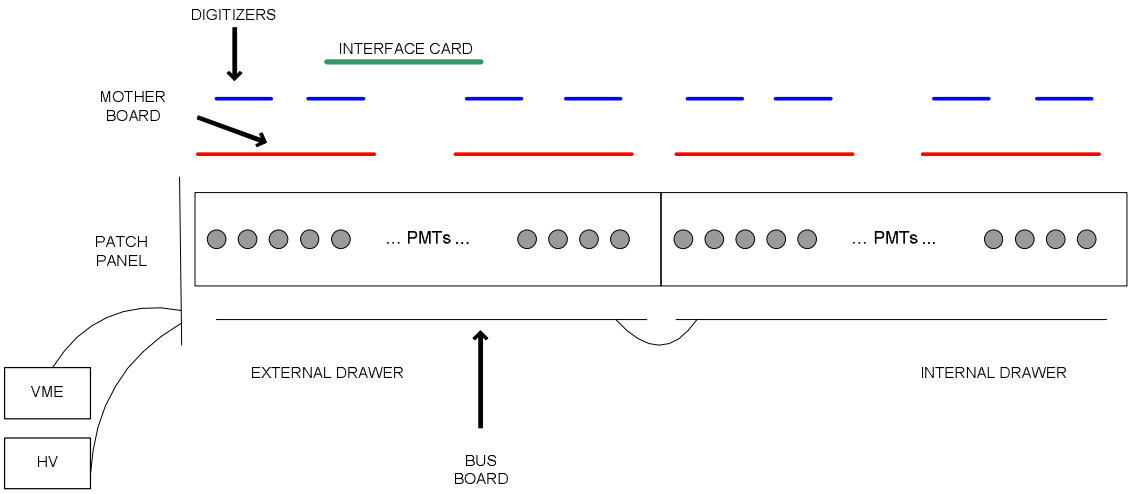

Figure 7: Layout of the electronics in a drawer.

2.3.1 PMT Blocks

The function of a PMT block is to convert the light from the scintillating tiles into electric signals. Each PMT block is an assembly of four parts:

- A light mixer.
- A photomultiplier tube.
- A HV divider.
- And a 3-in-1 card.

The PMT blocks (Figure 8) are located in holes inside the rigid aluminium structure of the drawers and are shielded with a mu-metal cylinder to provide magnetic shielding of up to 200Gauss in any direction.

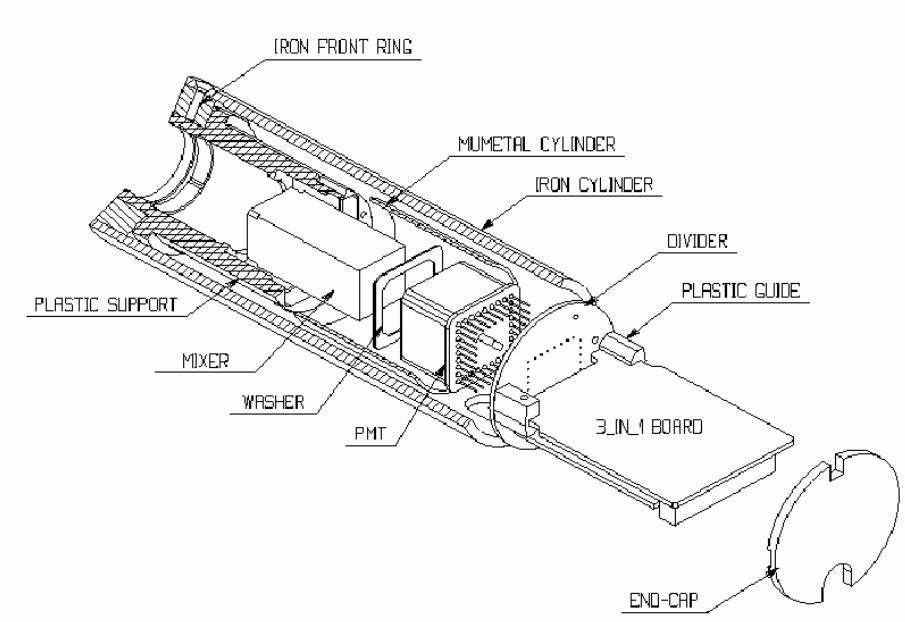

Figure 8: Arrangement of a PMT block.

Light mixer: A light mixer is placed between the fibres and the photocathode to optimize detection uniformity (Figure 9).

PMT: Several studies have been made in order to select the best photomultiplier. It was found that the Hamamatsu R7877 (Figure 10) was the best choice. A summary of the main features of the PMT can be found on Table 2 .

QUANTITY	REQUIREMENT
Quantum efficiency at 480 nm	> 18 %
Nominal gain with full efficiency	10^5
Dark current at nominal gain	< 1 nA
Rise time	< 2.5 ns
High voltage at nominal gain	< 1000 V
Gain variation of 1 % with shielding in every direction	500 Gauss
Max. non-linearity for pulses of 15 ns and 50 mA currents	< 2 %
Useful photocathode area	< 300 mm^2
Length	< 50 mm

Table 2: Main PMT requirements.

HV divider: The primary purpose of the divider (Figure 11) is to partition the high voltage between the dynodes of the PMT. The Tile Calorimeter divider also serves as a socket to allow the connection of the PMT to the front-end electronics without any interconnecting wires. This design minimizes the capacitance between the PMT and the electronics and is important to reduce noise and unreliable connections.

3-in-1 card: This card (Figure 12) has 3 functionalities:

- Provide a high and low gain shaped output pulse for the digitizer boards.
- Charge injection calibration.
- Slow integration of the PMT signals for monitoring and calibration.

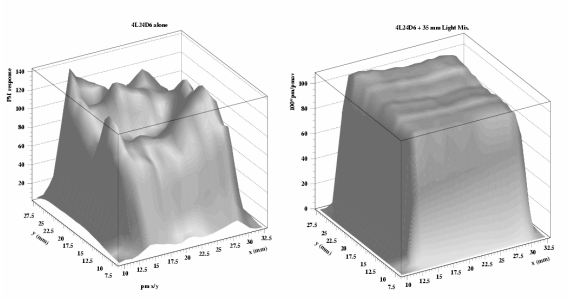

Figure 9: Response of a PMT without (left) and with (right) a light mixer as a function of the fibre position.

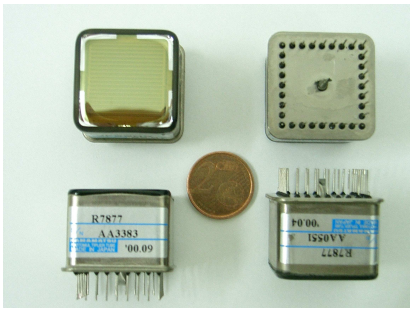

Figure 10: Photomultipier, Hamamatsu R7877.

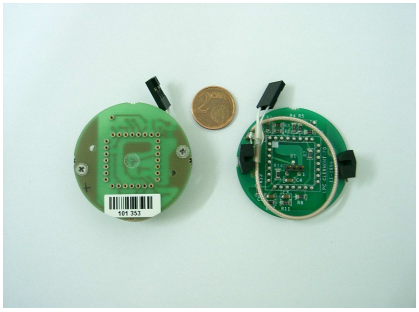

Figure 11: Picture of TileCal dividers.

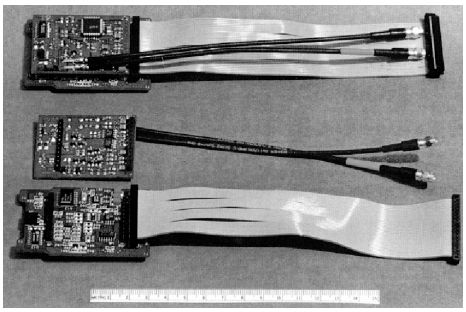

Figure 12: The 3-in-1 card.

2.3.2 Mother Boards

Signals and services are transmitted along the electronics drawer using a 3-layer structure. The inner layer carries low voltage power and digital control signals to the PMT blocks. The middle layer supports cables carrying the analogue signals from the PMT blocks to the digitizers, while the outer layer contains the level 1 (LVL1) trigger summation circuits and their output cables.

2.3.3 Digitizer System

Fast pulse signals from the 3-in-1 cards are digitized in 8 digitizer boards and sent down a digital pipeline [4]. On receipt of a LVL1 trigger, the digitizer boards capture an event frame consisting of a string of digitisations. The events (data frames) are stored locally and queued for transmission to the interface link. A general diagram for one channel is shown in Figure 13.

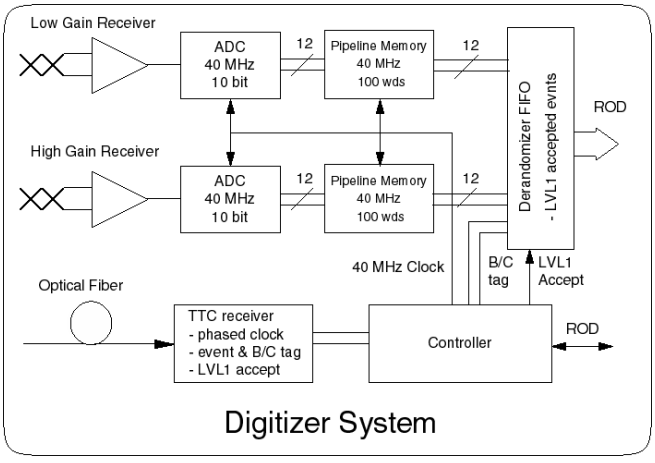

Figure 13: Digitizer system for a single channel.

2.3.4 Interface links

The interface link card receives the TTC information and sends it to the digitizer boards. When an event is accepted the 8 digitizer boards send serialized data to the interface link card where it is derandomized and sent through an optical link (HDMP-1032) to the Read-Out Drivers (RODs) (see Figure 14).

The actual design of the interface links [5] is based on the S-Link protocol using HP G-link chips as physical layer.

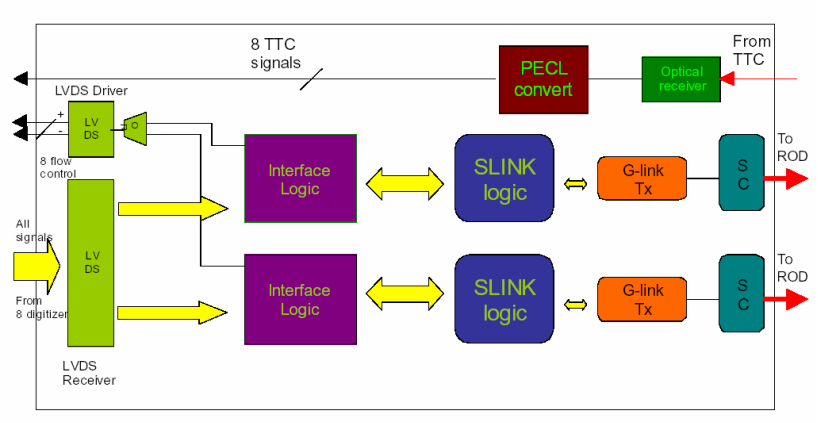

Figure 14: Block diagram of the interface board.

2.4 Back - End Electronics: ROD

The general scheme of the Read-Out electronics can be split into three parts:

Optical part: tiles, fibres and photomultipliers.

Analogue part: HV dividers and 3-in-1 cards.

Digital part: digitizers and interface boards, which are part of the Front-End electronics and the Read-Out Drivers or the so-called Back-End electronics.

From the drawer the digital information is transferred to the RODs in the counting room via optical fibres. A diagram of the read-out flow is shown in Figure 15 .

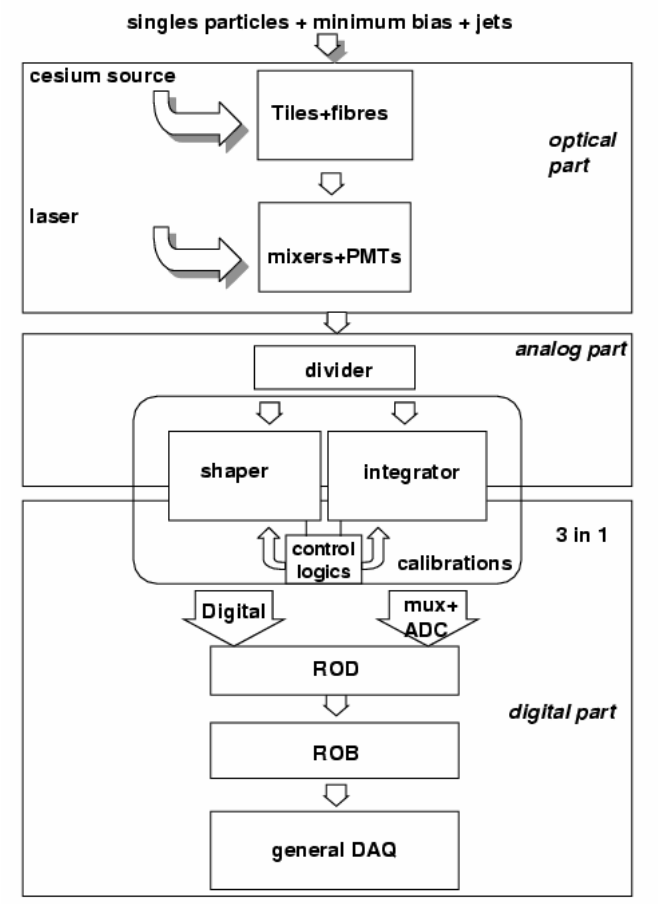

Figure 15: General scheme of the Tile Calorimeter Read-Out.

3 READ-OUT DRIVER: ROD

3.1 Overview set-up

The RODs [6] will be situated in 4 crates corresponding to the 4 read-out partitions (two for the two extended barrels and two for the central barrel). Each partition is managed by the so-called TTC crate and is equipped by standard modules developed by the RD12 [7] collaboration of the LHC experiments.

The ROD crate (Figure 16) is a standard ATLAS 9U crate. Each crate can hold a crate controller, a Trigger and Busy module, and up to 16 ROD motherboards with a Transition Module associated but at present 8 ROD motherboard are enough for full system read-out in each crate.

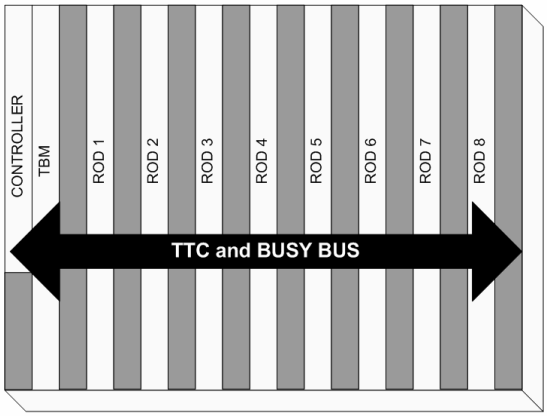

Figure 16: ROD Crate and modules.

- **Crate controller.** The Crate controller is a 6U VME module with a CPU for the initialization of the modules as well as monitoring and support of the VME access (configuration and data transfer). The VP-110 from Concurrent Technologies [8] is the standard accepted by the ATLAS collaboration but also a second crate controller was used for tests, the BIT 3 from SBS_{TM} Technologies [9].

- **Trigger and Busy Module (TBM).** The TBM is a 9U VME module which receives trigger signals from the TTC crate. The TBM distributes the trigger level 1 accept and the clock signals over a custom P3 backplane. Through this backplane the TBM also receives the busy signals and produces a logical OR of these signals to generate a "crate busy" signal.

- **ROD motherboard.** The ROD motherboard (Figure 17) is a 9U VME module that can read up to 8 optical fibres from the Front-End electronics. It has up to 4 mezzanine cards, called Processing Units, to process the data online before sending it to the transition module installed at the back of the VME crate.

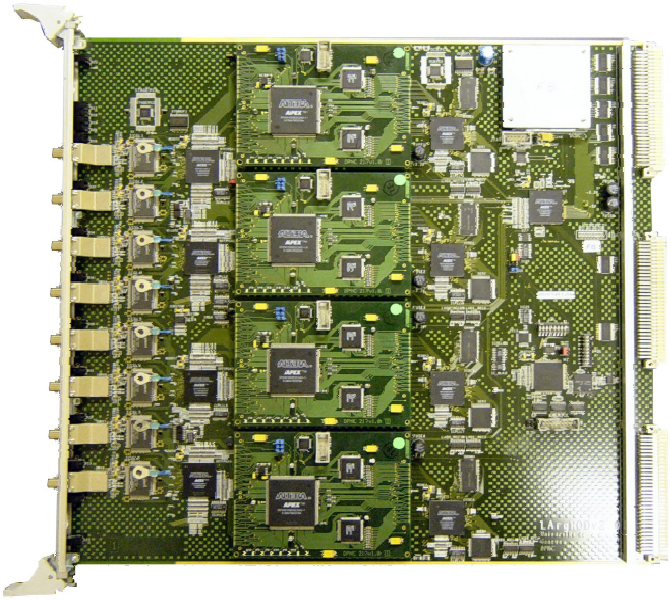

Figure 17: ROD motherboard.

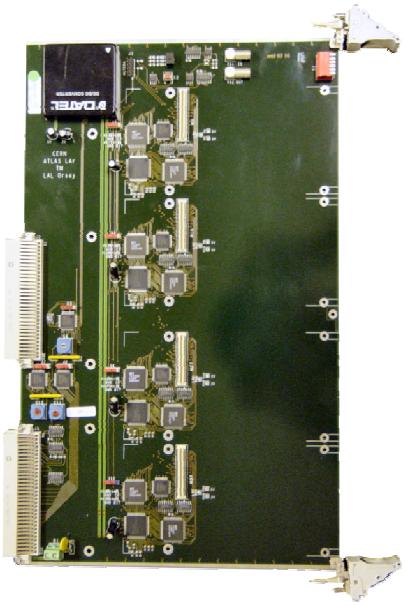

Figure 18: Transition module.

- **Transition Modules (TM).** (Figure 18) These boards are place just behind each ROD module. Each TM can hold up to 4 S-Link [10] mezzanine cards (called Link Source Cards, LSC). The ROD module sends the data through the backplane P2 and P3 to the TM which transmits it via optical fibres to the next step in the read-out chain. Serializers and deserializers are located in ROD and TM respectively to allow this functionality because of limited number of pins in P2 and custom P3.

3.2 Hardware Description

A general block diagram of the ROD motherboard can be seen in Figure 19.

3.2.1 Optical receivers and G-link chips

The ROD motherboard has up to 8 optical receivers (ORX). These ORXs are mezzanine boards that receive the optical signals coming from the Front-End electronics. Then 8 G-link chips (HDMP-1024) deserialize the incoming data and send it to the Staging FPGAs (Field Programmable Gate Arrays).

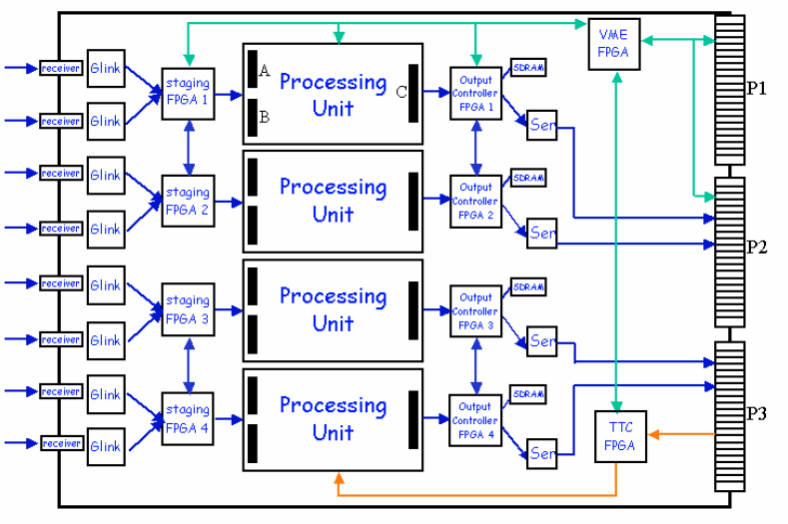

Figure 19: Layout of the ROD motherboard.

3.2.2 Staging FPGAs

Four Staging FPGAs, ACEX EP1k50, are used in the ROD motherboard. Each one receives deserialized data from two G-link chips. Three main functionalities can be identified for the Staging FPGAs:

- Route the input data to the Processing Unit.
- Monitor the G-link temperature.
- Send event data to the Processing Unit for testing.

In the design of the ROD motherboard from the Liquid Argon Collaboration a PLL, Phase-Locked Loop, (CY2308 [11]) is placed near the Staging FPGA which doubles the TTC clock frequency (from 40.08 to 80.16 MHz). For the TileCal ROD motherboard this clock was exchanged with one having the same frequency as the TTC clock (40.08 MHz).

3.2.3 Output Controller FPGAs

An ACEX EP1k100 is used as Output Controller FPGA (OC). There are 4 OC in a ROD motherboard. The OC reads from the FIFO of the Processing Units and sends the data either to a SDRAM [12] to read the data through the VME protocol or to a serializer chip to send the data to the TM.

3.2.4 VME FPGA

The VME interface is implemented in a FPGA ACEX EP1k100. The ROD module is considered as a VME slave module and can be accessed by the Crate Controller of the ROD crate. The data bus is D32 (32 bit data words) and the addressing is A32 (32 bit address words) for data transfer. A description of the software implemented for accessing the ROD module will be given in a later chapter and a table of the ROD registers can be found in Appendix I.

3.2.5 TTC Controller FPGA

Depending on the mode of operation of the ROD, the clock of the board is generated differently. The TTC FPGA is an APEX EP1k30 and gets the LHC clock from the back plane to distribute it to the ROD.

3.2.6 Processing Units

Online data processing is done by the so-called Processing Units (PUs) which are mezzanine cards that are connected to the ROD motherboard via 3 connectors. Each ROD motherboard can hold up to 4 PUs. Nowadays two kinds of PUs are available. The Field Programmable Gate Arrays (FPGA) based PU [14] and the Digital Signal Processor (DSP) based PU [13]. The FPGA PUs are mainly used for testing purposes; the design is based on a FPGA and two FIFOs. The DSP PUs, also called Final PUs, are more complex; they have three FPGAs, two DSPs and two FIFOs. Due to the DSPs these PUs can performe digital signal reconstruction. Some algorithms such as Optimal Filtering (OF) or Flat Filtering (FF) will be implemented in order to calculate energy, time and a quality factor χ^2.

3.2.6.1 FPGA PU

The FPGA PU does not apply any online reconstruction algorithm. A picture of the FPGA PU is shown in Figure 20, the core for dataflow management is implemented in the main chip (Altera APEX20kE FPGA) of the card, while two external FIFOs have the task of data buffering from the Processing Unit to the Output FPGA.

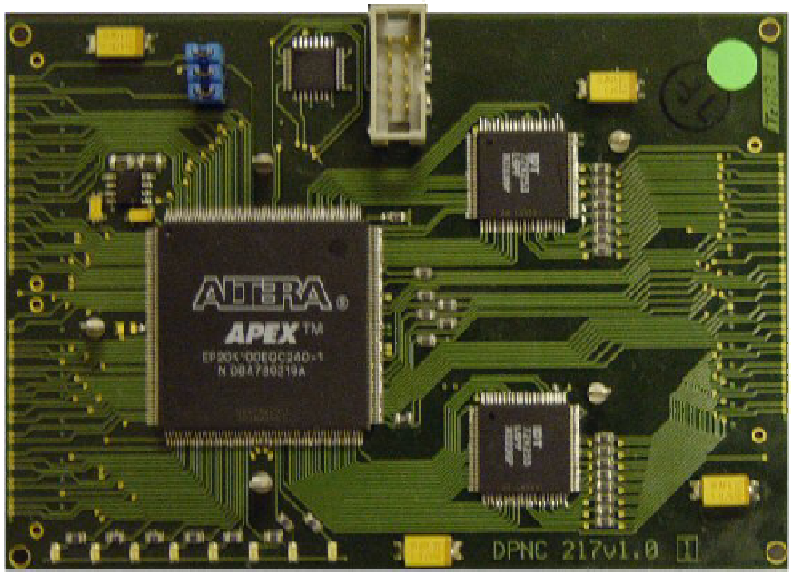

Figure 20: Picture of the DUMMY PU.

3.2.6.2 DSP PU

The DSP PU (Figure 21) is a mezzanine card with the dimensions of 120 × 85 mm. It is composed of two blocks, each one can process up to 48 read-out channels in normal mode and 96 read-out channels in staging mode (with half the number of DSP PUs in the motherboard). Each DSP block is composed of an input FPGA Cyclone EP1C6, a TMS320C6414 DSP from Texas Instruments and an external output FIFO. The DSP PU also contains an output FPGA Cyclone EP1C6 used for VME and TTC interface.

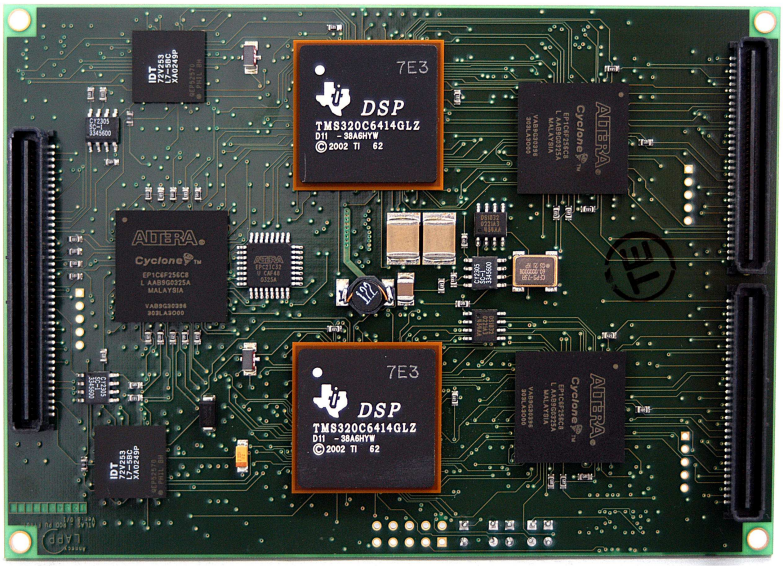

Figure 21: Picture of the DSP PU.

The input FPGAs and the DSPs can be programmed via VME. As soon as the board is powered the EEPROM uploads the code into the output FPGA which allows VME access to the whole board. Now the input FPGA code and the DSP code can be uploaded through the VME interface. In the ROD software, library methods for this booting sequence are implemented.

3.2.6.3 PU main functionalities

The main tasks of the FPGA PU are summarised below:

- Dataflow management.
- Data formatting.
- TTC reception.
- Buffering and synchronization.

The DSP PU has the same functionalities as the FPGA PU and is additionally capable of:

- Data processing with reconstruction algorithms.
- Error detection, several checks can be performed like the presence of the start and the end of an event, the parity of each word, etc.

