The ALICE Level 0 Pixel Trigger Driver Layer

Cesar Torcato de Matos [a,b] Alex Kluge[a],Costanza Cavicchioli[a], Gianluca Aglieri Rinella[a], Giuseppe Marangio[c], Fernando Ribeiro[b], Michel Morel[a]

[a] CERN, 1211 Geneva 23, Switzerland
[b] University of Minho, Campus de Azurem, 4800 Guimaraes , Portugal
[c] INFN Sezione di Bari, Italy

cesar.matos@cern.ch

Abstract

The ALICE[1] Silicon Pixel Detector (SPD) [2] includes 120 detector modules each containing 10 pixel chips. Each pixel chip is capable of generating a FastOR signal indicating the presence of at least one pixel hit in the corresponding 8192 pixel matrix.

The Pixel Trigger (PIT) [3][4] System has been implemented to process the 1200 Fast-Or signals from the SPD and to provide an input signal to the ALICE Central Trigger Processor (CTP)[5] for the fastest (Level 0) trigger decision within a latency of 800 ns.

Working as a decision criteria for ALICE, the data flow need to be monitored carefully and status information needs to be made available. Therefore the PIT control system required an accurate design of hardware and software solutions to implement a coordinated operation of the PIT and the ALICE systems to which it interfaces.

A driver layer was developed under stringent requirements of robustness and reusability. It qualifies as a general purpose hardware driver for electronic systems. It uses the ALICE Digital Data Link (DDL) [6] front end board (SIU) to communicate with the PIT hardware.

We present here the design, and the implementation of the Pixel Trigger Front End Device (FED) Server [7].

I. The Pixel Trigger Control Hardware

The pixel trigger hardware is composed of a main processing board where 10 mezzanine cards (Optin boards) are plugged. The Optin boards are capable of reading the output of 12 SPD detector modules (Optical Links).

A PCI inspired control bus with transaction acknowledgement and late parity check is used to communicate between all on-board instances.

A DDL was included as the communication medium for the control interface. Commands are received and status information is read back via a bridge between this device and the internal control bus.

A second FPGA on the processing board is dedicated to the slow control, to the system interfaces and to the reconfiguration of the main processing FPGA. Status monitoring and control is implemented via registers in all the programmable devices available in the hardware. Remote programming of the processing FPGA is foreseen: the programming file for a given processing algorithm will be downloaded via the DDL link and the control FPGA to a local SRAM memory and then transferred to the Flash PROM connected to the processing FPGA.

II. The PIT Control System

The Pixel Trigger control system was designed to operate and control the pixel trigger hardware. It takes appropriate corrective actions to maintain the triggering stability and ensure the data quality.

It is composed of two computers: a Linux PC to act as the driver layer of the system using the ALICE Data Acquisition (DAQ) standard hardware equipped with a SIU/DDL module to interface with the PIT electronics, a Windows PC which is the supervision layer of the system running CERN's standard Supervisory Control and Data Acquisition (SCADA) framework, PVSS II [8].

The PIT control system is part of the ALICE Detector Control System (DCS) [9]. The architecture of the ALICE on-line software is strictly hierarchical. The main systems: DCS, DAQ, CTP and High Level Trigger (HLT) work as autonomous applications. At the highest level of this hierarchy is the Experimental Control System (ECS) [10], which controls all on-line applications.

The global ALICE DCS is partitioned in sub-detectors that are seen as independent control systems integrated in a hierarchical Finite State Machine (FSM) [11].

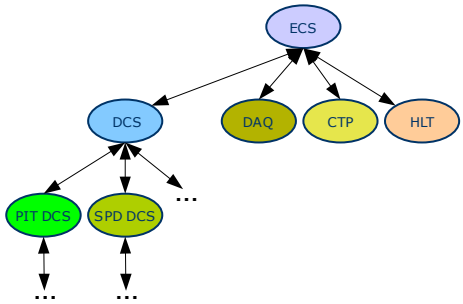

Figure 1: The model of Alice Detector Control Systems. Partitioning is based on sub-detectors. The ECS is the highest level instance with the control of all online applications.

III. The PIT FED Server

The Pixel trigger FED Server is a software developed to act as the driver layer of the system. It uses a SIU/DDL to

interface with the PIT electronics and it is capable of publishing status information and receiving commands from several computers over the network.

It was developed in C++ to provide an interface to all hardware features and to be the first layer of control of the system displaying the overall trigger status. It can freely access the 19 bit register address space in the hardware monitoring the status registers of the 120 optical links and 1200 FastOR counts and publishes them as DIM services for the supervision PVSS layer and Finite State Machine (FSM) for trigger quality calculation and FastOR tuning.

In addiction it publishes status services for alarm conditions, provides hardware debugging tools and automatic tests on the system ensuring the data quality.