3.3 ROD main functionalities

The ROD main functionalities [14] are given by:

Data Processing: Raw data gathering from the first level de-randomizers at the L1A event rate of 100 kHz. The ROD provides energy, timing and pile up estimation (χ^2) to the next trigger level by processing the data with the algorithms implemented in the Processing Units. Depending on the DAQ constrains (pile-up, high energy events...) there is also the possibility to send only raw data without processing.

Trigger: TTC signals will be present (latency ~ 2μs after L1A) at each module, providing ROD L1ID (Level 1 Identifier), ROD BCID (Bunch Crossing Identifier) and Ttype (Trigger type).

Error detection: The ROD checks that the owner BCID and L1ID numbers match with the numbers received from the Front-End. If a mismatch is detected, an error flag is set with some error code.

Data links: At an event rate of 100 kHz (L1A event rate) the ROD sends the data to the next step in the acquisition chain (Read-Out Buffers, ROB) using the standard ATLAS read-out links.

Busy generation: The Trigger and Busy Module provides a busy signal which stops the L1A generation.

Local monitoring: Part of data can be acquired through VME for monitoring tasks (e.g. Online Histograming).

4 TILECAL ROD MOTHERBOARD SOFTWARE LIBRARY

4.1 Introduction

The trigger and data acquisition (TDAQ) back-end hardware of the Hadronic Tile Calorimeter of ATLAS is organized in custom 9U VME boards, called RODs. There will be a total of 32 ROD boards associated to the whole readout of the detector. Each ROD board receives a total of 8 input optical links with data from the front-end electronics of the calorimeter modules. Up to 45 channels are sent per link, each channel matching a photomultiplier output signal.

The ROD system constitutes the link between the frond-end hardware of the calorimeter and the standard general data acquisition. Data is collected from the front-end system and processed inside the ROD. For this purpose each ROD can handle up to 4 PUs which contain fixed-point DSPs from Texas Instruments. Other functionalities of the ROD are to receive the TTC (Timing Trigger and Control) signals from the CTP (Central Trigger Processor), and to synchronize them with the ones coming from the front-end electronics. The ROD must read, process and format about 9856 channels at a rate of 100 KHz (Level 1 Trigger rate)

In the final configuration 4 crates are reserved for the ROD system organization, each one holding up to 8 motherboards. This organization follows the detector Regions of Interest (RoI). The calorimeter is divided in 4 RoI, 2 extended barrels and 2 central barrels. Each RoI constitutes a partition from the TDAQ view.

4.2 ROD motherboard VME address space

The VME address space for the ROD motherboard is described in Table 3 where the geographical address of the ROD modules corresponds to the slot position on the crate.

VMEADD	31	31	29	28 …… 24	23 …… 11	10	9	8	7	6 …… 2	1	0
LOCAL/BUSY				GEOGRAPH.ADD		0	0	0	0	INTERNAL ADDRESS		
BOOT				GEOGRAPH.ADD		0	0	0	1	INTERNAL ADDRESS		
TTC				GEOGRAPH.ADD		0	0	1	0	INTERNAL ADDRESS		
IRQ				GEOGRAPH.ADD		0	0	1	1	INTERNAL ADDRESS		
PU1				GEOGRAPH.ADD		0	1	0	0	INTERNAL ADDRESS		
PU2				GEOGRAPH.ADD		0	1	0	1	INTERNAL ADDRESS		
PU3				GEOGRAPH.ADD		0	1	1	0	INTERNAL ADDRESS		
PU4				GEOGRAPH.ADD		0	1	1	1	INTERNAL ADDRESS		
OC1				GEOGRAPH.ADD		1	0	0	0	INTERNAL ADDRESS		
OC2				GEOGRAPH.ADD		1	0	0	1	INTERNAL ADDRESS		
OC3				GEOGRAPH.ADD		1	0	1	0	INTERNAL ADDRESS		
OC4				GEOGRAPH.ADD		1	0	1	1	INTERNAL ADDRESS		
STAGING1				GEOGRAPH.ADD		1	1	0	0	INTERNAL ADDRESS		
STAGING2				GEOGRAPH.ADD		1	1	0	1	INTERNAL ADDRESS		
STAGING3				GEOGRAPH.ADD		1	1	1	0	INTERNAL ADDRESS		
STAGING4				GEOGRAPH.ADD		1	1	1	1	INTERNAL ADDRESS		

Table 3: VME Address Schema

4.3 Software organization

The ROD motherboard software library is part of the TiCalOnline software package. The library is written in C++ and it is organized as a standard ATLAS CMT online software package. The different versions of the software can be found from the directory TileRODFinal. The present version of the package corresponds to v3r2p1. This version uses the Data Flow version 00-08-00 and online 00-21-01. From the directory TileRODFinal the users have access to a CMT like structure of directories (Figure 22) where header, source and binary files are stored.

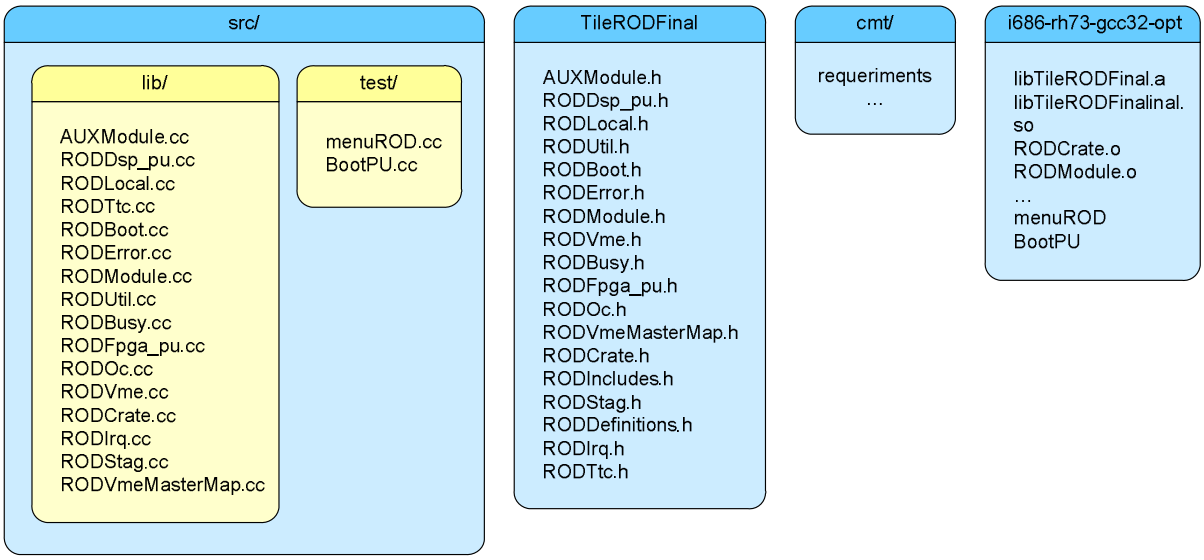

Figure 22: Directory structure of the TileRODFinal CMT package.

The software is based on a hierarchy of classes (see Figure 23). Class VME and VMEMasterMap are responsible of VME access either through the VP-110 controller from Concurrent Technologies or through the BIT-3 controller from SBS Technologies. Although the standard of ATLAS is the VP-110, we had incorporated the option of using another controller for tests on the ROD system.

CRATE class, ROD class and AUX class are the ones that describe our system, for instance, in the CRATE class it is defined the number of ROD modules in a crate and the number of Processing Units in each ROD module. Through the CRATE class a VME initialization and mapping is done. This class contains an array of pointers to the classes ROD and AUX. And it is inside the classes ROD and AUX where the users can read and write in the module registers.

The AUX class is reserved for accessing to the registers of additional modules.

Through the ROD class users can read and write in the ROD module registers. For that purpose this class has pointers to the following classes:

LOCAL class: access to the ROD motherboard LOCAL registers.

BOOT class: access to the ROD motherboard BOOT registers.

BUSY class: access to the ROD motherboard BUSY registers.

TTC class: access to the ROD motherboard TTC FPGA registers.

OC class: access to the ROD motherboard Output Controller FPGA registers.

IRQ class: access to the ROD motherboard IRQ registers.

STAGING class: access to the ROD motherboard Staging FPGA registers.

FPGA_PU class: access to the ROD Dummy PU (Processing Unit) control.

DSP_PU class: access to the ROD PU (Processing Unit) control.

Declarations of the classes and its members are done in header files, while definitions are done in source files. Each class has an associated header and a source file with the same name as class preceded by the ROD prefix.

In the next Sections a more detailed description of each class, its members and its methods is presented. Figure 23 shows the hierarchy of classes of the ROD library.

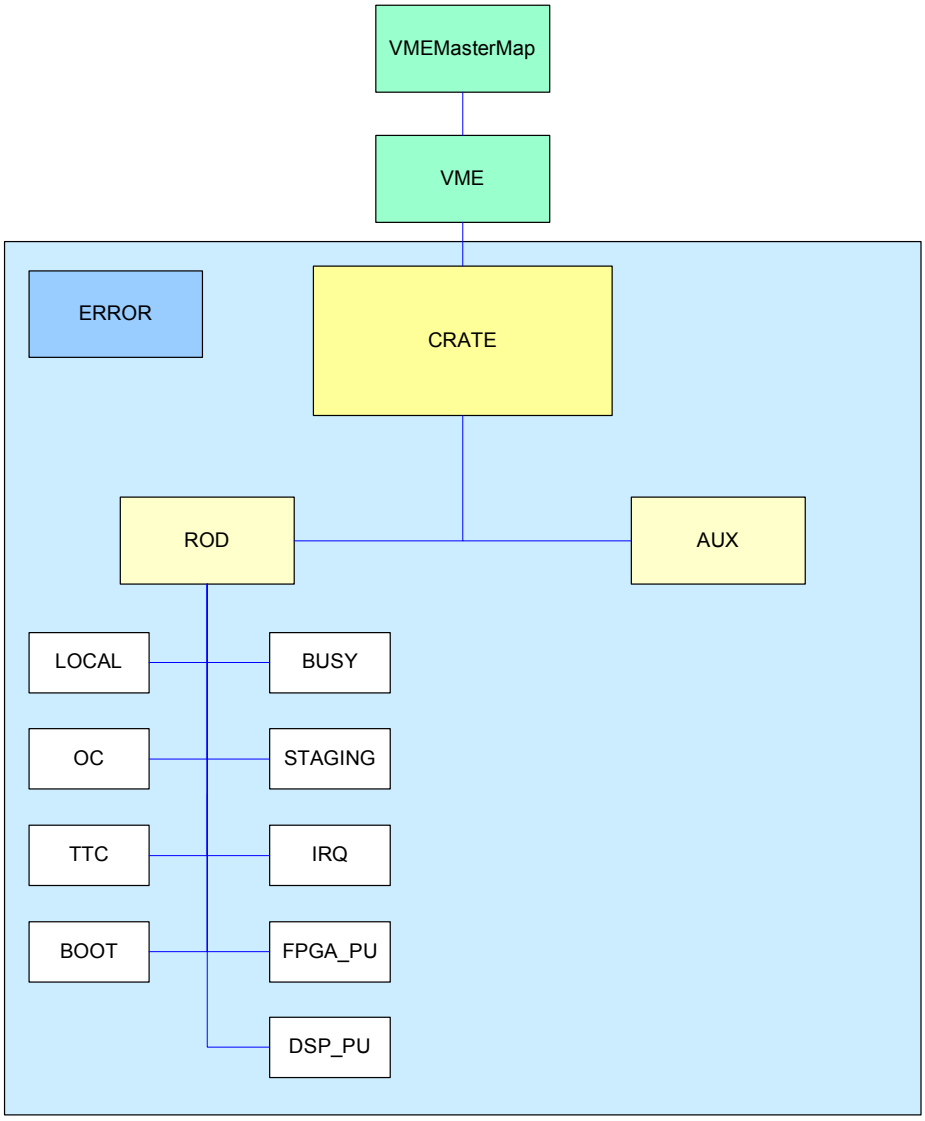

Figure 23: ROD library classes' hierarchy.

All necessary headers for the TileRODFinal library are included in a file called RODIncludes.h. This file includes also the file RODDefinitions.h where the offsets of the registers, error codes and other variables are defined. A set of important variables of the RODDefinitions.h file and their description is shown in Table 4.

NAME	VALUE	DESCRIPTION
WIN_SIZE	0x800	Size in bytes for the ROD mapping
FAST	1	Fast VME mode of reading
SAFE	0	Safe VME mode of reading
BIT3	20	Crate Controller from SBS Techonologies
VP110	30	Crate Controller from Concurrent Techonologies
MAXNOSTAG	4	Maximum number of Stagging FPGAs in a ROD module
MAXNOOCS	4	Maximum number of Output Controllers in a ROD module
MAXNOPUS	4	Maximum number of Processing Units in a ROD module
MIN_SLOT	4	First slot number that can accept a ROD or an AUX module
MAX_SLOT	20	Last slot number that can accept a ROD or an AUX module
MAXNORODS	16	Maximum number of ROD modules in a VME crate
MAXNOAUXS	5	Maximum number of AUXiliary modules in a VME crate
MAXTEMPID	7	Maximum temperature identifier.

Table 4: Description of some variables defined in RODDefinitions.h.

The error code variables will be explained in Section 4.4. A complete description of the methods inside the classes can be found in Appendix II.

4.4 Error return code

Most of the methods in the TileRODFinal library return an error corresponding either to the VME error code from the rcc_vme package, from the bit3_rcc package or from the TileRODFinal library itself. The methods to handle with these errors are in the ERROR class. In Table 5 one can see a description of the possible errors in the ROD library. Other error values are possible from the vme_rcc package and the bit3_rcc package during the access to the VME bus.

ERROR NAME	VALUE	DESCRIPTION
ROD_SUCCES	VME_SUCCES	All ok
ROD_FAIL	-1	Something fails
ROD_NO_KNOW	60	Error not known
ROD_FAIL_NORODS	61	Incorrect number of ROD modules
ROD_FAIL_NOAUXS	62	Incorrect number of auxiliary modules
ROD_FAIL_NOPUS	63	Incorrect number of processing units in a ROD module
ROD_FAIL_ALLOC	64	Failed allocating memory space
ROD_NO_MODULE	65	ROD module not found in the slot number
ROD_FAIL_OC	66	Incorrect Output Controller identifier
ROD_FAIL_STAG	67	Incorrect Staging FPGA identifier
ROD_FAIL_FPGA_PU	68	Incorrect Dummy Processing Unit
ROD_ERROR_S_F	69	Incorrect mode, nor SAFE nor FAST
ROD_STAG_TEMP	70	Incorrect temperature identifier. Up to 8 temperatures can be read, one per G-Link. From '0' to MAXTEMPID

Table 5: Error code description.

4.5 Application example

For accessing the registers of the ROD module a very simple application can be built following few steps:

1. In order to get all the definitions and includes ones should include the file RODIncludes.h. The classes are defined in a namespace, TileROD, which should be used in the application.

```
#include "TileRODFinal/RODIncludes.h"
using namespace RODTile;
```

2. Create a CRATE instance (or a pointer), and an ERROR instance (or a pointer).

```
CRATE * MyCrate = new CRATE;
ERROR * Error = new ERROR;
ROD_Error ret;
```

3. Init the CRATE object by calling the method CrateInit( ) from the CRATE class.

```
int NORODS = 1;
int NOAUXS = 0;
int DeviceName = VP110; // or BIT3
ret = MyCrate->CrateInit(NORODS,NOAUXS,DeviceName);
if (ret != ROD_SUCCESS) {
Error->RODErrorPrint(ret);
Return (ROD_FAIL);
}
```

4. In this application the user will defined only one ROD module. To access the registers of this ROD module it has to be initialized through the following code:

```
Int NOPUS = 2;    // Number of PU in a ROD module
Int SLOT = 7;     // Slot where the ROD is connected
ret = MyCrate->pRod[0]->RODInit(NOPUS, SLOT);
if (ret != ROD_SUCCESS) {
Error->RODErrorPrint(ret);
Return (ROD_FAIL);
}
```

   The index in the pRod[ ] pointer indicates that the user is accessing the first ROD module (in this case, there is only one).

5. Once the ROD module is initialized the user can access all the registers of the ROD.

```
u_int value, data;
int mode;
data = 0xfa12eb34;
mode = SAFE;                // or FAST
ret = MyCrate->pRod[0]->pStaging[3]->WriteCommande(data,mode);
if (ret != ROD_SUCCESS) {
Error->RODErrorPrint(ret);
Return (ROD_FAIL);
```

```
}
        ret = MyCrate->pRod[0]->pFpga_pu[0]->ReadStatus(&value,mode);
        if (ret != ROD_SUCCESS) {
        Error->RODErrorPrint(ret);
        Return (ROD_FAIL);
}
        ret = MyCrate->pRod[0]->pFpga_pu[0]->PrintStatus(value);
        if (ret != ROD_SUCCESS) {
        Error->RODErrorPrint(ret);
        Return (ROD_FAIL);
}
        data = 0xbabebabe;
        ret = MyCrate->pRod[0]->pTtc->WriteControl(data,mode);
        if (ret != ROD_SUCCESS) {
        Error->RODErrorPrint(ret);
        Return (ROD_FAIL);
}
```

6. At the end of the application the VME bus should be closed and all created pointers should be deleted. In order to do it we should call the CrateShutDown( ) method from the CRATE class.

```
MyCrate->CrateShutDown();
delete MyCrate;
delete Error;
```

There are many applications already built using the ROD library. The one called **menuROD** is a program that allows ROD registers access through a simple menu list.

5 RECONSTRUCTION ALGORITHMS: DSP

5.1 Introduction

The light produced in the scintillator tiles when a particle goes through the detector is converted in fast analogue pulses by the PMTs placed in the TileCal drawers. The pulses are then shaped by 7-pole shapers based on a Bessel filter which widen the signals (for simpler sampling) and provide amplitude and area proportional to the light collected by the PMTs. After the shaper, an amplification system with two gains converts the signal into two pulses (Low and High gain) with a factor of 64 between each one just before the digitalization with two 10-bits Analogical to Digital Converters (ADCs). We must emphasize the importance of the linear proportionality between the energy deposited in the calorimeter and both the amplitude and the area of the signal from its generation until its digitalization. Figure 24 shows a diagram of the read-out chain.

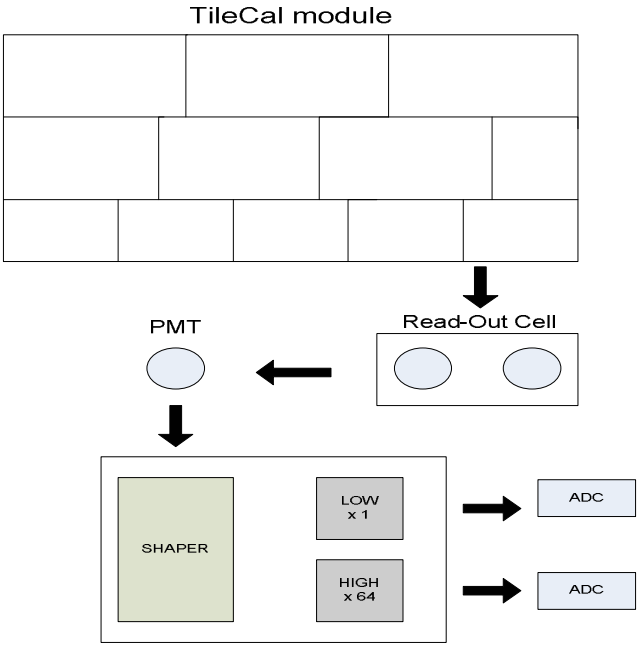

Figure 24: Diagram of the read-out chain.

Several algorithms have been studied to reconstruct the detected digital signal. In this Section we will describe a Flat Filter algorithm, an Optimal Filter algorithm and a Fit method (FIT), and we will study the implementation of some of them in a Digital Signal Processor for online reconstruction.

5.1.1 Flat Filter

The Flat Filter method is one of the simplest and fastest methods. It is based on an approximation of the pulse area which is proportional to the energy deposited in the read-out cell.

The algorithm calculates the maximum sum of five samples out of the total available samples, after pedestal subtraction. Pedestal is the signal due to noise and its calculation will be explained later.

The mathematical expression for the FF algorithm is given by:

$$A = \max\left(\sum_{i=j}^{j+4} S_i\right) \qquad j = 1,2... \ (N-4)$$

where A is the estimation of the pulse area, S_i is the sample at time t_i and N is the total number of samples.

The disadvantage of this algorithm with respect to the other algorithms is its worse energy resolution due to the imprecise estimation of the pulse area. In addition pulse time information is not provided by this method.

5.1.2 Three Parameter Fit

The three parameter Fit method is based on the minimization of the pulse shape χ^2 with respect to the variation of the amplitude, time and pedestal of the signal. Both the FF and the FIT are the official algorithms used in the beam test of TileCal for energy and time reconstruction.

The pulse shape can be described as:

$$y(t) = Ag(t) + c$$

where A is the amplitude of the signal, c is a constant pedestal and the function $g(t)$ represents the pulse shape pedestal subtracted and normalized in amplitude.

If the pulse is digitalized (in TileCal the signals are sampled every 25 ns) the above formula can be expressed as:

$$y(t_i) = Ag(t_i - \tau) + c \approx Ag(t_i) - A\tau g'(t_i) + c$$

where τ is the phase of the samples with respect to the maximum of the pulse. A graphical definition of τ is shown in Figure 25.

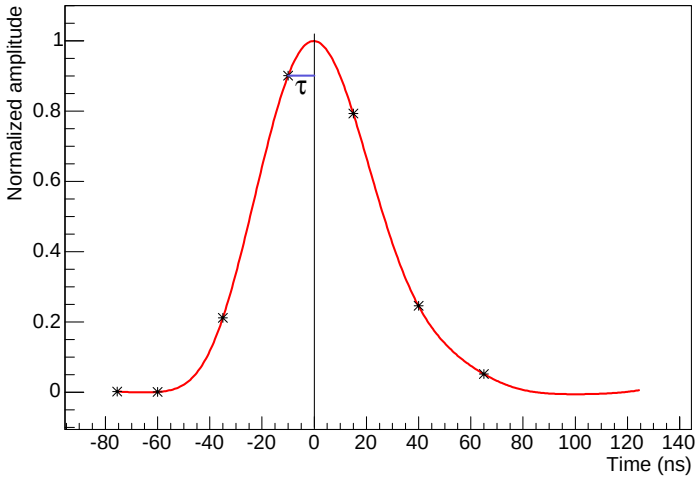

Figure 25: Sampled pulse shape and definition of τ. Seven samples are shown (starts) which are fitted to the pulse shape (red line).

The function $y(t_i)$ is the value of the pulse at time t_i, i.e., the value of the sample i, and the functions $g(t_i)$ and $g'(t_i)$ are the normalized values of the pulse and its derivative at time t_i, respectively.

The χ^2 of the fit is:

$$\chi^2 = \sum_{i=1}^{N} \frac{\{y(t_i) - (Ag(t_i) - A\tau g'(t_i) + c)\}^2}{\sigma_i^2}$$

where σ_i is the standard deviation of the noise at time t_i .

The conditions to minimize the χ^2 are:

$$\frac{\partial \chi^2}{\partial A} = 0 \qquad \frac{\partial \chi^2}{\partial (A\tau)} = 0 \qquad \frac{\partial \chi^2}{\partial c} = 0$$

Solving the system with the three last equations the solutions for the amplitude, time and pedestal are:

$$A = \frac{\left(\Delta_{yg}\Delta_{g'g'} - \Delta_{yg'}\Delta_{gg'}\right)}{\Delta_g}$$

$$A\tau = \frac{\left(\Delta_{yg}\Delta_{gg'} - \Delta_{yg'}\Delta_{gg}\right)}{\Delta_g}$$

$$c = \frac{1}{S}\left\{S_y - \frac{\left[\left(\Delta_{yg}\Delta_{g'g'} - \Delta_{yg'}\Delta_{gg'}\right)S_g + \left(\Delta_{yg}\Delta_{gg'} - \Delta_{yg'}\Delta_{gg'}\right)S_{g'}\right]}{\Delta_g}\right\}$$

where:

$$\Delta_g = \left(\Delta_{gg}\Delta_{g'g'} - \Delta_{gg'}\Delta_{gg'}\right) \qquad \Delta_{gg} = \left(S_{gg} - \frac{1}{S}S_g S_g\right) \qquad \Delta_{gg'} = \left(S_{gg'} - \frac{1}{S}S_g S_{g'}\right)$$

$$\Delta_{g'g'} = \left(S_{g'g'} - \frac{1}{S}S_{g'} S_{g'}\right) \qquad \Delta_{yg} = \left(S_{yg} - \frac{1}{S}S_y S_g\right) \qquad \Delta_{yg'} = \left(S_{yg'} - \frac{1}{S}S_y S_{g'}\right)$$

$$S_{gg} = \sum_{i=1}^{N} g(t_i)g(t_i)/\sigma_i^2 \qquad S_{gg'} = \sum_{i=1}^{N} g(t_i)g'(t_i)/\sigma_i^2 \qquad S_{g'g'} = \sum_{i=1}^{N} g'(t_i)g'(t_i)/\sigma_i^2$$

$$S_g = \sum_{i=1}^{N} g(t_i)/\sigma_i^2 \qquad S_{g'} = \sum_{i=1}^{N} g'(t_i)/\sigma_i^2 \qquad S_y = \sum_{i=1}^{N} y(t_i)/\sigma_i^2$$

$$S_{yg} = \sum_{i=1}^{N} y(t_i)g(t_i)/\sigma_i^2 \qquad S_{yg'} = \sum_{i=1}^{N} y(t_i)g'(t_i)/\sigma_i^2 \qquad S = \sum_{i=1}^{N} 1/\sigma_i^2$$

The above mathematical development shows the difficulty to implement the Fit method in the ROD online DSPs. Nevertheless, the use of Multiply-Accumulative (MAC) operations, very common in these devices, would allow its implementation in the DSPs after the modification of the algorithm to minimize the mathematical operations.

5.1.3 Optimal Filter

The Optimal Filter algorithm reconstructs the amplitude and time of the sampled signal through linear combinations using weights previously calculated to minimize the noise contribution.

The noise to be minimized in this case is the electronic noise, which appears during the beam tests of TileCal. But for the LHC a deep study of other noise sources as pile-up and minimum bias would also be needed.

The weights for Optimal Filter are calculated from the reconstructed pulse shape, the derivative of the pulse shape and noise [15] events. The pulse shape was reconstructed from dedicated charge injection runs. The calibration system of TileCal is composed of three parts, the Cesium system, the Laser system and the Charge injection system (CIS)[16]. The CIS allows the reconstruction of the pulse shape by sweeping the injection time of a fixed charge. This pulse is then fitted to a theoretical function given by:

$$g(t) = A\left(\frac{t-\tau}{\lambda}\right)^{\mu} \exp\left[-\mu\left(\frac{t-\tau}{\lambda}\right)\right] + c$$

where A, μ, τ, λ and c are parameters of the function.

The derivative $g'(t)$ is given by:

$$g'(t) = \frac{A\mu}{\lambda}\left(\frac{t-\tau}{\lambda}\right)^{\mu-1} \exp\left[-\mu\left(\frac{t-\tau}{\lambda}\right)\right] + A\left(\frac{t-\tau}{\lambda}\right)^{\mu} \exp\left[-\mu\left(\frac{t-\tau}{\lambda}\right)\right] - \frac{\mu}{\lambda}$$

and the sampled signal can now be described as:

$$y(t_i) = Ag(t_i - \tau) \approx Ag(t_i) - A\tau g'(t_i)$$

where A is the amplitude, $y(t_i)$ represents the sample at time t_i, $g(t_i)$ and $g'(t_i)$ are the normalized pulse shape in amplitude and its derivative, respectively, and τ is the phase of the sample with respect to the maximum (see Figure 25).

The signal amplitude and the phase are given by:

$$A = \sum_{i=1}^{N} a_i S_i \qquad\qquad A\tau = \sum_{i=1}^{N} b_i S_i$$

where a_i and b_i are the coefficients for the amplitude and phase, respectively, S_i is the sample at time t_i and N is the total number of samples. Note that in the mathematical description of the pulse, the pedestal is not considered as the samples S_i are pedestal subtracted. In Section 5.2 we will show how this pedestal can be estimated.

The advantage of this method is that it is very simple to apply to the DSPs once the weights have been calculated. The calculation of the weights can be done offline and then uploaded to the DSP through VME when needed. Preliminary studies of the resolution of the TileCal calorimeter comparing the Flat Filter and the Optimal Filter results can be found elsewhere [17].

5.2 Implementation of the algorithms

Starting from an initial bunch-crossing rate of 40 MHz (LHC bunch-crossing rate) the final rate of the selected events should be reduced to ~100 Hz for permanent storage. To reduce this rate the ATLAS TDAQ system is based on three levels of online event selection which are:

- Level 1 Trigger up to ~100 kHz.
- Level 2 Trigger up to ~1 kHz.
- Event Filter up to ~100 Hz.

The Read-Out Drivers are located between the first and the second level trigger. They should be able to read and compute in real time about 10000 channels of sampled data in less than 10μs, corresponding to the ATLAS Level 1 Trigger latency.

The use of processors rated by operating systems which do not operate in real-time (as Linux) is not considered as they do not provide a deterministic estimation of the latency timing for the processing.

The following results have been obtained simulation the behaviour of the TMS320C6414 DSP of Texas Instruments.

5.2.1 Digital Signal Processor: TMS320C6414-7E3 DSP

The TMS320C6414x™ DSP is a fixed-point processor from the C6000™ DSP platform of Texas Instruments [18]. The TMS320x family has an architecture designed specifically for real-time signal processing.

The TMS320C64x™ use the VelociTI™ architecture, a high performance and advanced very long instruction word (VLIW) architecture, making these DSPs excellent choices for multichannel and multifunction applications. A traditional VLIW architecture consists of multiple execution units running in parallel, performing multiple instructions during a single cycle clock.

The TMS320C64x™ DSPs execute up to eight 32-bit instructions per cycle. The central processing unit (CPU) core consists of 64 general-purpose 32-bit registers and 8 functional units which allow to execute up to 6000 millions of instructions per second (MIPS).

The main features of theses devices are summarised as following:

- Eight functional units:
  - Two multipliers.
  - Six arithmetic and logical units (ALUs).
- 8/16/32-bit data support.
- 40-bit arithmetic options for extra precision.
- Clock cycle of 720 MHz.
- Data memory cache of 128 kb.
- Program memory cache of 128 kb.
- Random access memory (RAM) of 8 Mb.
- Real-time processing.

Note that this device is only able to operate with MAC instructions as its CPU contains multiplier and ALUs units. Divisions are not allowed in the TMS320C64x™ DSP family. To fix this problem, shift (SSH) instructions were implemented to divide in powers of 2, and the use of look-up tables is mandatory to implement more complex operations.

The operations used were chosen to save time and memory, so data should be stored with the minimum number of bits per word.

5.2.2 Flat Filter

The ADCs used in TileCal provide integer samples of 10 bits spaced in time by 25 ns. Due to the simplicity of the Flat Filter algorithm the instructions which have been used at the DSP level are:

- Sums of two 16-bit integer words with a result of a 16-bit integer word.
- Right shift instructions to divide in powers of 2 (data >> number of shifts).
- Left shift instructions to multiply in powers of 2 (data << number of shifts).

Pedestals are calculated as shown in Table 6 both for the offline $\left(c_{off}\right)$ and online DSP $\left(c_{dsp}^{16-bit}\right)$ implementations.

For the offline reconstruction, the samples are given by floating point numbers $\left(S_{off,i} \quad i=1,\ldots,N\right)$ with standard divisions resulting also in floating point numbers. For the online reconstruction, pedestals and samples $\left(S_{dsp,i}^{16-bit} \quad i=1,\ldots,N\right)$ are given by integer numbers of 16 bits maximum and instead of dividing by two a rounding and a shift right operation has to be used.

OFFLINE PEDESTAL	ONLINE PEDESTAL
$c_{off} = \frac{S_{off,1} + S_{off,N}}{2}$	$c_{dsp}^{16-bit} = \left[\left(S_{dsp,1}^{16-bit} + S_{dsp,N}^{16-bit} + 2^0\right) >> 1\right]$

Table 6: FF offline and online DSP pedestal reconstruction.

Note that when the sum of the first and the last sample is an odd number, online and offline pedestals are different due to the shift operation and the rounding. This difference propagates when calculating the energy and is multiplied by a factor of 5 because of the sum of five samples (pedestal subtracted). To increase the precision on the pedestal calculation at the DSPs, the samples are thus multiplied by a factor of 2:

$$S_{dsp}^{16-bit} = 2 \cdot S_{off}^{16-bit} = \left(S_{off}^{16-bit} << 1\right)$$

In this way the sum of the first and last samples is always even and pedestals are well calculated.

The reconstruction of the energy is given by the formulas shown in Table 7, where the index *j* runs from *1* to *N-4*, with *N* the total number of samples.

OFFLINE ENERGY	ONLINE ENERGY
$A_{off} = \max\left(\sum_{i=j}^{j+4}\left(S_{off,i} - c_{off}\right)\right)$	$A_{dsp}^{16-bit} = \max\left(\sum_{i=j}^{j+4}\left(S_{dsp,i}^{16-bit} - c_{dsp}^{16-bit}\right)\right)$

Table 7: FF offline and online DSP energy reconstruction.

Note that after the reconstruction the energy should be rounded and right shifted to be in the same units as the offline energy.

$$A_{dsp}^{16-bit} = \left(A_{dsp}^{16-bit} + 2^0\right) >> 1$$

This induces differences between the offline and the online energy reconstruction which will be evaluated in Section 5.3.1.1.

5.2.3 Optimal Filter

5.2.3.1 Energy

For online OF reconstruction the operations used at the DSP are:

- Sums of two 16-bit integers with a result of a 16-bit integer.
- Sums of two 32-bit integers with a result of a 64-bit integer.
- Multiplications of one 16-bit integer with a 32-bit integer with a result of a 64-bit integer.
- Right shift instructions to divide in powers of 2 (data >> number of shifts).
- Left shift instructions to multiply in powers of 2 (data << number of shifts).

The difference with respect to the Flat Filter algorithm is that here we have to multiply the samples (pedestal subtracted) by the energy weights, and then sum the samples. Two new operations are thus needed for the OF implementation.

Pedestals are calculated in the same way as FF (Table 8).

OFFLINE PEDESTAL	ONLINE PEDESTAL
$c_{off} = \frac{S_{off,1} + S_{off,N}}{2}$	$c_{dsp}^{16-bit} = \left[\left(S_{dsp,1}^{16-bit} + S_{dsp,N}^{16-bit} + 2^0\right) >> 1\right]$

Table 8: OF offline and online pedestal reconstruction.

Note that samples are also here scaled by a factor of 2 to improve the pedestal estimation.

$$S_{dsp}^{16-bit} = 2 \cdot S_{off}^{16-bit} = \left(S_{off}^{16-bit} << 1\right)$$

Table 9 shows the formulas used for the energy reconstruction for both the offline and the DSP algorithms. The weights are given by:

- Offline: $a_{off,i}$, $b_{off,i}$.
- Online: $a_{dsp,i}^{32-bi}$, $b_{dsp,i}^{32-bit}$.

OFFLINE ENERGY	ONLINE ENERGY
$A_{off} = \sum_{i=1}^{N} \left[\left(S_{off,i} - c_{off} \right) \cdot a_{off,i} \right]$	$A_{dsp}^{64-bit} = \sum_{i=1}^{N} \left[\left(S_{dsp,i}^{16-bit} - c_{dsp}^{16-bit} \right) \cdot a_{dsp,i}^{32-bit} \right]$

Table 9: OF offline and online energy reconstruction.

A study of the energy weights shows that they are within the range $(-1,+1)$, as seen in Figure 26. Those weights were calculated for 9 samples and phases within the range $[-100,+100]$ *ns*. The weights values are distributed between ~0 and ~0.7. The peak around 0 corresponds to the weights for the first two samples and the last two samples, which are pedestal samples and its contribution is small. The peak around 0.7 corresponds to the weights for the maximum samples with larger contributions.

To express the weights as 32-bit integer numbers at the DSP we have to scale them by a normalization factor 2^X. While a fixed-point representation places a radix point somewhere in the middle of the digits (equivalent to use integer numbers which represent portions of some unit), a floating-point scheme basically represents real numbers as a base number and an exponent.

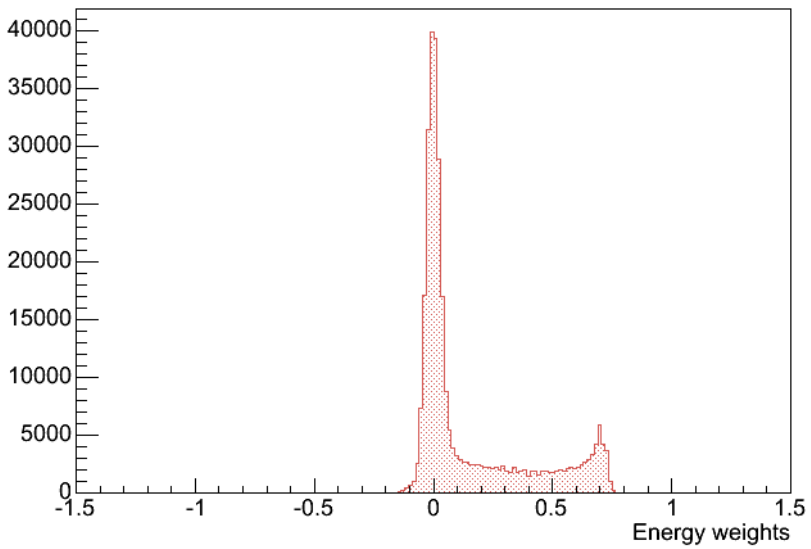

Figure 26: Energy weights distribution $a_{off,i}$.

The chosen normalization factor for these weights was 2^{19}. This value comes from requiring the quantity:

$$\left(S_{dsp,i}^{16-bit} - c_{dsp}^{16-bit}\right) \cdot a_{dsp,i}^{32-bit}$$

to be a 32-bit integer number. Indeed, as the samples $S_{dsp,i}^{16-bit}$ are given by 10-bit ADC and then scaled by *2*, this makes a total of 11 bits. This leaves 20 bits for the energy weights and 1 bit for the sign.

Scaled weights are then uploaded to the DSP as:

$$a_{dsp}^{32-bit} = 2^{19} \cdot a_{off}$$

Note that with these scale factors the reconstructed energy in the DSP is not comparable to the offline reconstructed energy. The energy should thus be rounded and renormalized to the same scale as the offline energy:

$$A_{dsp}^{64-bit} \approx A_{off} \cdot 2^{(19+1)} \quad \rightarrow \quad A_{dsp}^{32-bit} = \left[\left(A_{dsp}^{64-bit} + 2^{19}\right) >> 20\right] \approx A_{off}$$

5.2.3.2 Time

Table 10 shows the instructions to calculate the time for both the offline, $(\tau A)_{off}$, and the online DSP, $(\tau A)_{dsp}^{64-bit}$, algorithms.

OFFLINE TIME	ONLINE TIME
$(\tau A)_{off} = \sum_{i=1}^{N}\left[\left(S_{off,i} - c_{off}\right) \cdot b_{off,i}\right]$	$(\tau A)_{dsp}^{64-bit} = \sum_{i=1}^{N}\left[\left(S_{dsp,i}^{16-bit} - c_{dsp}^{16-bit}\right) \cdot b_{dsp,i}^{32-bit}\right]$

Table 10: OF offline and online time reconstruction.

The procedure to estimate the time is similar to the one used for the energy reconstruction. The difference is in the scale factor for the time weights. For phases within the range $[-100,+100]$ *ns*, time weights are given between $(-20,+20)$, see Figure 27. We choose thus normalization scale factor of 2^{14}.

$$b_{dsp}^{32-bit} = 2^{14} \cdot b_{off}$$

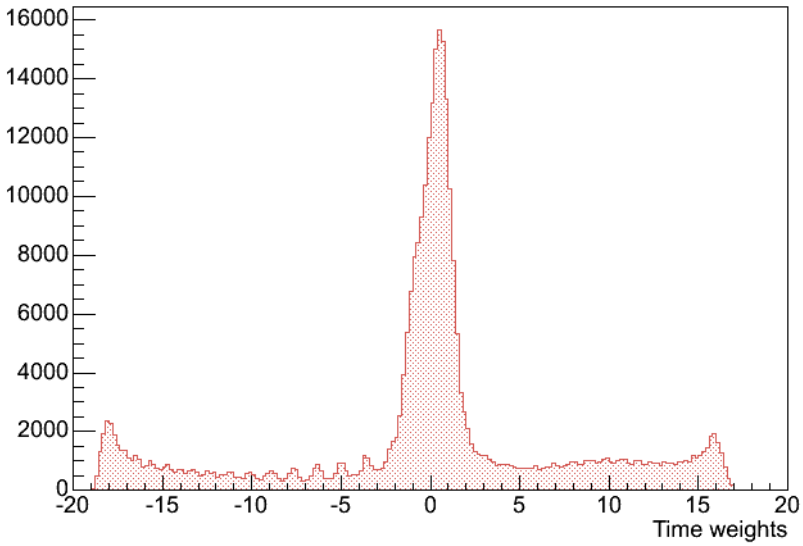

Figure 27: Time weights distribution.

Nevertheless note now that the result of the algorithm is a multiplication of time by amplitude. To get the time we should thus divide by the previously calculated amplitude. As the DSP can not performed divisions this operation is implemented as a look-up table.

A division by A_{dsp} is the same as a multiplication by $1/A_{dsp}$. We use, thus, a table with two columns, the first one represents amplitude values from *1* to *1200* in steps of *1*, while the second one shows the result of $1/A_{dsp}$ scaled by 2^{15} in order to express this quantity as a 16-bit integer word.

Once the amplitude is reconstructed we look-up in the table its corresponding $1/A_{dsp}$ value to reconstruct the time. We make sure that the value $(\tau A)_{dsp}^{64-bit}$ can be fitted in a 32-bit word and eventually normalized to the rest of applied scale factors (samples, time weights and $1/A_{dsp}$).

$$\tau_{dsp}^{16-bit} = \left\{ \left[(\tau A)_{dsp}^{32-bit} \cdot (1/A)_{dsp}^{16-bit} + 2^{(1+14+15-1)} \right] >> (1+14+15) \right\}$$

5.3 Results on Energy

For a proper comparison of the offline and online DSP pulse reconstruction a study of a single TileCal channel was done for both algorithms FF and OF. Then a study on the total energy resolution comparing the three algorithms, FF (offline and DSP), OF (offline and DSP) and FIT (offline) for different energies and particles was also performed.

5.3.1 Single channel results

The data used in this analysis corresponds to the beam test period of August 2003 of TileCal. The beam was mainly composed of electrons at *180 GeV* coming into cell A-2 of a negative central barrel of TileCal. The hit angle of the beam was *20º* (Figure 5). The PMT chosen for this analysis was number 6, which belongs to cell A-2.

5.3.1.1 Flat Filter

Figure 28 shows the reconstructed energy from the offline and online (as in the DSP) implementations for both low and high gain events. Both reconstructions are superimposed in the same plot. The X axis shows the energy in ADC counts, i.e., just the maximum sum of five samples out of the total number of samples.

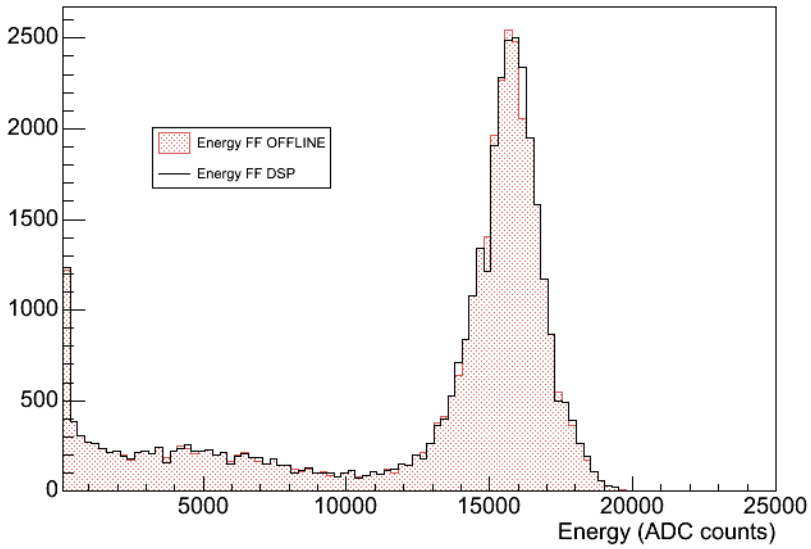

Figure 28: FF reconstruction for PMT 6 of a negative barrel module of TileCal.

The peak around *16000 ADC counts* corresponds to the energy deposited mainly by electrons which stop in the first cell of the calorimeter. The tail at lower energies corresponds to pions which deposit its energy between the first and the second cells of the module. Some small differences can be observed from the offline and the DSP reconstruction (see Figure 29). These differences are due to the last normalization in the calculation of the energy. The energy in the DSP is calculated as an integer, while the offline energy is given by a decimal number. When the sum over the five DSP samples is an odd number the rounding and the normalization induce thus this difference:

- For high gain events, if the sum over the five DSP samples is odd the result in the DSP is *0.5* bigger than the offline result.

- For low gain events, the difference is also *0.5* but the result should be multiplied by *64* if we want to compare events with both high and low gains. Then the DSP energy is 0.5×64 which is *32 ADC counts* larger than the offline energy.

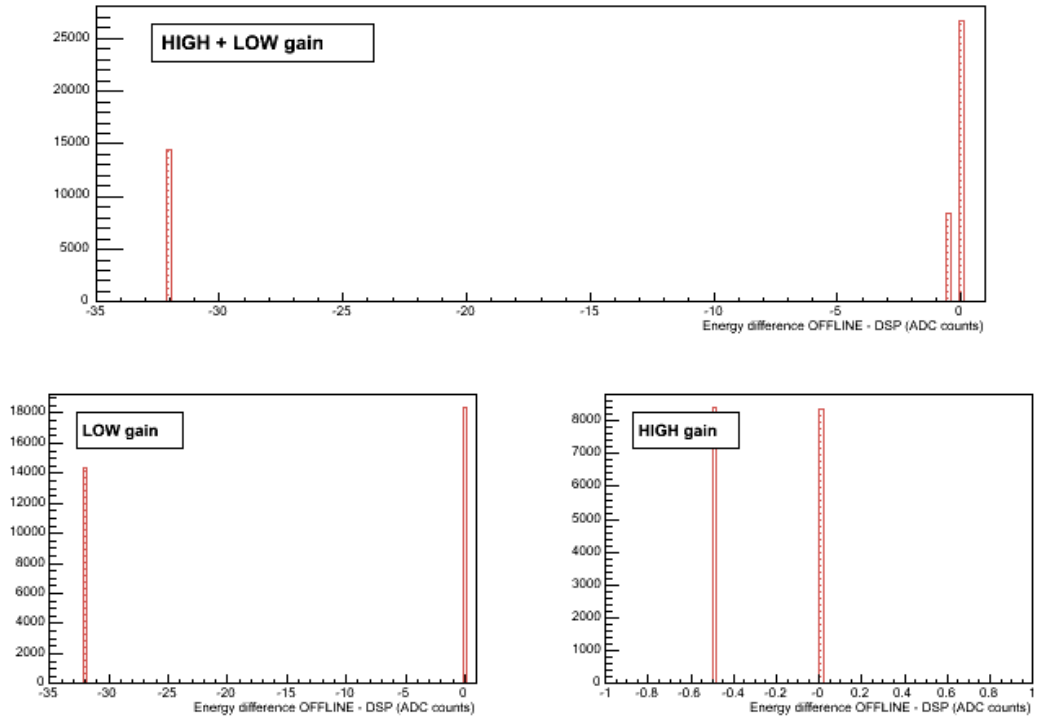

Figure 29: *Top:* Difference between FF offline and DSP energy reconstruction for all events. *Bottom:* The same as above for low gain events (left) and high gain events (right).

Figure 30 shows the relative energy difference between the offline and DSP reconstructions as a function of energy. The two lines correspond to the two different gains. The differences are less than 3% for high gain events and less than *1.5%* for low gain events.

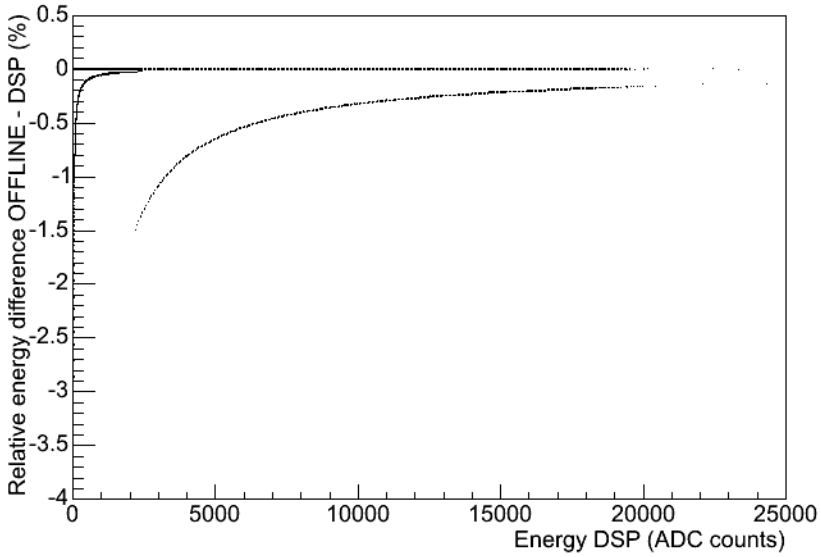

Figure 30: Relative difference (in %) of FF energy versus energy in the DSP (in ADC counts).

5.3.1.2 Optimal Filter

During the beam test of TileCal an asynchronous beam of particles comes into the calorimeter modules. Particles arrive with a non defined phase which, only in few cases, is comparable with the sampling time (25 ns). The OF weights are calculated for a fixed phase, thus if samples arrive at different phases the achieved reconstruction will not be optimal. At LHC the beam is nevertheless synchronous and all samples arrive with a fixed phase. In order to overcome this difficulty an algorithm with iterations was thought and implemented at the beam test. A first reconstruction is calculated with weights for phase equal to *0 ns* (OF without iterations). The results of this reconstruction are just a first estimation of the amplitude and the phase. As the phase is now known, one iteration can be implemented for weights calculated for this phase (Optimal Filter with 1 iteration).

5.3.1.2.1 OF without iterations

Figure 31 shows the reconstructed energy with Optimal Filtering without iterations. The weights used for this reconstruction were those calculated for phase equal to *0 ns*. The X axis shows the energy in ADC counts for PMT 6 of a negative barrel module of TileCal.

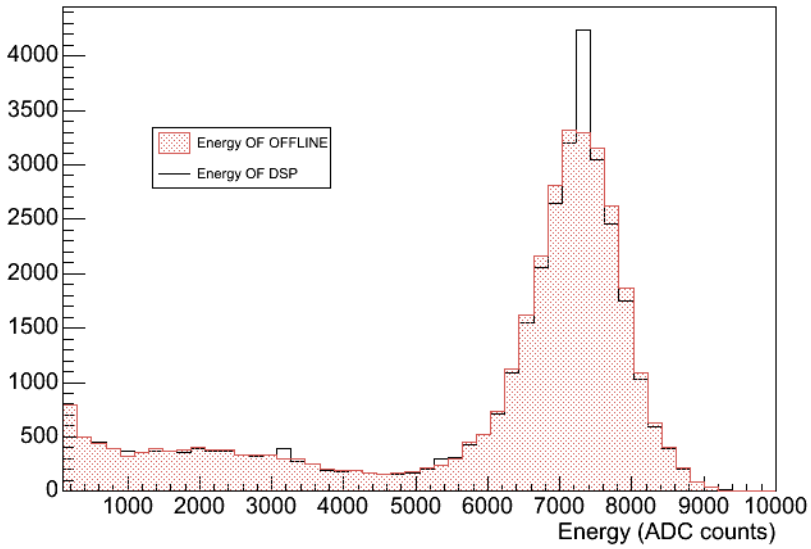

Figure 31: OF reconstruction without iterations for PMT 6 of a negative barrel module of TileCal

The difference between the offline reconstructed amplitude and the DSP amplitude is shown in Figure 32. The top plot shows the energy difference for both high gain events and low gain events. The two contributions (low and high gain) are again evident. The plots at the bottom show the same distributions separated for low (left) and high (right) gain events.

We note here differences which are uniformly distributed and explained by:

- For high gain events the limits of the distribution are between *-0.5* and *0.5 ADC counts*. These differences are due to rounding the result to an integer number in the case of the DSP.

- For low gain we get the same behaviour as before, but taking into account that the amplitudes are multiplied by a factor *64* so the limits of the distributions are between *-32* and *32 ADC counts*.

Figure 33 shows the relative difference (in %) between the offline reconstructed amplitude and the DSP amplitude. The contribution of the two gains is also here evident. At low energies the high gain is used, and the relative error does not exceed the *8%* or *10%*. For low gain events the relative error is less than *4%*. The vertical lines are explained as the amplitude in the DSP is always an integer number.