A. Main Libraries Used

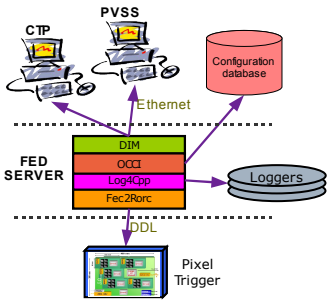

Figure 2: PIT FED Libraries diagram: Fec2Rorc is used to communicate with the DLL, Log4Cpp for logging, OCCI to interface with the configuration database and DIM to communicate with other software over the network.

The PIT FED server was developed on top of CERN standard libraries using:

DIM [12]: Developed at CERN stands for Distributed Information Management System. It provides a network transparent inter-process communication layer. It is used by the PIT FED to publish status information and to receive commands from other computers through the network.

OCCI [13]: Oracle C++ Call Interface (OCCI) is a high-performance and comprehensive object-oriented API to access the Oracle databases. It is used by the FED to access the DCS configuration database.

Log4cpp [14]: Is a library of C++ classes for flexible logging to files, syslog, IDSA and other destinations. It is modelled after the Log4j Java library, profiting of their API as much as possible. It is used for the extensive logging that exists in all operations of the FED.

Fec2Rorc [15]: Developed at CERN, part of the standard ALICE Data Acquisition and Test Environment (DATE) distribution, is a thin wrapper over the RORC driver libraries and is used by the lower level of the software to access the hardware using the DDL/SIU interface.

Figure 2 shows the library organization.

IV. The FED Server Command Structure

The PIT system was designed to receive commands from several ALICE sub-systems. There are 3 instances currently sending commands to the PIT FED server and relying on its status information: the PIT supervision layer for the control and operation of the system, the SPD FED server [16] in order to read status information for FastOR DAC calibration scans and the CTP in order to configure the PIT outputs (TINDET partition [17]). It can perform a wide range of operations: the execution status of these commands have to be available to all of the instances it interfaces with.

So for the PIT FED a command structure was devised to be flexible, capable of receiving a variable number of arguments, to accurately display the status of the execution of all commands and to be capable of dealing with the fact that several instances can send commands at the same time.

A command channel in the PIT FED is composed of a DIM command and 4 status informations brought through DIM services:

- Command: DIM command which is a white space separated string containing the command to be executed followed by all its parameters “command arg1 arg2..argN” ex: “write_register 0x18000 0xDEADBEEF”
- Command Status: DIM service with a string publishing the execution status of the command, possible values are: “FINISHED”, “EXECUTING”, “FAILED”
- Command Return: DIM service, an integer containing the return value of the command if any. If the command would be reading a register this service would contain, after finishing the command, the actual register value.
- Command ID: DIM service integer containing an unique id for the current command being executed, useful if several instances send commands at the same time

V. List Of Published Services

Connecting the PIT hardware to the rest of the world the PIT FED server has to work also as one information hub of the system.

Counters and status registers that are written by the PIT hardware are made available by issuing commands to the PIT FED but data that are relevant to the trigger quality and overall stability of the system need to be constantly monitored and to be forwarded through DIM services as PVSS datapoints that will be used by the supervision layer Finite State Machine (FSM). This is done by monitoring 2 sets of data: status of the links and status of the outputs.

A. Status of the links

There are 3 DIM services per optical link (link required, link status, link locked), making a total of 360 services. The data of these services are refreshed continuously from two set of registers in the Optin Board address space: the link status registers and the link settings register. There is one settings and one status register per optical link (120) in the system.

They are contained inside the Optin board address space. Registers 21 to 32 contain the link status registers and 33 to 44 the settings registers of the 12 optical links of the corresponding Optin board. Figure 3 and 4 show the format of these registers.

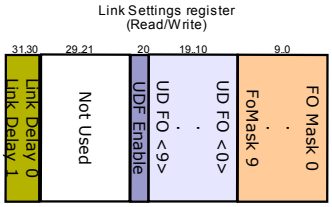

Figure 3: Link Settings register format.

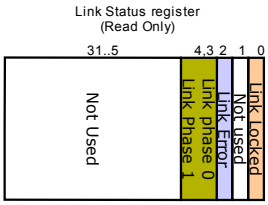

Figure 4: Link Status register format.

Link Required: integer service, 10 bits used for displaying which of the 10 FastOR channels from a link are being used in the trigger logic. It is read from the 10 least significant bits of the settings register, the mask bits, bits 0 to 9, one per FastOR chip. If one bit is enabled it means that the corresponding pixel chip will be excluded from the trigger logic. If all pixel chips are disabled the link is not required.

Link Locked: determines if the internal optical receiver is locked and getting a valid input data stream. It is read from the least significant bit, from the link status register that is forwarded by the deserializer device connected to the optical link receiver in the hardware. If a link is required by the trigger logic and there is no valid input data stream the data taking is stopped.

Link Error: determines if the rates of the FastOR of this channel are within the correct thresholds. In the hardware there are of 40 read/write registers where the maximum and minimum FastOR rates can be set: 20 for the maximum and minimum rates for pixels chips belonging to the inner layer of the SPD and 20 for the maximum and minimum rates for chips belonging to the outer layer. The Optin Board will then automatically count the incoming FastOR signals for a default period of one second and if at least one of the 10 FastOR channels of one link gets out of the thresholds the bit number 2 of the status register will be enabled stating that there is a configuration problem in this SPD halfstave (noise or inefficiency).