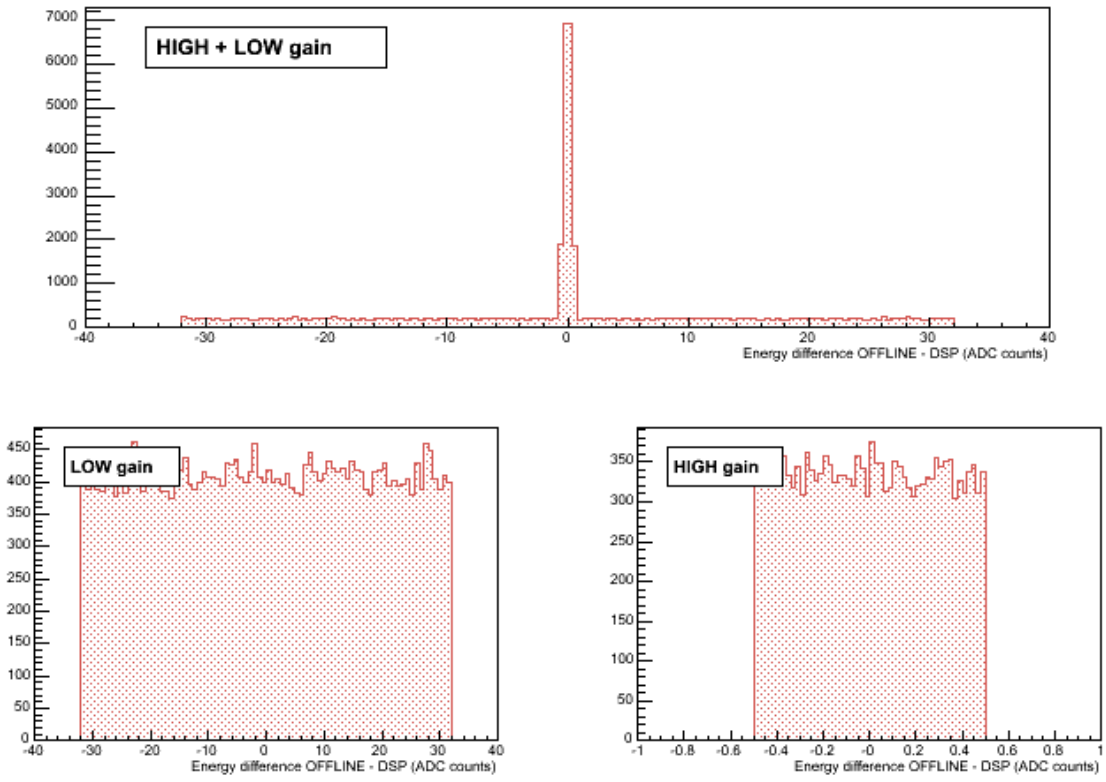

Figure 32: *Top:* Difference between OF offline energy reconstruction and OF DSP energy reconstruction for all events. *Bottom:* The same as above for low gain events (left) and high gain events (right).

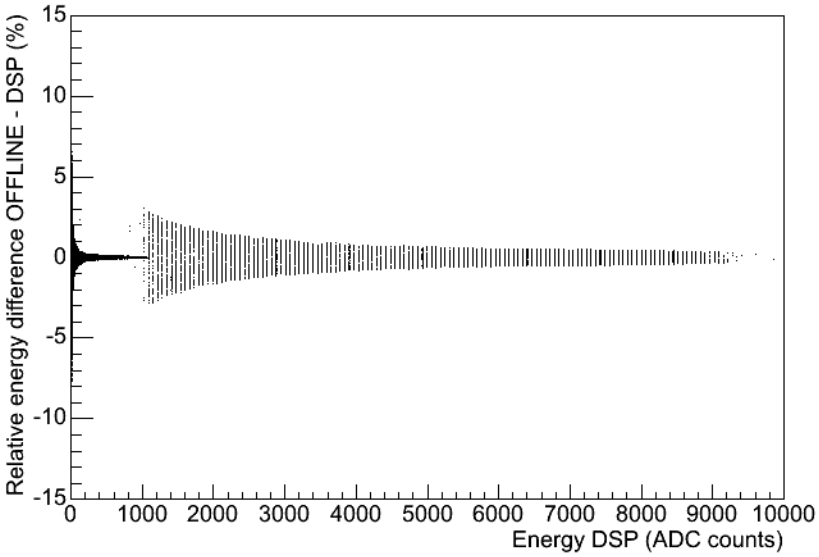

Figure 33: Relative difference (in %) versus OF energy in the DSP (in ADC counts).

5.3.1.2.2 OF with 1 iteration

In Section 5.3.1.2.1 the weights used to calculate the energy reconstruction were assumed for phase equal to *0 ns*. In this Section we calculate the energy using the weights correspondig to this phase τ. Figure 34 shows the reconstructed amplitude with this method (Optimal Filter wit 1iteration).

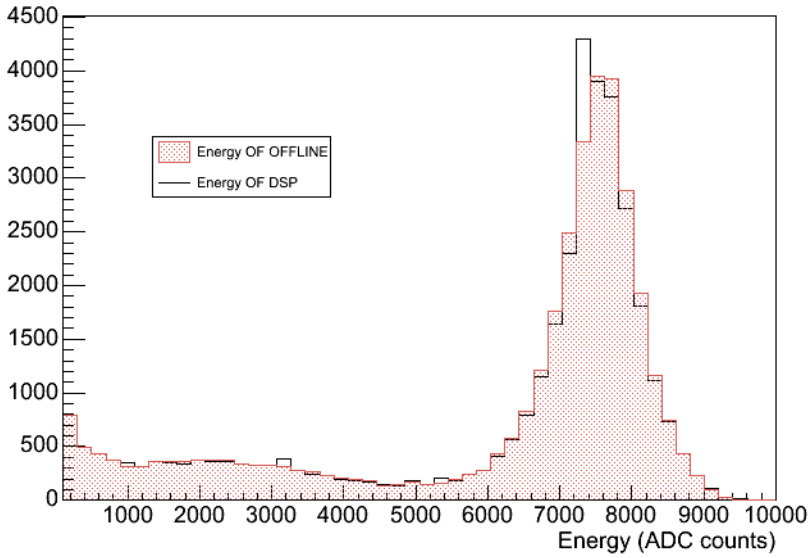

Figure 34: OF reconstruction with 1 iteration for PMT 6 of a negative barrel module of TileCal.

Figure 35 shows the difference between the offline and online reconstructions. The differences are quite similar to the ones with no iterations, the only difference is that there are some counts over the limits of *-0.5* and *0.5 ADC counts* for high gain events and *-32* and *32 ADC counts* for low gain events. In some cases the weight for the iteration are slightly different for offline and online due to the difference in reconstructing the time. The use of different weights makes those differences.

The relative difference, as in the case of zero iterations, does not exceed the *8%* for high gain and *4%* for low gain events.

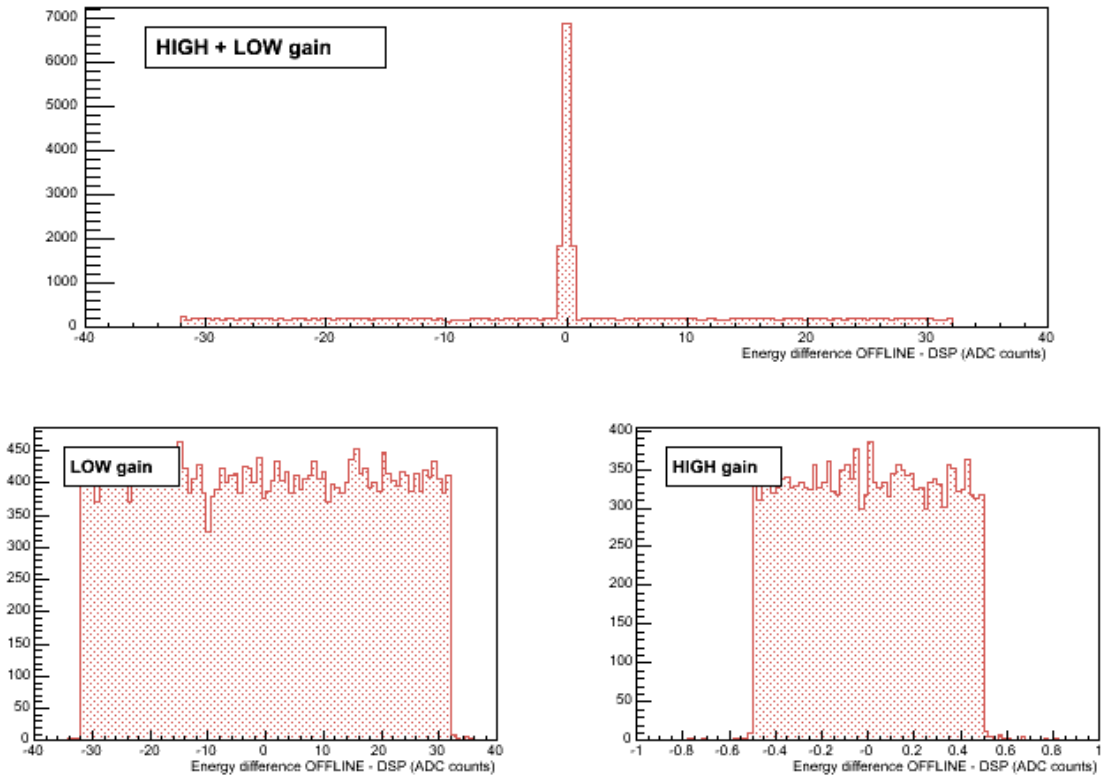

Figure 35: *Top:* Difference between OF 1 iteration offline and online energy reconstruction for all events. *Bottom:* The same as above with low gain events (left) and high gain events (right).

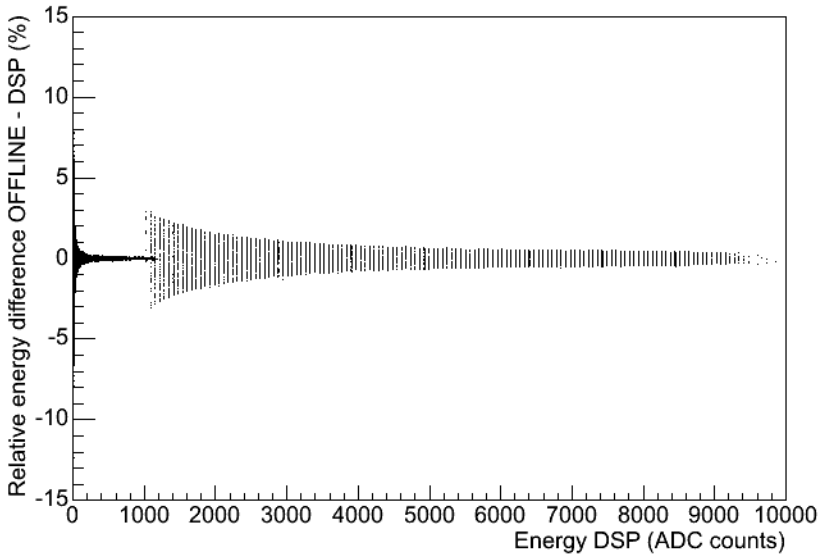

Figure 36: Relative difference (in %) for OF 1 iteration energy versus Energy in the DSP (in ADC counts).

Figure 37 shows a comparison of the offline reconstructed energy with OF using 1 iteration and without iterations for a single channel. Two big differences can be observed between both reconstructions:

- OF without iterations reconstructs lower energies than OF with 1 iteration. This effect comes from the multiplication of incorrect weights. When we apply the correct weights, the maximum sample is multiplied by the maximum weight (which has a value around 0.7). When incorrect weights are applied, the maximum sample is multiplied by a weight which does not correspond to the maximum weight and the energy is then underestimated. This difference is more evident for low gain events (larger energies), where the difference is multiplied by a *64* factor.
- OF using 1 iteration shows a better resolution on the energy calculation.

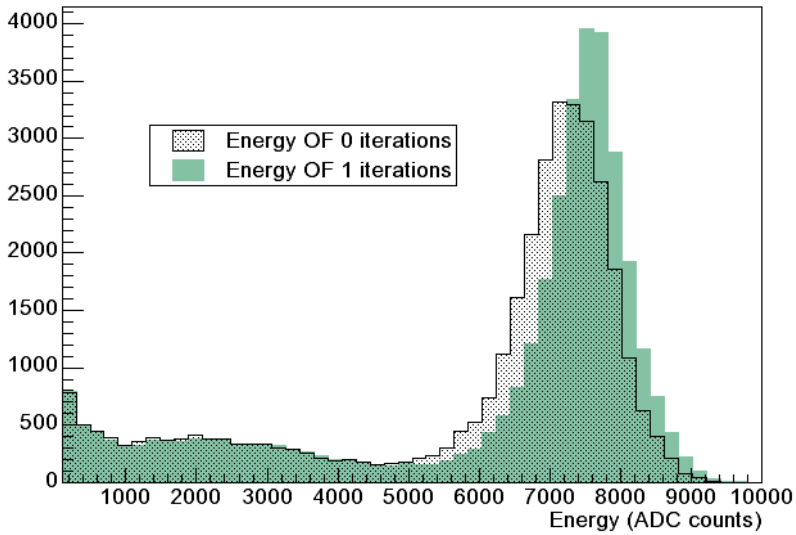

Figure 37: Offline reconstructed energy for OF with 1 iterations and without iterations for PMT 6 of a negative central barrel.

5.3.2 Calorimeter results

In this Section we will study the energy resolution of the Tile Calorimeter as a function of the energy comparing the three algorithms. The energy deposition in a module is calculated by summing the signals from all PMTs of a single module.

In order to compare the three algorithms (FF, OF and FIT) the result of the energy reconstruction has to be expressed in the same units (*pC*). CIS runs are used to calibrate the data by getting the *ADC* to *pC* constants. The CIS is composed by two capacitors, *5.1 pF* and *100pF*, which provide charges between *0 pC* and *40 pC*, or *0 pC* and *800 pC*, respectively. These charges are injected to the shaper and then the signal is amplified and sampled. Eventually samples of a well known amplitude (injected charge) are obtained and reconstructed with the use of these three algorithms.

Figure 38 shows a typical energy distribution from the beam test of TileCal. It represents the sum of the signal in all PMTs of a central barrel module, where a beam of mainly electrons (with

a percentage of pions and muons) at *180 GeV* is coming in. The peak at low energy (less than *50 pC*) is composed by muons, which do not stop in TileCal, and deposit only a small part of its energy in the calorimeter. Between *100 pC* and *300 pC* there are two peaks, one for pions and one for electrons, which are completely absorbed in TileCal. Note that the visible energy for electrons is lager than for pions because the Tile calorimeter is not compensated.

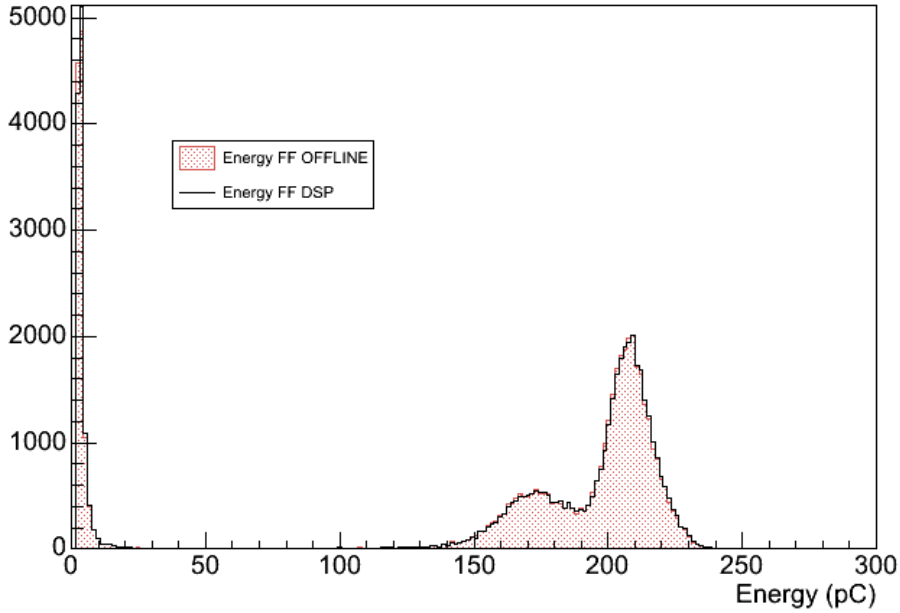

Figure 38: Sum of the signal in all channels in a central barrel module of TileCal. Energy is reconstructed with the FF algorithm as implemented offline and online.

For a proper calculation of the resolution the peaks of the particles should be separated. As the muons do not stop in the calorimeter we will only study the resolution for pions and electrons.

Hadrons are strongly interacting particles; the longitudinal distribution of the hadronic cascade or "shower" is longer than the electromagnetic shower. A good cut could be based on the spatial development of the cascade. As we saw above, TileCal modules are divided in three layers or samples, the first one is composed by cells A, the second one by cells B and C, and the third one by cells D. As a typical hadronic shower is larger than an electromagnetic one, one should get signal in both the first and second layers for pions, while for electrons the main deposition would be in the first layer (see Figure 39). Muons leave very few energy in the whole calorimeter, and they can be identified in the bottom-left corner of Figure 39.

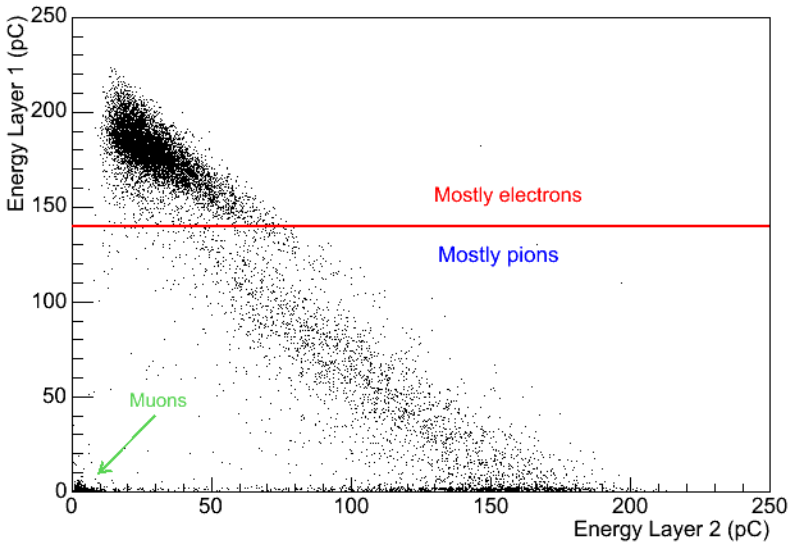

Figure 39: Energy distribution 1st layer of TileCal versus 2nd layer of TileCal.

Table 11 shows a summary of the applied cuts as a function of the energy. L1 corresponds to the deposited energy in the first layer of the calorimeter.

ENERGY (GeV)	FF	OF	FIT	FF	OF	FIT
	ELECTRONS			PIONS		
180	L1 > 140 pC	L1 > 130 pC	L1 > 125 pC	L1 < 140 pC	L1 < 130 pC	L1 < 125 pC

Table 11: Summary of the cuts applied to the analysis.

Figure 40, Figure 41 and Figure 42 show the total reconstructed energy in a TileCal central module calculated with FF, OF and FIT respectively. The first plot of each figure shows the total energy in a module without applying any cut. The second plot of each figure shows the energy deposited in the module by mainly electrons and the third one the energy deposited by pions. When calculating the resolution these peaks are fitted to a Gaussian function (in the $\pm\sigma$ region).

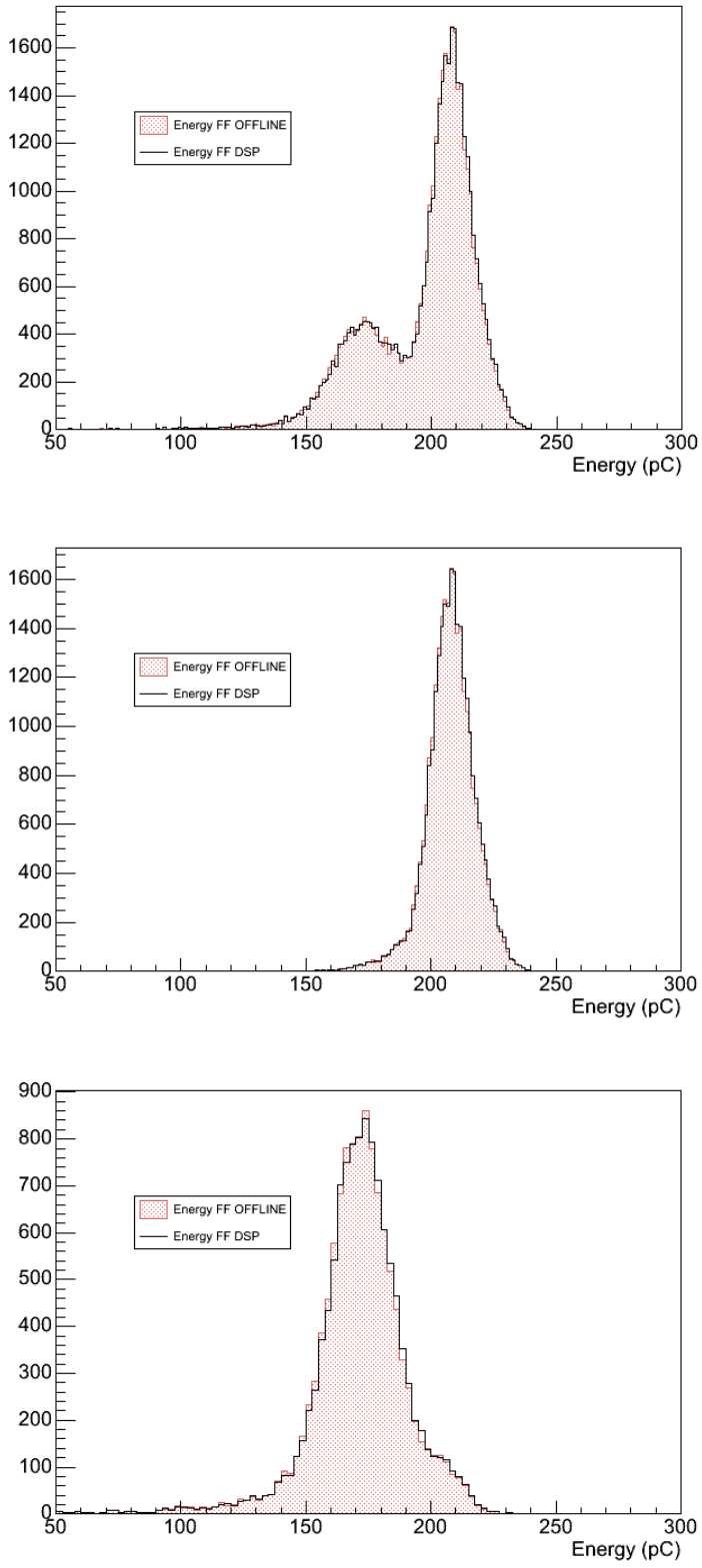

Figure 40: Total energy reconstruction with FF for both the offline and the online implementation of the algorithms. *Top:* sum of all channels in a central module. *Middle:* the same as above but for selected electrons. *Bottom:* the same as above but for selected pions.

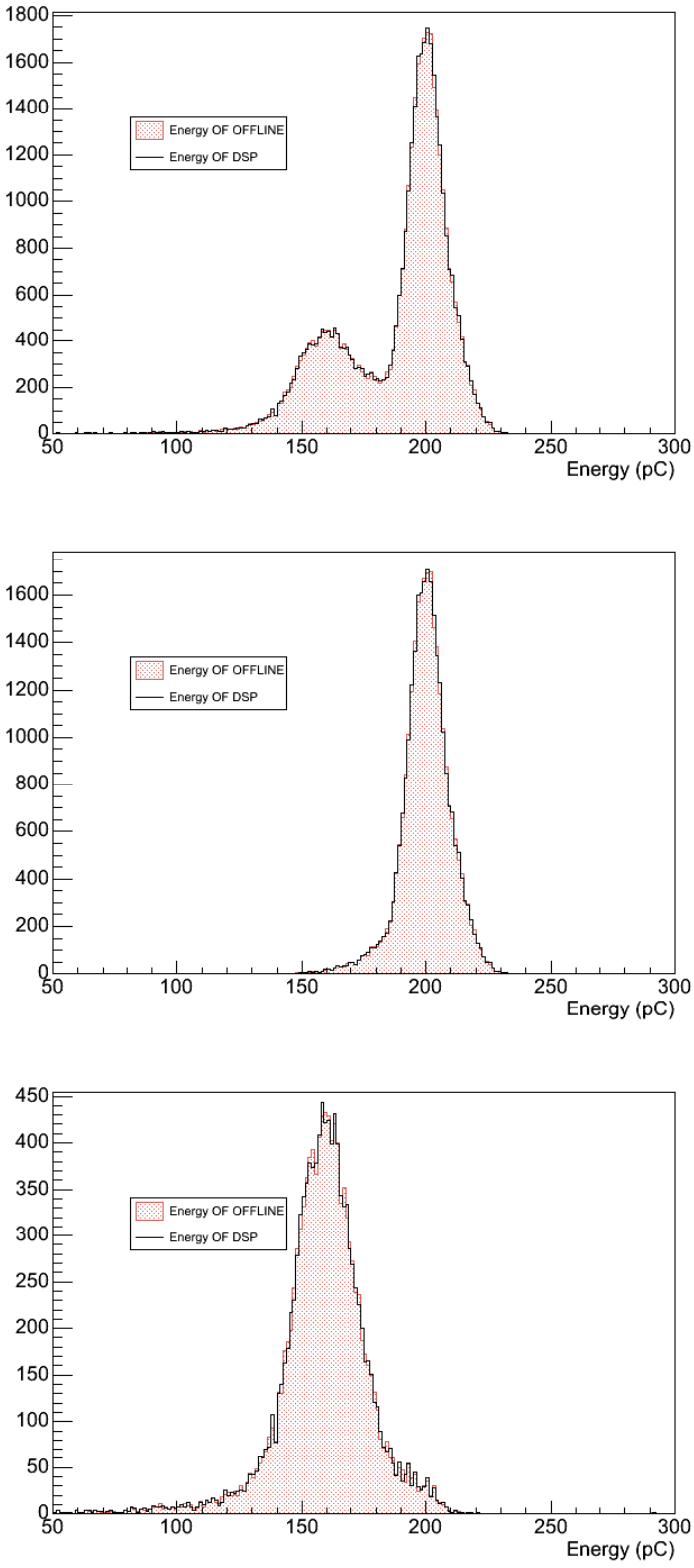

Figure 41: Total energy reconstruction with OF for both the offline and the online implementation of the algorithms. *Top:* sum of all channels in a central module. *Middle:* the same as above but for selected electrons. *Bottom:* the same as above but for selected pions.

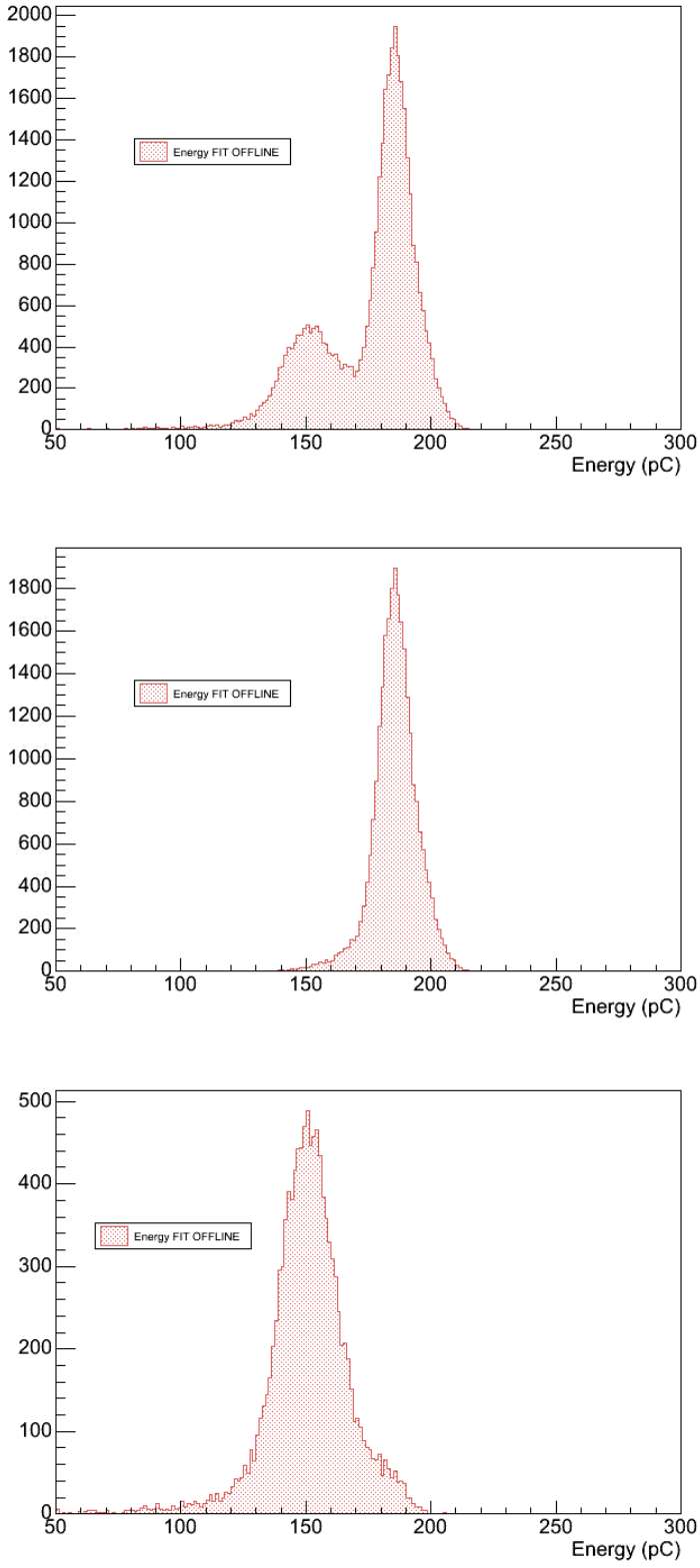

Figure 42: Total energy reconstruction with FIT for both the offline and the online implementation of the algorithms. *Top:* sum of all channels in a central module. *Middle:* the same as above but for selected electrons. *Bottom:* the same as above but for selected pions.

Table 12 shows the results of the Gaussian fit and the energy resolutions for electrons calculated with the three algorithms, FF (offline and DSP), OF (offline and DSP) and FIT (offline). The analyses were done for:

- Electrons at *180 GeV*, coming into cell A-2 of a negative side barrel module with a θ angle of 20º.
- For electrons at *9 GeV*, coming into tile raw 2 of a positive side barrel module with a θ angle of 90º.

ALGORITHM	ENERGY (GeV)	MEAN (pC)	SIGMA (pC)	RESOLUTION (%)
FF OFFLINE	180	209.2 ± 0.2	7.6 ± 0.3	3.6 ± 1.1
FF DSP	180	207.4 ± 0.2	7.2 ± 0.3	3.5 ± 1.2
OF OFFLINE	180	200.8 ± 0.2	6.9 ± 0.2	3.4 ± 0.8
OF DSP	180	200.8 ± 0.5	6.9 ± 0.2	3.4 ± 0.8
FIT OFFLINE	180	186.2 ± 0.2	6.4 ± 0.3	3.4 ± 1.4
FF OFFLINE	9	10.99 ± 0.01	0.90 ± 0.02	8.2 ± 0.3
FF DSP	9	10.90 ± 0.01	0.92 ± 0.02	8.4 ± 0.3
OF OFFLINE	9	10.06 ± 0.01	0.84 ± 0.02	8.3 ± 0.2
OF DSP	9	10.06 ± 0.01	0.83 ± 0.02	8.3 ± 0.2
FIT OFFLINE	9	9.67 ± 0.01	0.81 ± 0.02	8.4 ± 0.3

Table 12: Energy resolution for electrons.

In spite of the restrictions in the instruction types and the re-scaling, the resolution obtained in the DSP is the same as the offline for both FF and OF. Whit respect to FF, OF and FIT there are not differences in the resolution at this energy scale.

The results in Table 12 have been compared with previous studies with data from the beam test period of 2003 [19]. In those analyses the response for pions and electrons was compared as well as the energy reconstruction with FF, OF and FIT. The energy was calculated as the sum of all channels of a TileCal module. Figure 43 and Figure 44 show the resolution as a function of $1/\sqrt{E}$. The results presented in this note agree with the ones in [19]. Comparing Figure 43 and Figure 44 we can observe that OF improves the resolution at very low energies.

The experimental results in [19] are fitted to the standard expression:

$$\frac{\sigma}{E} = \frac{a}{\sqrt{E}} \oplus b$$

where a is the due to statistics variations in the shower development and b is a constant term which reflects uncertainties in the energy measurements.

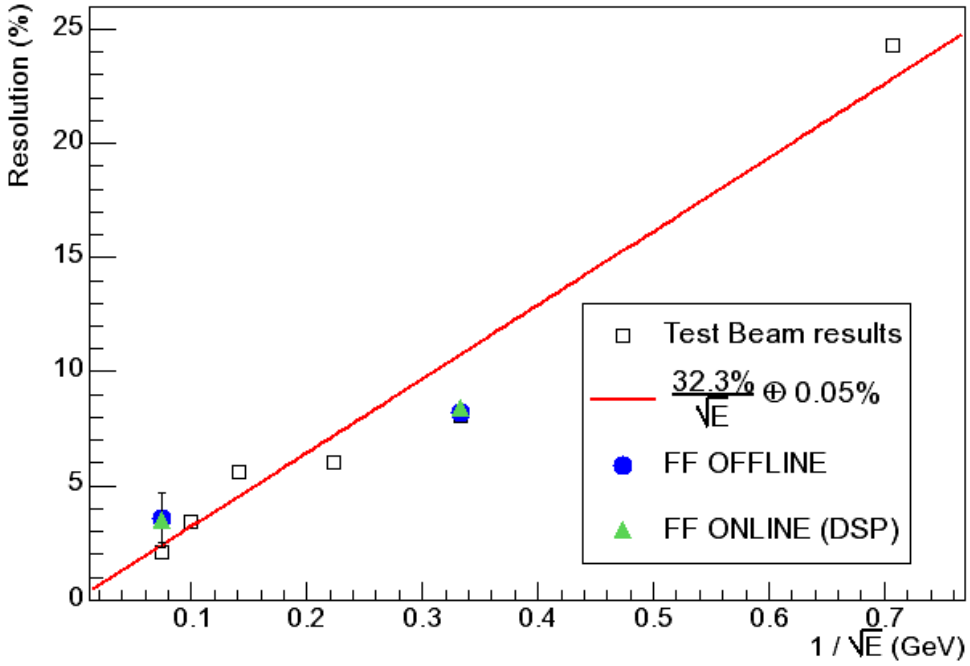

Figure 43: Energy resolution calculated with FF for electrons.

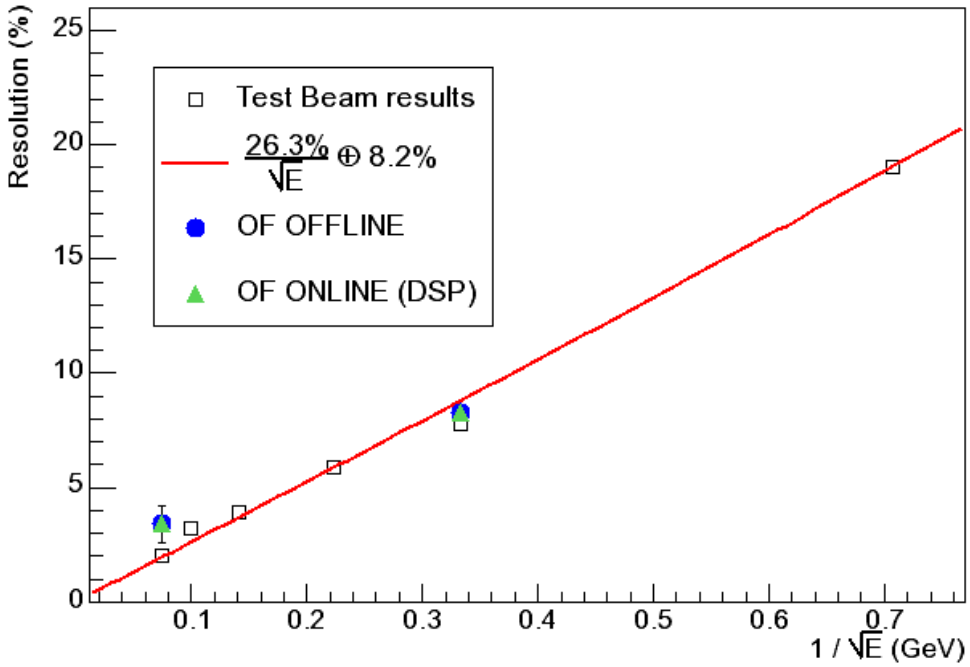

Figure 44: Energy resolution calculated with OF for electrons.

The same analyses were done for:

- Pions at *180 GeV*, coming into cell A-2 of a negative side barrel module with a θ angle of 20º.
- Pions at *100 GeV*, coming into cell A-3 of a negative side barrel module with a θ angle of 20º.

Table 13 shows the mean and the sigma of the Gaussian fit for pions, and the resolutions for the three algorithms.

ALGORITHM	ENERGY (GeV)	MEAN (pC)	SIGMA (pC)	RESOLUTION (%)
FF OFFLINE	180	173.6 ± 0.5	13.5 ± 0.9	7.8 ± 0.9
FF DSP	180	171.2 ± 0.5	13.6 ± 0.9	7.8 ± 0.9
OF OFFLINE	180	160.7 ± 0.5	13.0 ± 0.9	8.1 ± 0.9
OF DSP	180	160.8 ± 0.5	13.1 ± 0.8	8.1 ± 0.8
FIT OFFLINE	180	151.8 ± 0.5	12.3 ± 0.9	8.1 ± 0.9
FF OFFLINE	100	88.8 ± 0.2	9.5 ± 0.3	10.7 ± 0.3
FF DSP	100	87.2 ± 0.2	9.5 ± 0.3	10.9 ± 0.3
OF OFFLINE	100	80.9 ± 0.2	8.7 ± 0.3	10.8 ± 0.3
OF DSP	100	80.9 ± 0.2	8.9 ± 0.3	11.0 ± 0.3
FIT OFFLINE	100	78.8 ± 0.2	8.5 ± 0.3	10.8 ± 0.3

Table 13: Energy resolution for pions.

There are no differences between the resolution calculated offline and in the DSP within the error. The comparison between the three algorithms at *180 GeV* and *100 GeV* does not show large differences in the energy resolution.

Figure 45 and Figure 46 show the resolution for pions at energies between 350 *GeV* and 1*GeV* from [19]. The points were calculated with the sum of the signals of all the channels of a TileCal module reconstructed with FF and OF respectively. The resolutions in Table 13 are superimposed to the figure and they agree with the experimental results from [19]. As in the case of electrons, the improvement on the energy resolution is observed for very low energies when the noise contribution is comparable to the signal.

Comparing now the resolution for electrons and for pions we observe that the resolution for electrons is always better than for pions. The stochastic term of the fit for pions $(a \approx 80\%)$ is about twice as large as the one for electrons $(a \approx 30\%)$. This effect occurs in non compensated calorimeters, as TileCal, where the visible energy for electrons is larger than the visible energy for hadrons.

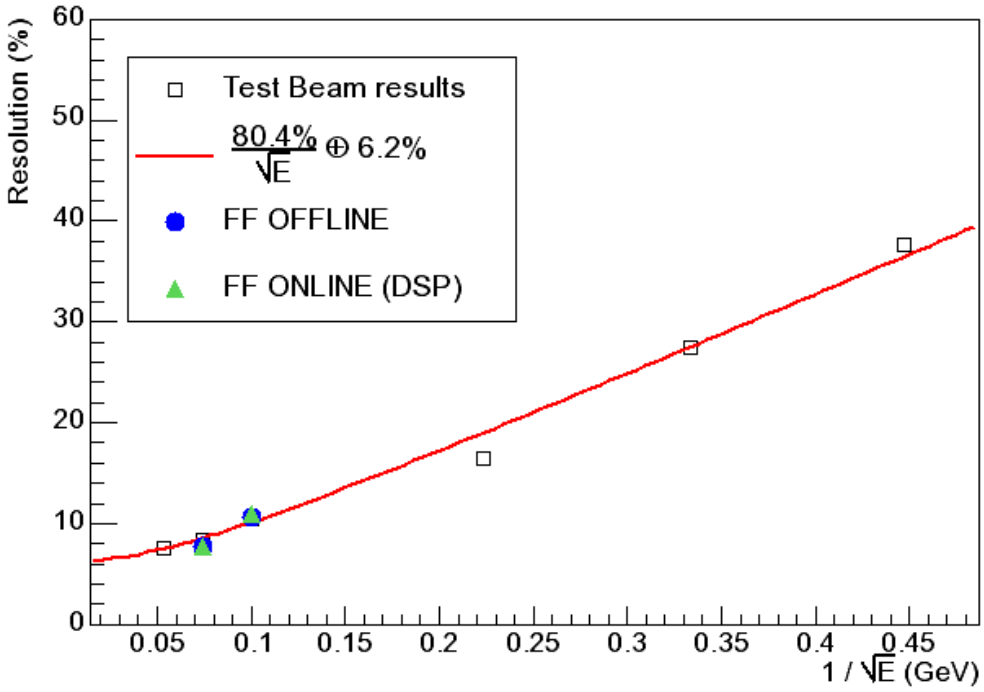

Figure 45: Energy resolution calculated with FF for pions.

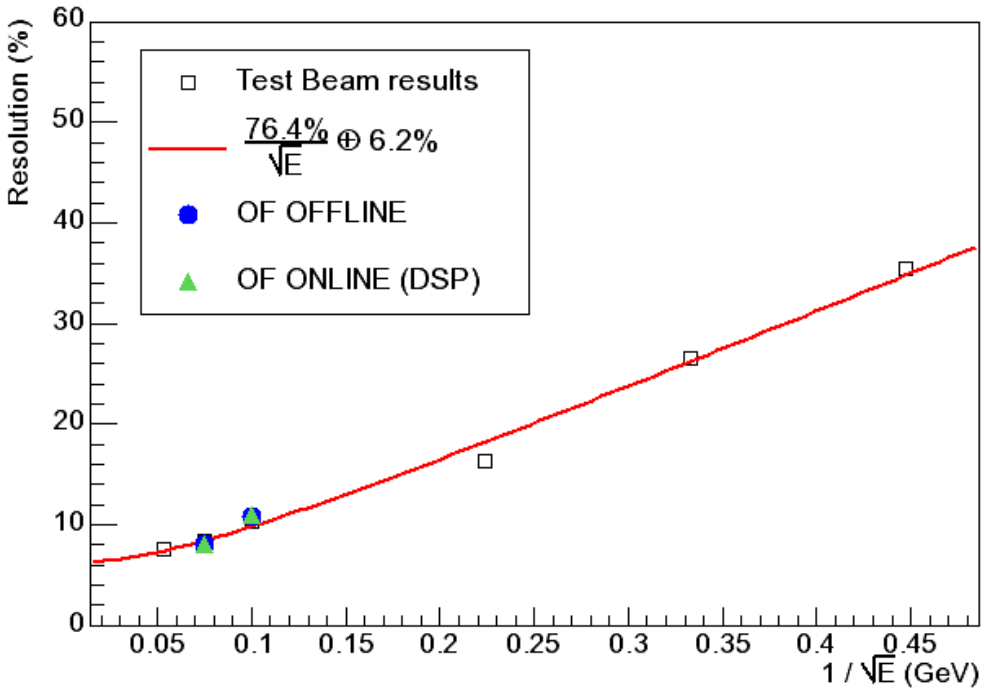

Figure 46: Energy resolution calculated with OF for pions.

5.4 Results on Time

5.4.1 Optimal Filter without iterations

The Optimal Fitler algorithm allows to calculate the time of the arriving signal, usually referred to as phase. In the beam test of TileCal the data taking is asynchronous, i.e., the particles come into the detector with a non well defined time. This behaviour is observed in Figure 47 which shows the phase of the samples for PMT 6 of a negative barrel module. This is a PMT with signal and, as expected, the particles are uniformly distributed on time. As the sample separation in time is always 25 *ns*, most of the events are distributed between -12.5 *ns* and 12.5 *ns*. Nevertheless when the maximum sample does not correspond to the central sample it appears an offset of 25 *ns*, and this range is overcome.

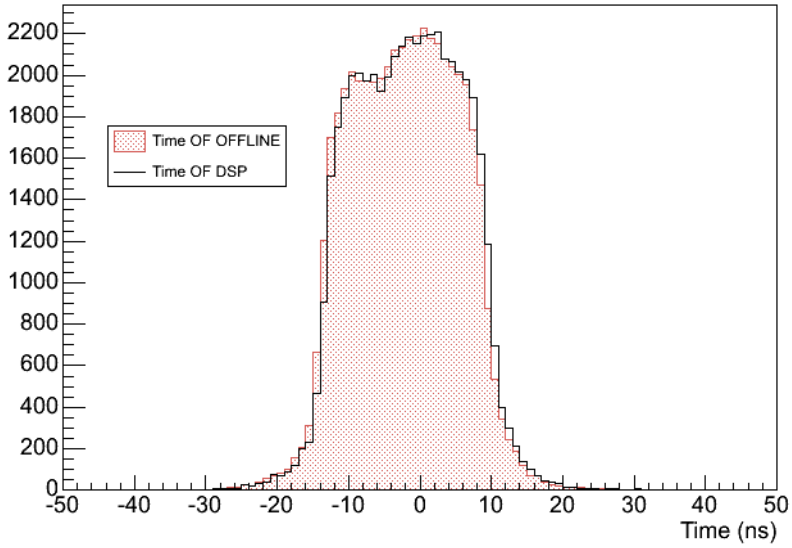

Figure 47: Time reconstruction with OF without iterations.

The difference between the reconstructed phase between the OF (without iterations) offline and DSP algorithms is represented in Figure 48. The differences are distributed mainly between -0.5 *ns* and 0.5 *ns* because the result of the DSP is an integer number. The events with a difference larger than ± 0.5 *ns* correspond to those with larger phases (Figure 49). This can be explained as the weights have been calculated for a fixed phase equal to zero and then applied to events with larger phases, which deteriorates the reconstruction. To correct for this effect iterations are applied. In the next Section the results on time for OF with 1 iteration will be shown.

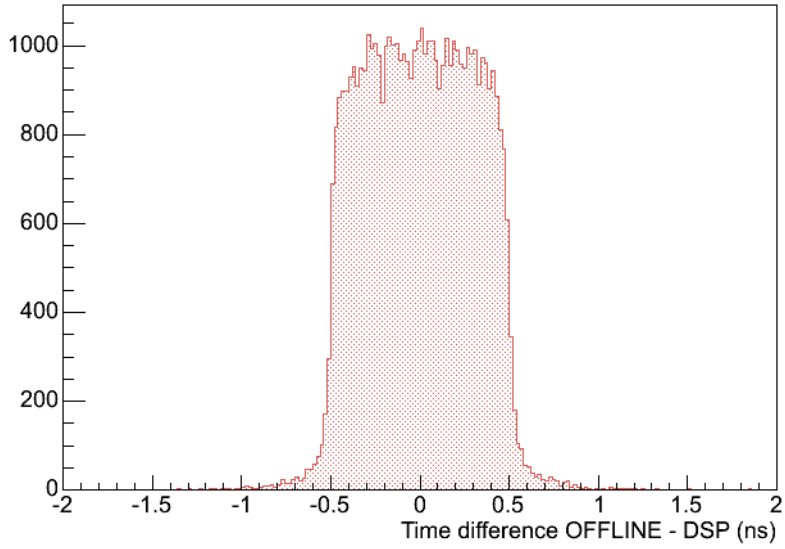

Figure 48: Time difference between OF without iterations offline and online.

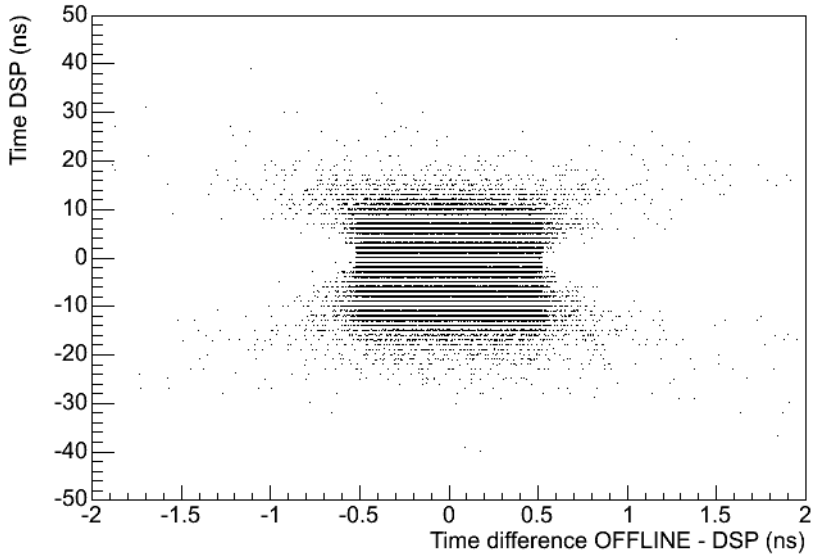

Figure 49: Time in the DSP versus time difference OF without iterations offline minus DSP.

5.4.2 Optimal Filter with 1 iteration

Better results can be obtained if the Optimal Filter with 1 iteration algorithm is applied. The improvement can be observed in Figure 50, where the time distribution is flatter than the one calculated without iterations.

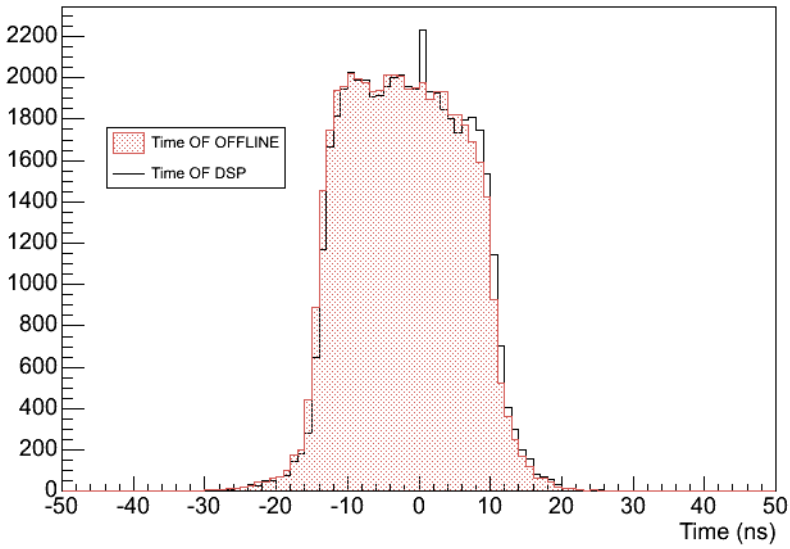

Figure 50: Time reconstruction with OF 1 iteration.

Figure 51 shows the difference between the phase calculated with the OF with 1 iteration offline and DSP algorithm. There are more events between the limits -0.5 *ns* and 0.5 *ns*, which mean that when more iterations are applied the differences between DSP and offline will only be due to the rounding, as the amplitude will be better reconstructed. In Figure 52 the same behaviour is shown.

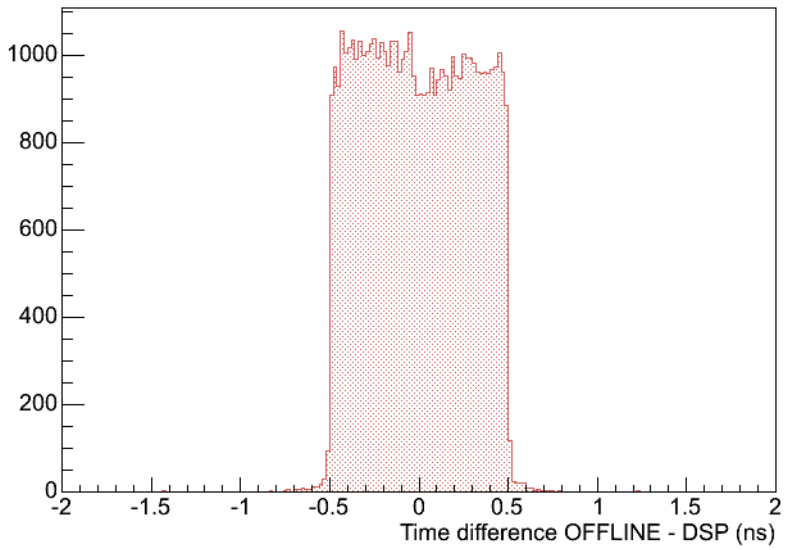

Figure 51: Time difference between OF 1 iteration offline and online.

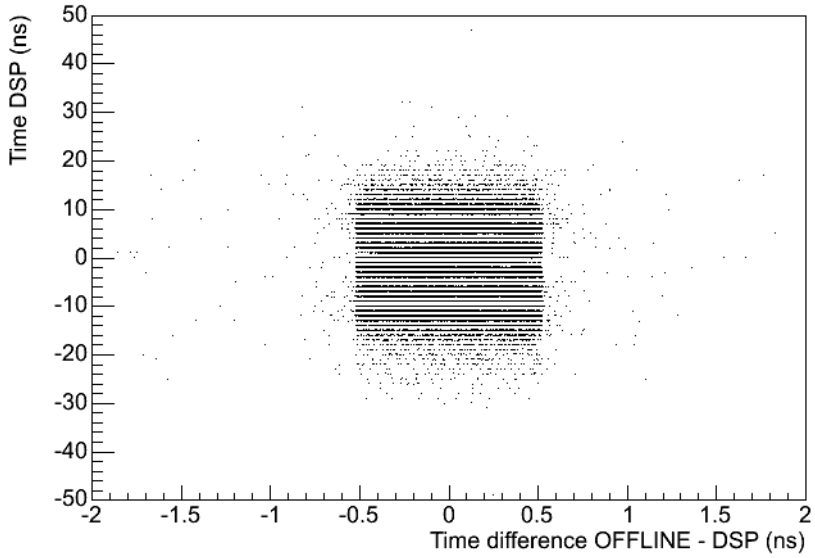

Figure 52: Time in the DSP versus time difference OF 1 iteration offline minus DSP.

6 CONCLUSIONS AND FUTURE PLANS

In this work we have first developed a software library, TileRODFinal, which allows to program and control the ROD boards and its components. The software has been programmed in C++ and is now part of the ATLAS official DataFlow software. For that purpose the work has been done within the CMT environment, the standard online software ATLAS framework, and built the TileRODFinal library as a CMT package.

The final version of the library contains more than 100 C++ methods to write, read and decode information from the following ROD registers:

- Local registers of the ROD motherboard.
- Busy registers of the ROD motherboard.
- Registers of the 4 Output Controller FPGAs.
- Registers of the 4 Staging FPGAs.
- Registers of the TTC FPGA.
- 4 FPGA Processing Units:
  - Registers of the FPGA.
- 4 DPS Processing Units:
  - Registers of the 2 DSPs.
  - Registers of the 2 Input FPGAs.
  - Registers of the Output FPGA.

Besides the control of the ROD boards the TileRODFinal library contains C++ methods to boot the Processing Units. By calling these methods, firmware can be uploaded through VME to the FPGAs as well as software code to control the DSPs of the PUs.

Since February 2004 the TileCal TDAQ collaboration is using this software to configure the ROD for data acquisition during the TileCal commissioning with muons. RODs are, in this way, programmed to acquire data through VME at low rates (~1 Hz) with successful results. During summer 2004 the Combined Beam Test took place and once more RODs have been configured and accessed using the TileRODFinal software.

The library is continuing being updated due to new developments on the electronics and firmware upgrades. The nearly future plans regarding the ROD library are to implement the management of interruptions and busy signals to handle larger DAQ rates and an increased number of read-out channels.

In the second part of this work we have studied the online implementation of the OF energy and time reconstruction algorithms in the ROD DSP processors. This study has been performed through an offline simulation of the TMS320C6414 DSPs of Texas Instruments. The use of these DSPs limits the precision on the energy and time calculations by requiring fast operations and limited memory in order to cope with the limitations of the ATLAS first level trigger.

A good precision on both energy and time reconstructions provided by the DSPs will be necessary for the ATLAS second level trigger and, eventually, to be used also offline. We have shown a good performance obtained with the proposed DSP algorithms, as seen in the comparison between offline and online reconstructions. The main differences between both reconstructions come from the use of integer numbers in the DSPs.

At present the Processing Units with the TMS320C6414 DSPs are available and those studies will continue with the actual implementation of the algorithms inside the DSPs. In this implementation we will include a modification of the Fit method and the calculation of the quality factor χ^2.

7 APPENDIX I: ROD REGISTERS

7.1 LOCAL registers

LOCAL Registers (A10:7) = 0000						
DECIMAL	**BINARY**	**HEXADECIMAL**	**R/W**	**FUNCTION**	**BITS**	**BITS DESCRIPTION**
0	00000	0	W/R	Base Address	31:0	
1	00001	4	W	General Reset		Sent to all chips
2	00010	8	R	Status	9	
					8	
					4	Busy vme blocks busy initialisation
					3:0	PU ready
3	00011	C	R	Board Identifier	7:0	Switch hard on board
4	00100	10	W	Force Busy	0	1: busy forced to 1 (default during initialisation)
					1	1: data bus extern chip VME (vmeconfig)
					2	1: quick answer 2clock(vmeconfig)
					3	1: quick answer 4clock(vmeconfig)
15	01111	3C		Version		

7.2 BUSY registers

BUSY Registers (A10:7) = 0000						
DECIMAL	**BINARY**	**HEXADECIMAL**	**R/W**	**FUNCTION**	**BITS**	**BITS DESCRIPTION**
1	10001	44	W	Reset and Control	16	Send sreq
					8	Write duration busy to fifo
					4	Reset timing
					3	Reset interval wr fifo counter
					2	Reset sreq counter
					1	Reset busy duration
					0	Reset fifo
2	10010	48	R	Status	25	Busy active
					24	Busy PU
					23:16	Mask busy
					14:13	Fifo (FF-4xxx) , (EF 2xxx)
					12	Enable Sreq
					11	Sreq out
					10	Sreq in
					9	Enable busy
					8	Busy out
					7:0	Busy in
3	10011	4C	W/R	Miscellaneous	23:16	Mask busy
					12	Enable sreq
					9	Enable busy
					1	**1:** mode normal
					1	**0:** mode manual
					0	Busy force, **0:** pu, **1:** VME
4	10100	50	W/R	Interval wr fifo	31:0	Write to fifo each (div_clock*value)
5	10101	54	W/R	Sreq max	31:0	Sreq sent if count_busy > value

6	10110	58	W/R	Div clock	7:0	Busy count each (100 ns + 25ns·(n-1))
7	10111	5C	R	Read fifo	19:18	Full fifo 8xxxx/ empty fifo 4xxxx before reading fifo
					17:16	Full fifo 2xxxx/ empty fifo 1xxxx after reading fifo
8	11000	60	R	Read duration busy	15:0	
9	11001	64	R	Read sreq counter	31:0	
A	11010	68	W	Send busy	15:8	Send busy

7.3 OUTPUT CONTROLLER registers

OUTPUT CONTROLLER Registers (A10:7) = 1000 → 1011						
DECIMAL	**BINARY**	**HEXADECIMAL**	**R/W**	**FUNCTION**	**BITS**	**BITS DESCRIPTION**
0	00000	400	W/R	Dummy register	31:0	
1	00001	404	W	Reset		
2	00010	408	R	Status	26	End of event (eOe) flag, **1:** eOe not found
					25	Link failure
					24	Memory complete
					23	Memory empty
					22	Data available, **1:** Memory can be read, generate IRQ
					21:0	Memory size, write pointer address
4	00100	410	W/R	Config	28	Xfer all 1 equal to all data transfer
					27	Enable test link **1:** link in test mode. Pulse bit: Only W (not R/W). When a **1** is written there is a pulse sent to TM HOLA that manages test mode
					36:24	Data takin mode, enable spy, enable Slink
						1x0: data to Sdram **0x0:** data storage stopped in Sdram
						111: next event to Slink and Sdram
						011: enable VME event read from Sdram
					21:20	In_out_n, staging mode
						11: fifo2 to previous OC **01:** fifo2 to next OC
					17:16	Mask fifo 2, mask fifo 1
					13:0	Bank max 1, bank max 0, row max (11:0) only for Sdram
8	00110	418	R	Sdram register		Read the Sdram
16	11111	47C	W	Version		

7.4 STAGING registers

STAGING Registers (A10:7) = 1100 → 1111, 60-68-70-78						
DECIMAL	**BINARY**	**HEXADECIMAL**	**R/W**	**FUNCTION**	**BITS**	**BITS DESCRIPTION**
0	00000	600	W/R	Dummy register	31:0	Read/Write general-purpose
1	00001	604	W	Set/Reset register	20	Rst_mem_led (reset latched fifo flags)
					19	Rst_mem_fifo (reset latched fifo flags)
					18	Rst_min_max temp

					17:16	Rst_count_feb2, feb1 error
					11:10	Error link2s, link1s (VME set- debut RAM purpose)
					9:8	Link not ready link2s, link1s (VME set-debug RAM purpose)
					7:6	Error link2, link1 (VME set - debug RAM pupose)
					5:4	Link not ready link2, lin1 (VME set - debug RAM purpose)
					3:2	Reset link2s , link1s
					1:0	Reset link2, link1
2	00010	608	R	Status register	31:30	Feb2_cav, feb1_cav
					28:29	Feb2_cav, feb1_cav
					27:24	Feb2_lnk_rdy, feb1_lnk_rdy, feb2_error, feb1_error
					23:22	Real time full2, empty 2 of internal feb FIFO 2
					21:20	Real time full1, empty 1 of internal feb FIFO 1
					19:18	Latched full2, empty 2 of internal feb FIFO 2
					17:16	Latched full1, empty 1 of internal feb FIFO 2
				Count_feb2_error	15:8	If using injector sendin incremented data. Number of errors in FEB2 (only a counter through g-link) else latched mem_led_high (bit0=mem_error_feb2) → could be reseted
				Count_feb1_error	7:0	If using injector sendin incremented data. Number of errors in FEB1 (only a counter through g-link) else latched mem_led_high (bit0=mem_error_feb1) → could be reseted
4	00100	610	W/R	Configuration 1	9:0	Number of words per event in RAM
					31:16	Number of events to send from RAM
5	00101	614	W/R	Configuration 2	31:30	Reserved
					29	FEB2_CTRL_N
					28	FEB1_CTRL_N (**0** when header 0x51115110 and trailer 0xFFFFFFF0, and **1** for event data)
				Delay between events	27:16	Delay Nb*25ns between events sent from RAM
				Disable input	14	**1:** FEB input is not disable if g-link not locked **0:** FEB input disable if g-link not locked
				Infinite loop	13	**1:** Data from RAM is sent infinite time with a loop **0:** Finite number of events noted in conf_reg1
				En_busy_stag	12	**1:** Enable PU busy signal from dsp1_busy and dsp2_busy. This is why this version is called stag_busy. Link_error signals, 1 and 2 are used as inputs with this behaviour **0:** The busy from DSP will not be taken into account
				LED_MODE	11:10	**"00"** => FEB_mode1 (LED_HIGH&LED_LOW = lnkerror or linkrdyn feb1 and 2) **"01"** => FEB_mode2 (LED_HIGH = lights start of event feb1, LED_LOW: feb2; visible up to 100Hz) **"10"** => RAM_mode (LED_HIGH = lights when event is sent from RAM,

						LED_LOW: lights if infinite
					9	Not used
				Feb_input enable	8	**1:** FEB data to PU **0:** Int. RAM data to PU
					7	Not used
				Pu_not_rdy	6	**0:** PU is ready **1:** PU is not ready (should be taken from LOCAL FPGA pu-rdy (3:0) bit at startup)
				Stag2puOK	5	In stagin mode: **1:** data_out to own PU **0:** data out to other PU
				Staging	4	**1:** stagin mode **0:** not in staging mode
				Sel_out	3	**0:** datafeb2s = 0 **1:** datafeb2s= feb2sor data ram(31-16)
					2	**0:** datafeb2 = 0 **1:** datafeb2= feb2or data ram(31-16)
					1	**0:** datafeb1s = 0 **1:** datafeb1s = feb1s or data ram (15-0)
					0	**0:** datafeb1 = 0 **1:** datafeb1 = feb1 or data ram (15-0)
6	00110	618	W/R	Data to Ram	31:0	Data to RAM. Could be read after write of each event
7	00111	61C	W/R	Start Address	9:0	Actual version only allows 512 words: address (8:0). Write initial address (is internally incremented after you send data to RAM)
8	01000	620	W	Start TX	0	Start to send event with a delay configured with “Delay between Events”. Should b e asserted after an event is stored in RAM.
9	01001	624	W/R	Link_confi	5:4	Div1, div0
					3:2	Feb_eqn2, 1
					1:0	Feb_flag 2, 1
16	10000 - 10111	740 → 75C	R	Temperature	23:0	23:16 max temperature 15:8 min temperature 7:0 actual temperature
	11111	67C		Version		

7.5 TTC registers

TTC Registers (A10:7) = 0010						
DECIMAL	**BINARY**	**HEXADECIMAL**	**R/W**	**FUNCTION**	**BITS**	**BITS DESCRIPTION**
0	00000	100	R/W	Dummy		
1	00001	104	R/W	Control	ctrl_reg bits	High is active
					0	VME
					1	LOCAL
					2	TTC
					3	CLK_SEL
					4	L1A_MODE
					5	FL_BUFF
					6	STAT_CLEAR
					7	TTC_RESET
					8	EV_CNT_RES_CNT_RESET
2	00010	108	W	BCID	12:0	

3	00011	10C	W	Ttype	8:0	
4	00100	110	R	Status	Stat_reg_bits	High is active
					7:0	SIN_ERR
					15:8	DB_ERR
					23:16	DL_ERR
					24	TTC_READY
					25	TTC_CLK
					26	OLDWR_RD
6	00110	118	R/W	Ttype subaddress	Ttype subaddress	High is active
					7:0	Ttype_subaddr
F	01111	13C	R	Version	Version	High is active
					31:0	

7.6 DUMMY PU registers

DUMMY PU Registers (A10:7) = 0100 → 0111						
DECIMAL	**BINARY**	**HEXADECIMAL**	**R/W**	**FUNCTION**	**BITS**	**BITS DESCRIPTION**
0	00000	200	R/W	Dummy		
1	00001	204	W	Reset	6	Reset latched busy and FIFO flags
					5:4	5: fifo_int , 4: fifo_ext
					3:2	Reset feb2s , feb1s
					1:0	Reset feb2, feb1
2	00010	208	R	Status	28:27	Busy2 , busy1 (real time)
					26:25	Latched busy2, busy 1
					24:17	Latched pan2, pan1, paf1, full2, full1, empty2, empty1 EXT FIFO
					16	Config(0), 1: read normal, 2: read stag
					11:8	Full2, full1, empty2, empty1 INT FIFO
					7:0	Pan2, pan1, paf2, paf1, full2, full1, empty2, empty1 EXT FIFO
4	00100	210	R/w	Configuration	31:16	Pulse length
					0	**1:** fifo1 - 2 ← feb 1- 2 **0:** fifo 1- 2 ← feb 1- 2S
					1	Led_ini: **1:** LEDs display start of evt, busy and fifo state **0:** knight rider...
					3:2	TTC_SEL: 00 => BCID, Ttype and EvtID taken from VME reg (TTCpr compatible). 01 => BCID extracted from feb data and Event_ID incremented after an initial value stored in Extended_L1ID register (not used ECR so EvtID will be 24 bits), Ttype taken from VME reg. 10 => TTtpe, BCID, EVTID take from TTC_FPGA=>TTCrx link. 11 => BCID from VME a and Event_ID incremented after an initial value estored in Extended_L1ID register (not used ECR so EvtID will be 24 bits), Ttype taken from VME reg.
					4	**1:** PU_header G-LINK/FIFO1

						enabled **0:** disabled
					5	**1:** PU_header G-LINK/FIFO2 enabled **0:** disabled
5	00101	214	R	Read fifo_int		
6	00110	218	R/W	Pulse and set/reset busy	9:8	Set reset, 9: irq 2, 8: irq 1
					5:4	Set reset, 5: busy2 , 4: busy 1
					1:0	Pulse busy, 1: busy2, 0: busy 1
7	00111	21C	R/W	Format version number	31:0	
8	01000	220	R/W	Source identifier	31:0	
9	01001	224	R/W	Run number	31:0	
10	01010	228	R/W	Extended L1ID	31:0	
11	01011	22C	R/W	BCX_ID / LEVEL 1 TTYPE	31:0	
					11:0	BCX
					23:16	TType
12	01100	230	R/W	Detector event type	31:0	
13	01101	234	R/W	Number of words per event	15:0	
14	01110	238	R	Number of events FEB1	31:0	
15	01111	23C	R	Number of events FEB2	31:0	
16	10000	240	R	Link_error counter g-link 1	31:0	Number of times link_error_signal was asserted in g-link
17	10001	244	R	Link_ready_counter g-link 1	31:0	Number of times link_ready_signal was lost in g-link
18	10010	248	R	Link_error counter g-link 2	31:0	Number of times link_error_signal was asserted in g-link
19	10011	24C	R	Link_ready_counter g-link 2		Number of times link_ready_signal was lost in g-link
20	10100	250	R/W	Source identifier FEB 2	31:0	
21	101001	254	R/W	Sub fragment FEB1	31:0	
22	10110	258	R/W	Sub fragment FEB 2	31:0	
23	11111	27C	R/W	Version		

7.7 DSP PU registers

DSP Registers (A10:7) = 0100 → 0111						
DECIMAL	**BINARY**	**HEXADECIMAL**	**R/W**	**FUNCTION**	**BITS**	**BITS DESCRIPTION**
0	00000	0	R/W	Test register		
1	00001	4	R/W	Control 1	0	DSP reset
					1	Partial FIFO reset
					2	Input FPGA reset
					3	HPI reset
					4	HPI burst
					5	DSP launch
					6	/Input fpga nconfig
2	00010	8	R/W	HPI (DSP1)		

3	00011	C	R	Status	7:0	FIFO counter status
					8	output FIFO empty flag
					9	output FIFO almost empty flag
					10	output FIFO half full flag
					11	output FIFO Almost full flag
					12	output FIFO full flag
					[14..13]	FIFO counter status [9..8]
					[20..16]	Nb of words in McBSP2 FIFO
					21	McBSP2 FIFO empty flag
					22	McBSP2 FIFO almost full flag (> 24 words)
					23	McBSP2 FIFO full flag (32 words)
					24	GP11
					25	GP12
					26	GP13
					27	output FIFO full flag
					28	HPI INT
					29	HPI Ready
					30	InFPGA nstatus
					31	InFPGA confdone
4	00100	10	W	Broadcast HPI (both DSP)		
5	00101	14	W	serial data to McBSP2 (DSP1)		
6	00110	18	R	Serial data from McBSP2 (DSP1)		
7	00111	+1C	W	InFPGA1 configuration (16 LSB)		
8	01000	20	W	InFPGA1 programmation (8 MSB)		
9	01001	24	R	InFPGA1 status		
10	01010	28	W	Broadcast InFPGA programmation (both InFPGA)		
11	01011	+2C	R	OutFPGA version		
16	10000	40	R/W	Test register		
17	10001	44	R/W	control 2		
18	10010	48	R/W	HPI (DSP2)		
19	10011	+4C	R	Status 2		
20	10100	40	W	Broadcast HPI (both DSP)		
21	10101	54	W	serial data to McBSP2 (DSP2)		
22	10110	58	R	Serial data from McBSP2 (DSP2)		
23	10111	+5C	W	InFPGA2 configuration (16 LSB)		
24	11000	60	W	InFPGA2 programmation		

				(8 MSB)		
25	11001	64	R	InFPGA2 status		
26	11010	68	W	Broadcast InFPGA programmation (both InFPGA)		
27	11011	+6C	R	OutFPGA version		

7.8 IRQ registers

IRQ Registers (A10:7) = 00011						
DECIMAL	**BINARY**	**HEXADECIMAL**	**R/W**	**FUNCTION**	**BITS**	**BITS DESCRIPTION**
1	00001	184	W	reset	13:0	reset pending and request IRQ by VME
2	00010	188	R	status1	13:0	state of input IRQ (cf list) before mask
					23:17	state of REAL output IRQ after enable
2	00011	18C	R	status2	13:0	state of interrupt request not sent
					28:17	state of interrupt request to be sent
3	00100	190	W/R	mask/enable	16	enable irqout
					13:0	mask irqinput
9	00101	194	W/R	mapping 1 ouput IRQ	7:1	1 output IRQ
					10:8	1 output coded
						102=IRQ1,204=IRQ2, 308=IRQ3, 410=IRQ4
						520=IRQ5, 640=IRQ6
10	10000	1C0		INPUT IRQ STATUS ID	7:0	pu1_irq1(0) defaut 1
	10004	1C4				pu1_irq2(1) defaut 2
	10010	1C8				pu2_irq1(2) defaut 3
	10011	1CC				pu2_irq2(3) defaut 4
	10100	1D0				pu3_irq1(4) defaut 5
	10101	1D4				pu3_irq2(5) defaut 6
	10110	1D8				pu4_irq1(6) defaut 7
	10111	1DC				pu4_irq2(7) defaut 8
	11000	1E0				oc1_irq(8) defaut 9
	11001	1E4				oc2_irq (9) defaut 10 A
	11010	1E8				oc3_irq (10) defaut 11 B
	11011	1EC				oc4_irq(11) defaut 12 C
	11100	1F0				irq_temp(12) defaut 13 D
	11101	1F4				irq_busy_direct ON(13) defaut 14 E

8 APPENDIX II: DESCRIPTION OF THE TILERODFINAL SOFTWARE METHODS

8.1 Class CRATE

FILES

RODCrate.h

RODCrate.cc

PUBLIC VARIABLES MEMBERS

NAME	DESCRIPTION
int NoRODModules	Number of ROD modules inside a Crate.
int NoAUXModules	Number of Auxiliary modules inside a Crate.
ROD* pRod[MAXNORODS]	Array of pointers to an object of the ROD class.
AUX* pAux[MAXNOAUXS]	Array of pointers to an object of the AUX class.
VME* pVme	Pointer to an object of the VME class.

PUBLIC METHODS MEMBERS

8.1.1 CRATE ( )

Description

Constructor of the class, it does nothing.

8.1.2 ~CRATE ( )

Description

Destructor of the class, it does nothing.

8.1.3 ROD_Error CrateInit (int NoROD, int NoAUX, int DeviceName)

Description

This method initializes the crate by allocating the required memory for the pRod[ ] and pAux[ ] pointers. It also initializes the variable members NoRODModules and NoAUXModules and opens the VMEbus library/driver. It must be called at the beginning of any application to allow VME access.

Input values

NAME	DESCRIPTION
int NoROD	Number of ROD modules inside a Crate.
int NoAUX	Number of Auxiliary modules inside a Crate.
Int DeviceName	The device name can be set to **VP110** or to **BIT3.**

Return values

It returns ROD_SUCCESS on success and the corresponding eror on failure. Possible errors in this method are: ROD_FAIL_NORODS, ROD_FAIL_NOAUXS, ROD_FAIL_ALLOC or the error code of the vme_rcc and bit3_rcc libraries.

8.1.4 ROD_Error CrateShutDown ( )

Description

It must be called before ending any application. CrateShutDown( ) frees the memory of several internal pointers, pRod[ ] and pAux[ ], and closes the VME library/driver.

Return values

It returns ROD_SUCCESS on success and the error code on failure.

8.2 Class ROD

FILES

RODModule.h

RODModule.cc

PUBLIC VARIABLES MEMBERS

NAME	DESCRIPTION
int npus	Number of Processing Units inside a ROD (up to 4).
int slot	Crate slot number where a ROD motherboard is placed.
VMEMasterMap * pVmm	Pointer to an object of the VMEMasterMap classs.
VME* p_My_Vme	Pointer to an object of the VME class.
u_int base_address	Base Address of a ROD board, It is built when the RODInit( ) method of the ROD class is called.
u_int window_size	The window size is initialized when the RODInit( ), method of the ROD class, is called.
u_int add_modif	The address modifier is initialized when the RODInit( ), method of the ROD class, is called.
u_int options	Set to '0'.
LOCAL* pLocal	Pointer to an object of the LOCAL class.
BUSY* pBusy	Pointer to an object of the BUSY class.
OC* pOc[MAXNOOCS]	Array of pointers to an object of the OC class.
TTC* pTtc	Pointer to an object of the TTC class.
STAGING* pStaging[MAXNOSTAG]	Pointer to an object of the STAGING class.
FPGA_PU* pFpga_pu[MAXNOPUS]	Pointer to an object of the FPGA_PU class.

PUBLIC METHODS MEMBERS

8.2.1 ROD ( )

Description

Constructor of the class, it does nothing.

8.2.2 ~ROD ( )

Description

Destructor of the class. It frees the memory space reserved for the pointers pLocal, pBusy, pOc[ ], pTtc[ ], pStaging[ ] and pFpga_pu[ ] defined as public variable members. It unmaps VME by calling to the MasterUnmap( ) method of the VME class.

8.2.3 ROD_Error RODInit (int NoPu, int Slot)

Description

This method must be called before accessing any register in the ROD motherboard. It initializes the variables members of the class, performs a VME map of the ROD board by calling to the MasterMap( ) method of the VME class and reserves the necessary memory space for the declared pointers. Also checks if the ROD module is really there by writing the hexadecimal word 0xbabebabe on the ROD Base Address.

Input values

NAME	DESCRIPTION
int NoPu	Number of PU modules inside a the ROD (up to 4).
int Slot	Crate slot number where a ROD motherboard is installed.

Return values

If no errors occur RODInit( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.3 Class AUX

FILES

AUXModule.h

AUXModule.cc

PUBLIC VARIABLES MEMBERS

NAME	DESCRIPTION
int slot	Crate slot number where a auxiliary module is set.
VMEMaterMap* pVmm	Pointer to an object of the VMEMasterMap class.
u_int base_address	Base Address of an Auxiliary module. It is initialized when running the AUXInit( ) method of this class (see below).
u_int window_size	The windows size is initialized when the AUXInit( ) method is called.
u_int add_modif	The address modifier is initialized when the AUXInit( ) method is called.
u_int options	Set to '0'.

PUBLIC METHODS MEMBERS

8.3.1 AUX ( )

Description

Constructor of the class, it does nothing.

8.3.2 ~AUX ( )

Description

Destructor of the class. It does the master unmaping of the auxiliary module done in AUXInit( )

8.3.3 ROD_Error AUXInit (int Slot)

Description

This method must be called before accessing any register in the AUX VME module. It initializes the variables members of the class, and does a VME map of the auxiliary board by calling to the MasterMap( ) method of the VME class.

Input values

NAME	DESCRIPTION
int Slot	Crate slot number where an auxiliary card is installed.

Return values

If no errors occur RODInit( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.4 Class LOCAL

FILES

RODLocal.h

RODLocal.cc

PUBLIC VARIABLES MEMBERS

NAME	DESCRIPTION
VMEMasterMap* p_My_Vmm	Pointer to an object of the VMEMasterMap classs.

PUBLIC METHODS MEMBERS

8.4.1 LOCAL ( )

Description

Constructor of the class, it reserves memory space for private pointers.

8.4.2 ~LOCAL ( )

Description

Destructor of the class. It frees the memory space reserved for private pointers.

8.4.3 ROD_Error WriteBaseAdd (u_int data, int mode)

Description

This method writes an unsigned integer word on the Local Base Address register of the ROD module.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.
u_int data	32-bit word to be written in the Local Base Address register of the ROD board.

Return values

If no errors occur WriteBaseAdd( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.4.4 ROD_Error ReadBaseAdd(u_int* value, int mode)

Description

This method reads in the Local Base Address register of the ROD module and keeps the value read in the pointer value. It returns an unsigned integer word with the corresponding error based on the VME Error Code from rcc_vme package or bit3_rcc package.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from the Local Base Address register of ROD module.

Return values

If no errors occur ReadBaseAdd( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.4.5 void PrintBaseAdd (u_int value)

Description

This method prints on the standard output the value read from the Base Address register of the ROD module.

Input values

NAME	DESCRIPTION
u_int value	Value to print on the screen

8.4.6 ROD_Error GeneralReset(int mode)

Description

This method sends a General Reset to the ROD board.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur GeneralReset( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.4.7 ROD_Error ReadStatus (u_int* value, int mode)

Description

This method reads in the Local Status register of the ROD board.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit words read from the Local Status register of the ROD module.

Return values

If no errors occur ReadStatus( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.4.8 void PrintStatus (u_int status)

Description

This method decodes and prints to the standard output the information read in the Local Status register of the ROD cards.

Input values

NAME	DESCRIPTION
u_int status	Word read from the Local Status register.

8.4.9 ROD_Error ReadBoardId (u_int* value, int mode)

Description

This method reads in the Local Board Identifier register of the ROD.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from the Local Board Identifier register.of the ROD module.

Return values

If no errors occur ReadBoardId( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.4.10 void PrintBoardId (u_int identif)

Description

This method decodes and prints to the standard output the information read in the Local Board Identifier register of the ROD module.

Input values

NAME	DESCRIPTION
u_int identif	Word read from the Local Board Identifier register of the ROD module.

8.4.11 ROD_Error ForceBusy (u_int data, int mode)

Description

This method writes in the Local Force Busy register of the ROD.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.
u_int data	32-bit word with the information to be written in the Local Force Busy register: bit 0 = 1 → Busy Forced to 1 (default during installation) bit 1 = 1 → Data bus extern chip VME (vmeconfig) bit 2 = 1 → Quick answer 2 clock (vmeconfig) bit 3 = 1 → Quick answer 4 clock (vmeconfig)

Return values

If no errors occur ForceBusy( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.5 Class OC

FILES

RODOc.h

RODOc.cc

PUBLIC VARIABLES MEMBERS

NAME	DESCRIPTION
int id	Output Controller FPGA identifier. It can be 0, 1, 2 or 3. It must be set in order to access to the registers of the corresponding Output Controller FPGA.
VMEMasterMap* p_My_Vmm	Pointer to an object of the VMEMasterMap classs.

The following methods will be applied to the corresponding Output Controller FPGA, selected by the ***id*** public variable. If the user does not set the ***id*** variable, the method will return ROD_FAIL_OC.

PUBLIC METHODS MEMBERS

8.5.1 OC ( )

Description

Constructor of the class, it reserves memory space for private pointers.

8.5.2 ~OC ( )

Description

Destructor of the class. It frees the memory space reserved for private pointers.

8.5.3 ROD_Error WriteDummy (u_int data, int mode)

Description

This method writes in a Dummy register of the selected Output Controller FPGA.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written in a Dummy register of the selected Output Controller FPGA. Depending on the setting of the variable class member **id** it will access to one Output Controller or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteDummy( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.5.4 ROD_Error ReadDummy (u_int* value, int mode)

Description

This Method reads in a Dummy register of the selected Output Controller FPGA.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from a Dummy register of the selected Output Controller FPGA.

Return values

If no errors occur ReadDummy( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.5.5 void PrintDummy (u_int value)

Description

This method decodes and prints to the standard output the information read in the Dummy register of the selected Output Controller FPGA.

Input values

NAME	DESCRIPTION
u_int identif	Word read from a Dummy register of the selected Output Controller FPGA.

8.5.6 ROD_Error Reset(u_int data, int mode)

Description

This method resets the selected Output Controller FPGA.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written in the Reset register of the selected Output Controller FPGA. Depending on the setting of the variable class member **id** it will access to one Output Controller or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur Reset( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.5.7 ROD_Error ReadStatus (u_int* value, int mode)

Description

This method reads in the Status register of the selected Output Controller FPGA.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from the Status register of the selected Out Controller FPGA.

Return values

If no errors occur ReadStatus( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.5.8 void PrintStatus (u_int value)

Description

This method decodes and prints to the standard output the information read in the Status register of the selected Output Controller FPGA.

Input values

NAME	DESCRIPTION
u_int value	Word read from the Status register of the selected Output Controller FPGA.

8.5.9 ROD_Error WriteConfig (u_int data, int mode)

Description

This method writes in the Configuration register of the selected Output Controller FPGA.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.
u_int data	32-bit word to be written in the Configuration register of the selected Output Controller FPGA. Depending on the setting of the variable class member **id** it will access to one Output Controller or another.

Return values

If no errors occur WriteConfig( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.5.10 ROD_Error ReadConfig (u_int* value, int mode)

Description

This method reads in the Configuration register of the selected Output Controller FPGA.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from the Configuration register of the selected Output Controller FPGA.

Return values

If no errors occur ReadConfig( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.5.11 void PrintConfig (u_int value)

Description

This method decodes and prints to the standard output the information read in the Configuration register of the selected Output Controller FPGA.

Input values

NAME	DESCRIPTION
u_int value	Word read from the Configuration register of the selected Output Controller FPGA.

8.5.12 ROD_Error ReadSdram (u_int* value, int mode)

Description

Reads the Output Controller SDram register.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	32-bit word read from the SDram register of the selected Output Controller FPGA.

Return values

If no errors occur ReadSdram( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.5.13 ROD_Error ReadVersion(u_int* value, int mode)

Description

This method reads in the Version register of the selected Output Controller FPGA.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from the Version register of the selected Output Controller FPGA.

Return values

If no errors occur ReadVersion( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.5.14 void PrintVersion (u_int value)

Description

This method decodes and prints to the standard output the information read in the Version register of the selected Output Controller FPGA.

Input values

NAME	DESCRIPTION
u_int value	Word read from the Version register of the selected Output Controller FPGA.

8.6 Class BUSY

FILES

RODBusy.h

RODBusy.cc

PUBLIC VARIABLES MEMBERS

NAME	DESCRIPTION
VMEMasterMap* p_My_Vmm	Pointer to an object of the VMEMasterMap classs.

PUBLIC METHODS MEMBERS

8.6.1 BUSY ( )

Description

Constructor of the class, it reserves memory space for private pointers.

8.6.2 ~BUSY ( )

Description

Destructor of the class. It frees the memory space reserved for private pointers.

8.6.3 ROD_Error ReadStatus (u_int* value, int mode)

Description

This method reads in the Busy Status register of the ROD module.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from the Busy Status register of the ROD module.

Return values

If no errors occur ReadStatus( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.6.4 void PrintStatus (u_int status)

Description

This method decodes and prints to the standard output the information read in the Busy Status register of the ROD module.

Input values

NAME	DESCRIPTION
u_int value	Word read from the Busy Status register of the ROD module.

8.6.5 ROD_Error WriteResetControl (u_int data, int mode)

Description

This method writes in the Busy Reset and Control register of the ROD module.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.
u_int data	32-bit word to be written in the Busy Reset and Control register of the ROD module. bit 0 = 1 → Reset fifo bit 1 = 1 → Reset busy duration counter bit 2 = 1 → Reset Sreq counter bit 4 = 1 → Reset timing: reset interval, duration and clock divider counters bit 8 = 1 → Write busy duration counter to fifo bit 16 = 1 → Send sreq

Return values

If no errors occur ReadBaseAdd( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.6.6 ROD_Error ReadMisc (u_int* value, int mode)

Description

This method reads the Busy Miscellaneous register of the ROD module.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from the Busy Miscellaneous register of the ROD module.

Return values

If no errors occur ReadMisc( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.6.7 void PrintMisc (u_int value)

Description

This method decodes and prints to the standard output the information read in the Busy Miscellaneous register of the ROD module.

Input values

NAME	DESCRIPTION
u_int value	Word read from the Busy Miscellaneous register of the ROD module.

8.6.8 ROD_Error WriteMisc (u_int data, int mode)

Description

This method writes in the Busy Miscellaneous register of the ROD module.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.
u_int data	32-bit word to be written in the Busy Miscellaneous register. bit 0 = 0 → Busy source VME bit 0 = 1 → Busy source PU bit 1 = 0 → Manual mode bit 1 = 1 → Normal mode bit 9 = 1 → Enable busy bit 12 = 1 → Enable sreq bit 23:16 → Mask busy (pu4_busy2 down to pu1_bus1)

Return values

If no errors occur WriteMisc( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.6.9 ROD_Error ReadIntervFifo (u_int* value, int mode)

Description

This method reads in the Busy Interval Fifo register of the ROD module.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from the Busy Interval Fifo register of ROD module.

Return values

If no errors occur ReadIntervFifo( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.6.10 ROD_Error WriteIntervFifo (u_int data, int mode)

Description

This method writes in the Busy Interval Fifo register of the ROD module.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.
u_int data	32-bit word to be written in the Busy Interval register of ROD module.

Return values

If no errors occur WriteIntervFifo( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.6.11 ROD_Error ReadSreqMax (u_int* value, int mode)

Description

This method reads in the Busy Sreq Maximum register of the ROD module. This register contains the maximum number of busies before send a Sreq.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from the Busy Sreq Maximum register of ROD module.

Return values

If no errors occur ReadSreqMax( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.6.12 ROD_Error WriteSreqMax (u_int data, int mode)

Description

This method writes in the Busy Sreq Maximum register of the ROD module.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.
u_int data	32-bit word to be written in the Busy Sreq maximum register of ROD module. Number of busy counts before send a IRQ

Return values

If no errors occur WriteSreqMax( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.6.13 ROD_Error ReadDivClock (u_int* value, int mode)

Description

This method reads in the Busy Divider Clock register.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from the Busy Sreq Maximum register of the ROD module.

Return values

If no errors occur ReadDivClock( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.6.14 ROD_Error WriteDivClock (u_int data, int mode)

Description

This method writes in the Busy Divider Clock register of the ROD module.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.
u_int data	32-bit word to be written in the Busy Divider Clock register of the ROD module. bit 7:0 $\rightarrow$ busy count each (100 nS + 25nS $\times$ (n - 1))

Return values

If no errors occur WriteDivClock( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.6.15 ROD_Error ReadFifo (u_int* value, int mode)

Description

This method reads in the Busy Fifo register of the ROD module.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from the Busy Fifo register of the ROD module.

Return values

If no errors occur ReadFifo( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.6.16 void PrintFifo(u_int value)

Description

This method decodes and prints to the standard output the information read in the Busy Fifo register of the ROD module.

Input values

NAME	DESCRIPTION
u_int value	Word read from the Busy Fifo register of the ROD module.

8.6.17 ROD_Error ReadDurationBusy (u_int* value, int mode)

Description

This method reads in the Busy Duration Busy register of the ROD module.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from the Busy Duration register of ROD module.

Return values

If no errors occur ReadDurationBusy( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.6.18 void PrintDurationBusy (u_int value)

Description

This method decodes and prints to the standard output the information read in the Busy Duration register of the ROD module.

Input values

NAME	DESCRIPTION
u_int value	Word read from the Busy Duration register of ROD module.

8.6.19 ROD_Error ReadSreqCounter (u_int* value, int mode)

Description

This method reads in the Busy Sreq Counter register of the ROD module.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from the Busy Sreq Counter register of ROD module.

Return values

If no errors occur ReadDurationBusy( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.6.20 void PrintSreqCounter (u_int value)

Description

This method decodes and prints to the standard output the information read in the Busy Sreq Counter register of the ROD module.

Input values

NAME	DESCRIPTION
u_int value	Word read from the Busy Sreq Counter register of ROD module.

8.6.21 ROD_Error SendBusy (int mode)

Description

This method writes in the Busy Send Busy register of the ROD module.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur SendBusy( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.7 Class TTC

FILES

RODTtc.h

RODTtc.cc

PUBLIC VARIABLES MEMBERS

NAME	DESCRIPTION
VMEMasterMap* p_My_Vmm	Pointer to an object of the VMEMasterMap class.

PUBLIC METHODS MEMBERS

8.7.1 TTC ( )

Description

Constructor of the class, it reserves memory space for private pointers.

8.7.2 ~TTC ( )

Description

Destructor of the class. It frees the memory space reserved for private pointers.

8.7.3 ROD_Error ReadStatus (u_int* value, int mode)

Description

This method reads in the TTC Status register of the ROD module.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from the TTC Status register of the ROD module.

Return values

If no errors occur ReadStatus( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.7.4 void PrintStatus (u_int value)

Description

This method decodes and prints to the standard output the information read in the TTC Status register of the ROD module.

Input values

NAME	DESCRIPTION
u_int value	Word read from the TTC Status register of the ROD module.

8.7.5 ROD_Error WriteTtype (u_int data, int mode)

Description

This method writes in the TTC Type register of the ROD module.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.
u_int data	32-bit word to be written in the TTC Type register of the ROD module. bit 7:0 → TTC type

Return values

If no errors occur WriteTtype( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.7.6 ROD_Error WriteEVID (u_int data, int mode)

Description

This method writes in the TTC Event ID register of the ROD module.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.
u_int data	32-bit word to be written in the TTC Event ID register of ROD module. bit 31:0 → TTC Event ID

Return values

If no errors occur WriteEVID( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.7.7 ROD_Error WriteBCID (u_int data, int mode)

Description

This method writes in the TTC Bunch Crossing ID register of ROD module.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.
u_int data	32-bit word to be written in the TTC Bunch Crossing ID register of the ROD module. bit 11:0 → TTC Bunch Crossing ID

Return values

If no errors occur WriteBCID( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.7.8 ROD_Error WriteControl (u_int data, int mode)

Description

This method writes in the TTC Control register of the ROD module.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.
u_int data	32-bit word to be written in the TTC Cotnrol ID register of ROD modules. bit 2:0 → Mode , Only one bit can be set to 1: bit 0 = 1 → Mode VME bit 1 = 1 → Mode Local bit 2 = 1 → Mode TTC bit 3 = 0 → Local clock selection bit 3 = 1 → TTCrx clock selection bit 4 = 1 → L1A mode double pulse bit 7:5 → Pulse bit 5 = 1 → Flush bit 6 = 1 → Clear Status bit 7 = 1 → TTC reset

Return values

If no errors occur WriteControl( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.7.9 ROD_Error ReadControl (u_int* value, int mode)

Description

This method reads in the TTC Control register of the ROD module.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from the TTC Control register of ROD module.

Return values

If no errors occur ReadControl( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.7.10 void PrintControl (u_int value)

Description

This method decodes and prints to the standard output the information read in the TTC Control register of the ROD module.

Input values

NAME	DESCRIPTION
u_int value	Word read from the TTC Control register of ROD module.

8.7.11 ROD_Error ReadDummy (u_int* value, int mode)

Description

This method reads in the TTC Dummy register of the ROD module.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from the TTC Dummy register of ROD module.

Return values

If no errors occur ReadDummy( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.7.12 ROD_Error SendTTCEvent (u_int TTYPE, u_int EVID, u_int BCID, int mode)

Description

This method writes in the TTC Type, Event ID and Bunch Crossing registers of the ROD module.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.
u_int BCID	32-bit word to be written in the TTC Bunch Crossing ID register.

u_int EVID	32-bit word to be written in the TTC Event ID register.
u_int TTYPE	32-bit word to be written in the TTC Type register.

Return values

If no errors occur SendTTCEvent( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.8 Class STAGING

FILES

RODStag.h

RODStag.cc

PUBLIC VARIABLES MEMBERS

NAME	DESCRIPTION
int id	Staging FPGA identifier. It can be 0, 1, 2, or 3. It must be set in order to access to the registers of the corresponding staging FPGA.
VMEMasterMap* p_My_Vmm	Pointer to an object of the VMEMasterMap classs.

The following methods will be applied to the corresponding staging FPGA, selected by the ***id*** public variable. If the user does not set the ***id*** variable, the method will return ROD_FAIL_STAG.

PUBLIC METHODS MEMBERS

8.8.1 STAGING ( )

Description

Constructor of the class, it reserves memory space for private pointers.

8.8.2 ~STAGING ( )

Description

Destructor of the class. It frees the memory space reserved for private pointers.

8.8.3 ROD_Error WriteDummy (u_int data, int mode)

Description

This method writes in a Dummy register of the selected Staging FPGA.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written in a Dummy register of the selected Staging FPGA. Depending on the setting of the variable class member **id** it will access to one Staging FPGA or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteDummy( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.8.4 ROD_Error ReadDummy (u_int* value, int mode)

Description

This method reads in the Dummy register of the selected Staging FPGA.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from a Dummy register of the selected Staging FPGA.

Return values

If no errors occur ReadDummy( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.8.5 void PrintDummy (u_int value)

Description

This method decodes and prints to the standard output the information read in the Dummy register of the selected Staging FPGA.

Input values

NAME	DESCRIPTION
u_int value	Word read from a Dummy register of the selected Staging FPGA.

8.8.6 ROD_Error WriteSetReset (u_int data, int mode)

Description

This method sets and resets the selected Staging FPGA.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to set and/or reset the selected Staging FPGA. Depending on the setting of the variable class member **id** it will access to one Staging FPGA or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteSetReset( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.8.7 ROD_Error ReadStatus (u_int* value, int mode)

Description

This method reads in the Status register of the selected Staging FPGA.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from a Status register of the selected Staging FPGA.

Return values

If no errors occur ReadStatus( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.8.8 void PrintStatus (u_int value)

Description

This method decodes and prints to the standard output the information read in the Status register of the selected Staging FPGA.

Input values

NAME	DESCRIPTION
u_int value	Word read from a Status register of the selected Staging FPGA.

8.8.9 ROD_Error WriteConfig1(u_int data, int mode)

Description

This method writes in the Configuration register 1 of the selected Staging FPGA.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written in the Configuration register 1 of the selected Staging FPGA. Depending on the setting of the variable class member **id** it will access to one Staging FPGA or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteConfig1( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.8.10 ROD_Error ReadConfig1 (u_int* value, int mode)

Description

This method reads in Configuration register 1 of the selected Staging FPGA.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from the Configuration register 1 of the selected Staging FPGA.

Return values

If no errors occur ReadConfig1( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.8.11 void PrintConfig1 (u_int value)

Description

This method decodes and prints to the standard output the information read in the Configuration register 1 of the selected Staging FPGA.

Input values

NAME	DESCRIPTION
u_int value	Word read from the Configuration register 1 of the selected Staging FPGA.

8.8.12 ROD_Error WriteConfig2(u_int data, int mode)

Description

This method writes in the Configuration register 2 of the selected Staging FPGA.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written in the Configuration register 2 of the selected Staging FPGA. Depending on the setting of the variable class member **id** it will access to one Staging FPGA or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteConfig2( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.8.13 ROD_Error ReadConfig2 (u_int* value, int mode)

Description

This method reads in the Configuration register 2 of the selected Staging FPGA.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from the Configuration register 2 of the selected Staging FPGA.

Return values

If no errors occur ReadConfig2( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.8.14 void PrintConfig2 (u_int value)

Description

This method decodes and prints to the standard output the information read in the Configuration register 2 of the selected Staging FPGA.

Input values

NAME	DESCRIPTION
u_int value	Word read from the Configuration register 2 of the selected Staging FPGA.

8.8.15 ROD_Error WriteStartAdd(u_int data, int mode)

Description

This method writes the initial address where data is sent to the RAM of the selected Staging FPGA. This version only allows 512 words, so, address should be written in a word of 9 bits (8:0).

Input values

NAME	DESCRIPTION
u_int data	9 bit addressed can be written for setting the initial address of the RAM of the selected Staging FPGA. Depending on the setting of the variable class member **id** it will access to one Staging FPGA or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteStartAdd( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.8.16 ROD_Error ReadStartAdd(u_int* value, int mode)

Description

Method to read the initial address of the RAM of the selected Staging FPGA Configuration register 1.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	32-bit word with the information of the initial address of the RAM of the selected Staging FPGA.

Return values

If no errors occur ReadStartAdd( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.8.17 void PrintStartAdd (u_int value)

Description

This method prints to the standard output the initial address of the RAM of the selected Staging FPGA.

Input values

NAME	DESCRIPTION
u_int value	Word read from the Start Address register of the selected Staging FPGA.

8.8.18 ROD_Error WriteDataRam(u_int data, int mode)

Description

This method writes a word in the memory RAM of the selected Staging FPGA.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written in memory RAM of the selected Staging FPGA. Depending on the setting of the variable class member **id** it will access to one Staging FPGA or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteDataRam( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.8.19 ROD_Error ReadDataRam (u_int* value, int mode)

Description

This method reads in the memory RAM of the selected Staging FPGA.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from memory RAM of the selected Staging FPGA.

Return values

If no errors occur ReadDataRam( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.8.20 void PrintDataRam(u_int value)

Description

This method decodes and prints to the standard output the information read in the memory RAM of the selected Staging FPGA.

Input values

NAME	DESCRIPTION
u_int value	Word read from the memory RAM of the selected Staging FPGA.

8.8.21 ROD_Error StartTransmission(int mode)

Description

This method starts the transmission in the selected Staging FPGA.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur StartTransmission( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.8.22 ROD_Error WriteLinkConfig(u_int data, int mode)

Description

This method writes the link configuration of the selected Staging FPGA.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written in the Link Configuration of the selected Staging FPGA. Depending on the setting of the variable class member **id** it will access to one Staging FPGA or another.

int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteLinkConfig( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.8.23 ROD_Error ReadLinkConfig (u_int* value, int mode)

Description

This method reads the link configuration of the selected Staging FPGA.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from memory RAM of the selected Staging FPGA.

Return values

If no errors occur ReadLinkConfig( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.8.24 void PrintLinkConfig(u_int value)

Description

This method decodes and prints to the standard output the information read in Link Configuration register of the selected Staging FPGA.

Input values

NAME	DESCRIPTION
u_int value	Word read from the Link Configuration register of the selected Staging FPGA.

8.8.25 ROD_Error ReadTemperature(int num, int* value, int mode)

Description

This method reads the Temperature of the selected Staging FPGA. With the input parameter ***num***, the user can read up to 8 temperature values from the 8 G-links of the ROD module. Never mind through which Staging FPGA the user reads the temperature, because this method always accesses the same registers.

Input values

NAME	DESCRIPTION
int num	Staging FPGA temperature identifier. This parameter can take values from '0' to '7' for reading the temperature of the eight G-links of the ROD module.

int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word for reading the Temperature of the selected Staging FPGA.

Return values

If no errors occur ReadTemperature( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.8.26 void PrintTemperature(u_int value)

Description

This method decodes and prints to the standard output the information of the Temperature of the selected Staging FPGA.

Input values

NAME	DESCRIPTION
u_int value	Word read from the Temperature register of the selected Staging FPGA. When it is decoded the user will read the maximum, the minimum and the actual temperature.

8.8.27 ROD_Error ReadVersion(int* value, int mode)

Description

This method reads the Version of the selected Staging FPGA.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read for reading the version of the selected Staging FPGA.

Return values

If no errors occur ReadVersion( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.8.28 void PrintVersion(u_int value)

Description

This method decodes and prints to the standard output the information of the Version of the selected Staging FPGA.

Input values

NAME	DESCRIPTION
u_int value	Word read from the Version register of the selected Staging FPGA.

8.9 Class FPGA_PU

FILES

RODFpga_pu.h

RODFpga_pu.cc

PUBLIC VARIABLES MEMBERS

NAME	DESCRIPTION
int id	Dummy/FPGA Processing Unit Identifier. It can be 0, 1, 2, or 3. It must be set in order to access to the registers of the corresponding Dummy/FPGA Processing Unit.
VMEMasterMap* p_My_Vmm	Pointer to an object of the VMEMasterMap classs.

The following methods will be applied to the corresponding FPGA Processing Unit, selected by the ***id*** public variable. If the user does not set the ***id*** variable, the method will return ROD_FAIL_FPGA_PU.

PUBLIC METHODS MEMBERS

8.9.1 FPGA_PU ( )

Description

Constructor of the class, it reserves memory space for private pointers.

8.9.2 ~FPGA_PU ( )

Description

Destructor of the class. It frees the memory space reserved for private pointers.

8.9.3 ROD_Error WriteDummy (u_int data, int mode)

Description

This method writes in a Dummy register of the selected Dummy/FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written in a Dummy register of the selected FPGA Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteDummy( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.4 ROD_Error ReadDummy (u_int* value, int mode)

Description

This method reads in the Dummy register of the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from a Dummy register of the selected FPGA Processing Unit.

Return values

If no errors occur ReadDummy( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.5 void PrintDummy (u_int value)

Description

This method decodes and prints to the standard output the information read in the Dummy register of the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word read from a Dummy register of the selected FPGA Processing Unit.

8.9.6 ROD_Error Reset (u_int data, int mode)

Description

This method sends a reset to the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written in a Reset register of the selected FPGA Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur Reset( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.7 ROD_Error ReadStatus (u_int* value, int mode)

Description

This method reads in the Status register of the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from a Status register of the selected FPGA Processing Unit.

Return values

If no errors occur ReadStatus( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.8 void PrintStatus (u_int value)

Description

This method decodes and prints to the standard output the information read in the Status register of the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word read from a Status register of the selected FPGA Processing Unit.

8.9.9 ROD_Error WriteConfig (u_int data, int mode)

Description

This method writes in a Configuration register of the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written in a Configuration register of the selected FPGA Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteConfig( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.10 ROD_Error ReadConfig (u_int* value, int mode)

Description

This method read in the Configuration register of the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from a Configuration register of the selected FPGA Processing Unit.

Return values

If no errors occur ReadConfig( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.11 void PrintConfig (u_int value)

Description

This method decodes and prints to the standard output the information read in the Configuration register of the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word read from a Configuration register of the selected FPGA Processing Unit.

8.9.12 ROD_Error ReadFifo (u_int* value, int mode)

Description

This method reads the FIFO the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from the FIFO of the selected FPGA Processing Unit.

Return values

If no errors occur ReadFifo( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.13 void PrintFifo (u_int value)

Description

This method decodes and prints to the standard output the information read from the FIFO of the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word read from the FIFO of the selected FPGA Processing Unit.

8.9.14 ROD_Error WriteSetResetBusy (u_int data, int mode)

Description

This method writes in the Pulse and Set/Reset Busy register of the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written in the Pulse and Set/Reset Busy register of the selected FPGA Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteSetResetBusy( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.15 ROD_Error ReadSetResetBusy (u_int* value, int mode)

Description

This method reads in the Pulse and Set/Reset Busy register of the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read from the Pulse and Set/Reset Busy register of the selected FPGA Processing Unit.

Return values

If no errors occur ReadSetResetBusy( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.16 void PrintSetResetBusy (u_int value)

Description

This method decodes and prints to the standard output the information read in the Pulse and Set/Reset register of the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word read from the Pulse and Set/Reset Busy register of the selected FPGA Processing Unit.

8.9.17 ROD_Error WriteFormatVersionNumber(u_int data, int mode)

Description

This method writes the Format Version Number for the data in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written to set the Format Version Number of the data in the selected FPGA Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteFormatVersionNumber( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.18 ROD_Error ReadFormatVersionNumber (u_int* value, int mode)

Description

This method reads the Format Version Number of the data in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word with the Format Version Number the information of the data in the selected FPGA Processing Unit.

Return values

If no errors occur ReadFormatVersionNumber( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.19 void PrintFormatVersionNumber (u_int value)

Description

This method decodes and prints to the standard output the Format Version Number of the data in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word with the Format Version Number information of the data in the selected FPGA Processing Unit.

8.9.20 ROD_Error WriteSouceID _FEB1(u_int data, int mode)

Description

This method writes the Source Identifier for the data in the FEB 1 in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written to set the Source Identifier of the data in the FEB 1 in the selected FPGA Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteSourceID_FEB1( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.21 ROD_Error ReadSourceID_FEB1 (u_int* value, int mode)

Description

This method reads the Source Identifier of the data in the FEB 1 in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word with the Source Identifier information of the data in the FEB 1 in the selected FPGA Processing Unit.

Return values

If no errors occur ReadSourceID_FEB1( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.22 void PrintSourceID_FEB1(u_int value)

Description

This method decodes and prints to the standard output the Source Identifier of the data in the FEB 1 in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word with the Source Identifier information of the data in the FEB 1 in the selected FPGA Processing Unit.

8.9.23 ROD_Error WriteSouceID _FEB2(u_int data, int mode)

Description

This method writes the Source Identifier for the data in the FEB 2 in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written to set the Source Identifier of the data in the FEB 2 in the selected FPGA Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteSourceID_FEB2( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.24 ROD_Error ReadSourceID_FEB2 (u_int* value, int mode)

Description

This method reads the Source Identifier of the data in the FEB 2 in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word with the Source Identifier information of the data in the FEB 2 in the selected FPGA Processing Unit.

Return values

If no errors occur ReadSourceID_FEB2( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.25 void PrintSourceID_FEB2(u_int value)

Description

This method decodes and prints to the standard output the Source Identifier of the data in the FEB 2 in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word with the Source Identifier information of the data in the FEB 2 in the selected FPGA Processing Unit.

8.9.26 ROD_Error WriteRunNumber(u_int data, int mode)

Description

This method writes the Run Number for the data in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written to set the Run Number of the data in the selected FPGA Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteRunNumber( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.27 ROD_Error ReadRunNumber (u_int* value, int mode)

Description

The method reads the Run Number of the data in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word read the Run Number of the data in the selected FPGA Processing Unit.

Return values

If no errors occur ReadRunNumber( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.28 void PrintRunNumber (u_int value)

Description

This method decodes and prints to the standard output the Run Number of the data in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word with the Run Number information of the data in the selected FPGA Processing Unit.

8.9.29 ROD_Error WriteExtendedL1ID (u_int data, int mode)

Description

This method writes the Extended L1ID (Level 1 Identifier) for the data in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written to set the Extended L1ID (Level 1 Identifier) of the data in the selected FPGA Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteExtendedL1ID( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.30 ROD_Error ReadExtendedL1ID (u_int* value, int mode)

Description

This method reads the Extended L1ID (Level 1 Identifier) of the data in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word with the Extended L1ID (Level 1 Identifier) information of the data in the selected FPGA Processing Unit.

Return values

If no errors occur ReadExtededL1ID( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.31 void PrintExtendedL1ID (u_int value)

Description

This method decodes and prints to the standard output the Extended L1ID (Level 1 Identifier) of the data in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word with the Extended L1ID (Level 1 Identifier) information of the data in the selected FPGA Processing Unit.

8.9.32 ROD_Error WriteBCID_L1_Ttype (u_int data, int mode)

Description

This method writes the BCID (Bunch Crossing Identifier) and L1 Ttype (Level 1 Trigger type) for the data in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written to set the BCID and L1 Ttype of the data in the selected FPGA Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteBCID_L1_Ttype( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.33 ROD_Error ReadBCID_L1_Ttype (u_int* value, int mode)

Description

This method reads the BCID and L1 Ttype of the data in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word with the BCID and L1 Ttype information of the data in the selected FPGA Processing Unit.

Return values

If no errors occur ReadBCID_L1_Ttype( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.34 void PrintBCID_L1_Ttype (u_int value)

Description

This method decodes and prints to the standard output the BCID and L1 Ttype of the data in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word with the BCID and L1 Ttype information of the data in the selected FPGA Processing Unit.

8.9.35 ROD_Error WriteDetectorEventType (u_int data, int mode)

Description

This method writes the Detector Event Type for the data in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written to set the Detector Event Type of the data in the selected FPGA Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteDetectorEventType( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.36 ROD_Error ReadDetectorEventType (u_int* value, int mode)

Description

This method reads the Detector Event Type of the data in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word with the Detector Event Type information of the data in the selected FPGA Processing Unit.

Return values

If no errors occur ReadDetectorEventType( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.37 void PrintDetectorEventType (u_int value)

Description

This method decodes and prints to the standard output the Detector Event Type of the data in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word with the Detector Event Type information of the data in the selected FPGA Processing Unit.

8.9.38 ROD_Error WriteWordsPerEvent (u_int data, int mode)

Description

This method writes the number of words sent per event in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written to set the number of words sent per event in the selected FPGA Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteWordsPerEvent( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.39 ROD_Error ReadWordsPerEvent (u_int* value, int mode)

Description

This method reads the number of words sent per event in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word with the number of words sent per event in the selected FPGA Processing Unit.

Return values

If no errors occur ReadWordsPerEvent( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.40 void PrintWordsPerEvent (u_int value)

Description

This method decodes and prints to the standard output the number of words sent per event in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word with the number of words sent per event in the selected FPGA Processing Unit.

8.9.41 ROD_Error ReadEvents_FEB1 (u_int* value, int mode)

Description

This method reads the Events from the FEB 1 (Front-End Board) in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word with the Events from the FEB 1 in the selected FPGA Processing Unit.

Return values

If no errors occur ReadEvents_FEB1( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.42 void PrintEvents_FEB1 (u_int value)

Description

This method decodes and prints to the standard output the Events from the FEB 1 in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word with Events from the FEB 1 in the selected FPGA Processing Unit.

8.9.43 ROD_Error ReadEvents_FEB2 (u_int* value, int mode)

Description

This method reads the Events from the FEB 2 (Front-End Board) in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word with the Events from the FEB 2 in the selected FPGA Processing Unit.

Return values

If no errors occur ReadEvents_FEB2( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.44 void PrintEvents_FEB2 (u_int value)

Description

This method decodes and prints to the standard output the Events from the FEB 2 in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word with Events from the FEB 2 in the selected FPGA Processing Unit.

8.9.45 ROD_Error ReadLinkErrorCounterGLink1 (u_int* value, int mode)

Description

This method reads the number of times a link error signal was asserted in the G-Link 1 of the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word with the number of link errors asserted in the G-Link 1 of the selected FPGA Processing Unit.

Return values

If no errors occur ReadLinkErrorCounterGLink1( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.46 void PrintLinkErrorCounterGLink1 (u_int value)

Description

This method decodes and prints to the standard output the number of link errors asserted in the G-Link 1 of the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word with the number of link errors asserted in the G-Link 1 of the selected FPGA Processing Unit.

8.9.47 ROD_Error ReadLinkReadyCounterGLink1 (u_int* value, int mode)

Description

This method reads the number of times a link ready signal was lost in the G-Link 1 of the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word with the number of times a link ready signal was lost in the G-Link 1 of the selected FPGA Processing Unit.

Return values

If no errors occur ReadLinkReadyCounterGLink1( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.48 void PrintLinkReadyCounterGLink1 (u_int value)

Description

This method decodes and prints to the standard output the number of times a link ready signal was in the G-Link 1 of the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word with the number of times a link ready signal was lost in the G-Link 1 of the selected FPGA Processing Unit.

8.9.49 ROD_Error ReadLinkErrorCounterGLink2 (u_int* value, int mode)

Description

This method reads the number of times a link error signal was asserted in the G-Link 2 of the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word with the number of link errors asserted in the G-Link 2 of the selected FPGA Processing Unit.

Return values

If no errors occur ReadLinkErrorCounterGLink2( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.50 void PrintLinkErrorCounterGLink2 (u_int value)

Description

This method decodes and prints to the standard output the number of link errors asserted in the G-Link 2 of the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word with the number of link errors asserted in the G-Link 2 of the selected FPGA Processing Unit.

8.9.51 ROD_Error ReadLinkReadyCounterGLink2 (u_int* value, int mode)

Description

This method reads the number of times a link ready signal was lost in the G-Link 2 of the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word with the number of times a link ready signal was lost in the G-Link 2 of the selected FPGA Processing Unit.

Return values

If no errors occur ReadLinkReadyCounterGLink2( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.52 void PrintLinkReadyCounterGLink2 (u_int value)

Description

This method decodes and prints to the standard output the number of times a link ready signal was in the G-Link 2 of the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word with the number of times a link ready signal was lost in the G-Link 2 of the selected FPGA Processing Unit.

8.9.53 ROD_Error WriteSubFragmentID_FEB1 (u_int data, int mode)

Description

This method writes the Sub fragment Identifier in data from FEB 1 in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written to set the Sub fragment Identifier in data from FEB 1 in the selected FPGA Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteSubFragmentID_FEB1( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.54 ROD_Error ReadSubFragmentID_FEB1 (u_int* value, int mode)

Description

This method reads the Sub fragment Identifier in data from FEB 1 in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word with the Sub fragment Identifier in data from FEB 1 in the selected FPGA Processing Unit.

Return values

If no errors occur ReadSubFragmentID_FEB1( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.55 void PrintSubFragmentID_FEB1 (u_int value)

Description

This method decodes and prints to the standard output the Sub fragment Identifier in data from FEB 1 in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word with the Sub fragment Identifier in data from FEB 1 in the selected FPGA Processing Unit.

8.9.56 ROD_Error WriteSubFragmentID_FEB2 (u_int data, int mode)

Description

This method writes the Sub fragment Identifier in data from FEB 2 in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written to set the Sub fragment Identifier in data from FEB 2 in the selected FPGA Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteSubFragmentID_FEB2( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.57 ROD_Error ReadSubFragmentID_FEB2 (u_int* value, int mode)

Description

This method reads the Sub fragment Identifier in data from FEB 2 in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word with the Sub fragment Identifier in data from FEB 2 in the selected FPGA Processing Unit.

Return values

If no errors occur ReadSubFragmentID_FEB2( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.58 void PrintSubFragmentID_FEB2 (u_int value)

Description

This method decodes and prints to the standard output the Sub fragment Identifier in data from FEB 2 in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word with the Sub fragment Identifier in data from FEB 2 in the selected FPGA Processing Unit.

8.9.59 ROD_Error WriteVersion (u_int data, int mode)

Description

This method writes the Version in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written to set the Version in the selected FPGA Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteVersion( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.60 ROD_Error ReadVersion (u_int* value, int mode)

Description

This method reads the Version in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word with the Version in the selected FPGA Processing Unit.

Return values

If no errors occur ReadVersion( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.9.61 void PrintVersion (u_int value)

Description

This method decodes and prints to the standard output the Version in the selected FPGA Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word with the Version in the selected FPGA Processing Unit.

8.10 Class DSP_PU

FILES

RODDsp_pu.h

RODDsp_pu.cc

PUBLIC VARIABLES MEMBERS

NAME	DESCRIPTION
int id	DSP Processing Unit Identifier. It can be 0, 1, 2, or 3. It must be set in order to access to the registers of the corresponding DSP Processing Unit.
VMEMasterMap* p_My_Vmm	Pointer to an object of the VMEMasterMap classs.

The following methods will be applied to the corresponding DSP Processing Unit, selected by the ***id*** public variable. If the user does not set the ***id*** variable, the method will return ROD_FAIL_DSP_PU.

The DSP class is more complex than the other classes. Each PU contains two DSP processors, and two FPGA chips (called Input FPGA) that can be programmed through the VME interface. For this raison the DSP class incorporates methods for booting those chips and methods for accessing the PU registers (as the other classes). Most of the methods for booting the PU are in the private area of the class, and the user has only to run simple methods as Config( ) or StartSendData( ).

PUBLIC METHODS MEMBERS

8.10.1 DSP_PU ( )

Description

Constructor of the class, it reserves memory space for private pointers.

8.10.2 ~DSP_PU ( )

Description

Destructor of the class. It frees the memory space reserved for private pointers.

8.10.3 ROD_Error GeneralReset (int mode)

Description

This method resets all the components of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur GeneralReset( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.4 ROD_Error StartSendData (int mode)

Description

This method configures both DSPs of the selected PU to start send data.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur StartSendData( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.5 ROD_Error Config (int mode, int FlagInFpgaLoad, char* InFpgaFile, int FlagDspLoad, char* DspFile, int FlagDspLaunch)

Description

This method configures the selected DSP Processing Unit. Internally Config( ) calls several methods for booting the components of the processing units, as the DSP processors and the FPGA chips.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.
int FlagInFpgaLoad	INFPGA_FILE_1(1), INFPGA_FILE_2(2) and INFPGA_BROADCAST(0) The FPGA chips can be booted separately (INFPGA_FILE_1 and INFPGA_FILE_2) or at the same time with the same firmware (INFPGA_BROADCAST)
char* InFpgaFile	Name of the file to load in the FPGA chips
int FlagDspLoad	DSP_FILE_1(1), DSP_FILE_2(2) and DSP_FILE_BROADCAST(0) The DSP processors can be booted separately (DSP_FILE_1 and DSP_FILE_2) or at the same time with the same firmware (DSP_FILE_BROADCAST)
char* DspFile	Name of the file to load in the DSP chips
int FlagDspLaunch	DSP_LAUNCH_NOSTART(0) and DSP_LAUNCH_START(1) If FlagDspLaunch is set to DSP_LAUNCH_NOSTART the program is loaded into the DSP but it does not start running, otherwise if it is set to DSP_LAUNCH_START the program starts.

Return values

If no errors occur Config( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.6 ROD_Error ReadDspVersion1 (u_int* value, int mode)

Description

This method reads the Version of the code loaded in the DSP 1 of the selected Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word with the Version of the code loaded in the DSP 1 of the selected Processing Unit.

Return values

If no errors occur ReadDspVersion1( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.7 void PrintDspVersion1 (u_int value)

Description

This method decodes and prints to the standard output the Version of the code loaded in the DSP 1 of the selected Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word with the Version of the code loaded in the DSP 1 of the selected Processing Unit.

8.10.8 ROD_Error ReadDspVersion2 (u_int* value, int mode)

Description

This method reads the Version of the code loaded in the DSP 2 of the selected Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word with the Version of the code loaded in the DSP 2 of the selected Processing Unit.

Return values

If no errors occur ReadDspVersion2( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.9 void PrintDspVersion2 (u_int value)

Description

This method decodes and prints to the standard output the Version of the code loaded in the DSP 2 of the selected Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word with the Version of the code loaded in the DSP 2 of the selected Processing Unit.

8.10.10 ROD_Error WriteDummy1 (u_int data, int mode)

Description

This method writes in a Dummy register of the Output FPGA of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written in a Dummy register of the Output FPGA of the selected DSP Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteDummy1( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.11 ROD_Error ReadDummy1 (u_int* value, int mode)

Description

This method reads in a Dummy register of the Output FPGA of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word in a Dummy register of the Output FPGA of the selected DSP Processing Unit.

Return values

If no errors occur ReadDummy1( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.12 void PrintDummy1 (u_int value)

Description

This method decodes and prints to the standard output the word read in a Dummy register of the Output FPGA of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word read from a Dummy register of the Output FPGA of the selected DSP Processing Unit.

8.10.13 ROD_Error WriteDummy2 (u_int data, int mode)

Description

This method writes in a Dummy register of the Output FPGA of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written in a Dummy register of the Output FPGA of the selected DSP Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteDummy2( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.14 ROD_Error ReadDummy2 (u_int* value, int mode)

Description

This method reads in a Dummy register of the Output FPGA of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word in a Dummy register of the Output FPGA of the selected DSP Processing Unit.

Return values

If no errors occur ReadDummy2( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.15 void PrintDummy2 (u_int value)

Description

This method decodes and prints to the standard output the word read in a Dummy register of the Output FPGA of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word read from a Dummy register of the Output FPGA of the selected DSP Processing Unit.

8.10.16 ROD_Error WriteControl1 (u_int data, int mode)

Description

This method writes in the Control register of the part 1 of the selected DSP Processing Unit. By writting in this register a reset of the DSP 1, the Input FPGA 1 and FIFO 1 can be done.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written the Control register 1 of the selected DSP Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteControl1( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.17 ROD_Error ReadControl1 (u_int* value, int mode)

Description

This method reads in the Control register 1 of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word in the Control register 1 of the selected DSP Processing Unit.

Return values

If no errors occur ReadControl1( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.18 void PrintControl1 (u_int value)

Description

This method decodes and prints to the standard output the word read in the Control register 1 of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word read from the Control register 1 of the selected DSP Processing Unit.

8.10.19 ROD_Error WriteControl2 (u_int data, int mode)

Description

This method writes in the Control register of the part 2 of the selected DSP Processing Unit. By writting in this register a reset of the DSP 2, the Input FPGA 2 and FIFO 2 can be done.

Input values

NAME	DESCRIPTION
u_int data	32-bit word to be written the Control register 2 of the selected DSP Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteControl2( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.20 ROD_Error ReadControl2 (u_int* value, int mode)

Description

This method reads in the Control register 2 of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word in the Control register 2 of the selected DSP Processing Unit.

Return values

If no errors occur ReadControl2( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.21 void PrintControl2 (u_int value)

Description

This method decodes and prints to the standard output the word read in the Control register 2 of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word read from the Control register 2 of the selected DSP Processing Unit.

8.10.22 ROD_Error ReadStatus1 (u_int* value, int mode)

Description

This method reads in the Status register 1 of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word in the Status register 1 of the selected DSP Processing Unit.

Return values

If no errors occur ReadStatus1( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.23 void PrintStatus2 (u_int value)

Description

This method decodes and prints to the standard output the word read in the Status register 2 of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word read from the Status register 2 of the selected DSP Processing Unit.

8.10.24 ROD_Error ReadStatus2 (u_int* value, int mode)

Description

This method reads in the Status register 2 of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit word in the Status register 2 of the selected DSP Processing Unit.

Return values

If no errors occur ReadStatus2( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.25 void PrintStatus2 (u_int value)

Description

This method decodes and prints to the standard output the word read in the Status register 2 of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word read from the Status register 2 of the selected DSP Processing Unit.

8.10.26 ROD_Error WriteSerial1 (u_int data, int mode)

Description

This method writes Serial Data to McBSP2 (DSP1) of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
u_int data	32-bit data word written to the McBSP2 (DSP1) of the selected DSP Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteSerial1( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.27 ROD_Error ReadSerial1 (u_int* value, int mode)

Description

This method reads Serial Data from McBSP2 (DSP1) of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit data word from the McBSP2 (DSP1) of the selected DSP Processing Unit.

Return values

If no errors occur ReadSerial1( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.28 void PrintSerial1(u_int value)

Description

This method decodes and prints to the standard output the data from the McBSP2 (DSP1) of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word read from the McBSP1 (DSP1) of the selected DSP Processing Unit.

8.10.29 ROD_Error WriteSerial2 (u_int data, int mode)

Description

This method writes Serial Data to McBSP2 (DSP2) of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
u_int data	32-bit data word written to the McBSP2 (DSP2) of the selected DSP Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteSerial2( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.30 ROD_Error ReadSerial2 (u_int* value, int mode)

Description

This method reads Serial Data from McBSP2 (DSP2) of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit data word from the McBSP2 (DSP2) of the selected DSP Processing Unit.

Return values

If no errors occur ReadSerial2( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.31 void PrintSerial2(u_int value)

Description

This method decodes and prints to the standard output the data from the McBSP2 (DSP2) of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word read from the McBSP2 (DSP2) of the selected DSP Processing Unit.

8.10.32 ROD_Error WriteInFpgaConfig1 (u_int data, int mode)

Description

This method configures the Input FPGA 1 of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
u_int data	32-bit word, where only the 16 LSB are useful, for configuring the Input FPGA 1 of the selected DSP Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteInFpgaConfig1( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.33 ROD_Error WriteInFpgaConfig2 (u_int data, int mode)

Description

This method configures the Input FPGA 2 of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
u_int data	32-bit word, where only the 16 LSB are useful, for configuring the Input FPGA 2 of the selected DSP Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteInFpgaConfig2( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.34 ROD_Error WriteInFpgaProgram1 (u_int data, int mode)

Description

This method allows programming the Input FPGA 1 through VME of the selected DSP Processing Unit. The previous method Config( ) calls this method when booting the Input FPGA 1.

Input values

NAME	DESCRIPTION
u_int data	32-bit word for programming the Input FPGA 1 of the selected DSP Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteInFpgaProgram1( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.35 ROD_Error WriteInFpgaProgram2 (u_int data, int mode)

Description

This method allows programming the Input FPGA 2 through VME of the selected DSP Processing Unit. The previous method Config( ) calls this method when booting the Input FPGA 1.

Input values

NAME	DESCRIPTION
u_int data	32-bit word for programming the Input FPGA 2 of the selected DSP Processing Unit. Depending on the setting of the variable class member **id** it will access to one FPGA Processing Unit or another.
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Return values

If no errors occur WriteInFpgaProgram2( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.36 ROD_Error ReadOutFpgaVersion1 (u_int* value, int mode)

Description

This method reads the version of the firmware of the Output FPGA of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit data word from the Version register 1 of the selected DSP Processing Unit.

Return values

If no errors occur ReadOutFpgaVersion1( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.37 void PrintOutFpgaVersion1 (u_int value)

Description

This method decodes and prints to the standard output the version of the Output FPGA of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word read from the Version register 1 of the selected DSP Processing Unit.

8.10.38 ROD_Error ReadOutFpgaVersion2 (u_int* value, int mode)

Description

This method reads the version of the firmware of the Output FPGA of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
int mode	SAFE (0) and FAST (1) The FAST mode does not return any error message.

Output values

NAME	DESCRIPTION
u_int * value	Pointer to a 32-bit data word from the Version register 2 of the selected DSP Processing Unit.

Return values

If no errors occur ReadOutFpgaVersion2( ) returns ROD_SUCCESS otherwise returns the corresponding error code.

8.10.39 void PrintOutFpgaVersion2 (u_int value)

Description

This method decodes and prints to the standard output the version of the Output FPGA of the selected DSP Processing Unit.

Input values

NAME	DESCRIPTION
u_int value	Word read from the Version register 2 of the selected DSP Processing Unit.

9 APPENDIX III: ACRONYMS

ADC : Analogue to Digital Converter.

ALU : Arithmetic Logic Unit.

ATLAS : A Toroidal Lhc AparatuS.

BCID : Bunch Crossing IDentifier.

CERN : Centre Européen de Rechercher Nucléaire.

CIS : Charge Injection System.

CMT : Configuration Management Tool.

CPU : Central Processing Unit.

CTP : Central Trigger Processor.

DAQ : Data AcQuisition.

DSP : Digital Signal Processor.

FEB : Front End Board.

FF : Flat Filter.

FIT : Fit.

FIFO : First-In and First-Out memory.

FPGA : Field- Programmable Gate Arrays.

HV : High Voltage.

L1A : Level 1 trigger Accept

LHC : Large Hadron Collider.

LSC : Link Source Card.

MAC : Multiply Accumulative.

MIPS : Million Instructions Per Second.

OC : Output Controller.

OF : Optimal Filter.

ORX : Optical Receiver.

PMT : PhotoMul'Tiplier.

PLL : Phase-Locked Loop.

PU : Processing Unit.

RAM : Random Access Memory.

ROB : Read Out Buffer.

ROD : Read Out Driver.

SBC : Single Board Computer.

SSH : System Stack High.

TBM : Trigger and Busy Module.

TDAQ : Trigger and Data AcQuisition.

TM : Transition Module.

TTC : Timing, Trigger and Control.

Ttype : Trigger type.

VLIW : Very Long Instructions Word.

REFERENCES

[1] ATLAS Technical Proposal, CERN/LHCC/94-43 LHCC/P2, 15-12-1994.

[2] ATLAS Collaboration, Atlas Letter of Intent, CERN/LHCC/92-4, 1992.

[3] Tile Calorimeter Technical Design Report, CERN/LHCC 96-42, 15-12-1996.

[4] Digitizer Design Review.
Reference: http://www.physto.se/~ker/designreview/dr.html

[5] K. Anderson, A. Gupta, J. Pilcher, H. Sanders, F. Tang, R. Teuscher, H. Wu "ATLAS Tile Calorimeter Interface Card". LECC 2002 8th Workshop on Electronics for LHC Experiments [ISBN 92-9083-202-9].

[6] A. Blondel, D. La Marra, A. Léger, I. Riu, A. Straessner. "The ROD motherboard for the ATLAS Liquid Argon Calorimeter" University of Geneva.
Reference: http://dpnc.unige.ch/LArgROD/DocumentsMB.html

[7] RD12 - Timing, Trigger and Control (TTC) System for LHC Detectors.
Reference: http://ttc.web.cern.ch/TTC/intro.html

[8] VP-110/01x - 933 MHz or 800MHz Intel® Pentium® III Processor - Low Power based VME bard.
Reference: http://www.gocct.com/sheets/vp11001x.htm

[9] Model 965, 946, 1003, 993 & 983 Support Software, SBSTM Technologies.
Reference: http://www.hep.ph.ic.ac.uk/calice/elecPrototype/SBS620Manual.pdf

[10] CERN S-Link Homepage.
Reference: http://hsi.web.cern.ch/HSI/s-link

[11] The 3.3V Zero Delay Buffer CY2308 16-pin SOIC.
Reference: http://www.cypress.com/cfuploads/img/products/CY2308.pdf

[12] SDRAM in the ROD mother board.
Reference: http://download.micron.com/pdf/datasheets/dram/64MSDRAMx32_5.pdf

[13] Julie Prast, "The ATLAS Liquid Argon Calorimeters Read Out Dirver (ROD). The TMS320C6414 DSP Mezzanine board".
Referene: http://wwwlapp.in2p3.fr/atlas/Electronique/RODs/index.html

[14] José M. Castelo Currás, Research Report, "Data acquisition system (ROD) for the Tile hadronic Calorimeter. Testbeam Installation", Universitat de València.

[15] W. E Cleland and E. G. Stern. *NIM A338, 1994(467-497).*

[16] K. Anderson et al., "Bi-gain Front-end electronics for ATLAS Tile Calorimeter", *Workshop on electronics for LHC Experiments,* Rome 1998.

[17] E. Fulluna. "Energy reconstruction algorithms and their influence on the ATLAS Tile Calorimeter". *In Proceedings of the XI Interantional Conference on Calorimetry in High Energy Physics,* Perugia,2004.

[18] Texas Instruments, "TMS320C6000 CPU and Instruction Set Reference Guide", October 2000.

[19] E. Fullana. “Energy reconstruction with 7 vs 9 samples, impact on OF”, TileCal Analysis meeting, CERN, 23 February 2004.
Reference: http://agenda.cern.ch/fullAgenda.php?ida=a04565