B. Status of the outputs

The same logic is applied to the outputs of the system. Status registers and counters of the outputs are read from the processing FPGA address space to the higher level supervision layer in order to detect problems and stop the run if required.

There are 10 services indicating the output mode to the CTP visualized as strings ("normal", "random", "toggle" or "signature"). The different modes are used for debugging of the CTP: normal is the normal triggering mode, in random mode the corresponding output will send a random bit pattern, in toggle mode the output will switch polarity on every trigger and in signature mode the output will send a pre-defined pattern on every trigger. During run mode all outputs have to be in “normal” mode.

10 integer services indicating the current trigger rate of the outputs are foreseen, this will be forwarded to the supervision layer in order to detect abnormalities in the trigger algorithms.

C. FastOR Counters

One big service displays the FastOR count of all chips in the system (1200). Used by the SPD FED and refreshed on demand during FastOR Optimization DAC scans. This service is published due to constraints in performance during FastOR calibration scans. The SPD FED will loop over pixel chip FastOR DACS, set test pulse matrices, send triggers and measure the FastOR rates. Due to the large number of possible steps of this scan (minimum of 4*8 bit DACs, 3 test pulse matrices : $3*256^4 = 12.9*10^9$). The SPD FED will send a command to start the measurements on the PIT FED and retrieve the data in one big array instead of issuing several read counters commands.

VI. Command Line Interface

Using the high level architecture of the FED server allows sending commands directly through a command line interface. This feature is managed by the pit_keyboard class. It uses the “termios.h” and “poll.h” unix C libraries to scan from the keyboard input without stopping the execution loop of the FED server. It is able to perform normal tasks and commands from other sources at the same time while the operator types commands in the PIT FED console.

The command structure is the same as for the DIM commands, feedback of the execution status is available by following one of the log channels of the FED.

The command line interface was extremely useful during the early commissioning phase for testing the hardware features and to perform bit error rate measurements on the system.

VII. The PIT FED Server Implementation

In order to address problems like the long term maintenance, robustness and scalability for the PIT FED server a pure C++ object-oriented paradigm modelling parts of the hardware was chosen. There was a heavy investment in encapsulation where classes provide interfaces hiding hardware related data while providing high level functionalities. C++ exceptions for error management and the Standard Template Library (STL) [18] whenever possible was used. This proved to be an effective way of managing complexity increasing code robustness without hindering performance.

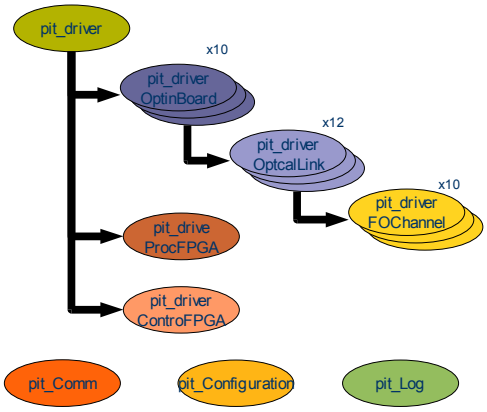

Figure 5: PIT FED class structure. Global pit_driver class containing collections of other driver classes. pit_comm and pit_configuration classes as singletons used by all other classes

A. Main Classes Developed

1. PIT Communication Class

The pit_comm class is positioned at the lower level of the class architecture. It is used to manage the communication through the DDL. It is a singleton [19] meaning that there is only one instance of the class in the project and its used by almost all other classes in this architecture. It is an object oriented wrapper around Fec2Rorc functions. Status bits are read automatically in all transactions. All data sent to the hardware are read back and verified. Exceptions are thrown if any error is detected.

The PIT communication class summary:

- manages the communication with the hardware
- a wrapper around the Fec2Rorc functions
- checks status bits automatically
- write followed by a read for verification
- one instance used by all hardware classes (singleton)
- throws exceptions on error detection

2. PIT Configuration Class

The pit_configuration class is designed to supply configuration data to all other classes. It manages the access to the oracle database or to configuration files. It is also a singleton having only one instance of this classe in the project supplying services to several other instances/classes.

The PIT configuration class summary:

- manages access to database or to configuration files
- one instance used by all classes (singleton)

3. PIT Driver Top Level Class

The pit_driver class is the higher level instance of all hardware related classes and contains collections of all other driver classes (processing FPGA, control FPGA, Optin boards etc.). It is capable of parsing the commands received from the DIM or from the command line interface and forwards them to the correct instances. This includes bit error rate tests, measurement and realignment of all phases, finding noisy FastOR channels and management of all global operations of the software.

The PIT Driver top level class summary:

- higher instance of the driver class
- has collections of all other driver classes (processing FPGA, control FPGA, Optin boards classes)
- manages commands coming from the different instances
- performs global operations in the system

4. Hardware modelling classes

The pit_driver_ProcFPGA, pit_driver_ControlFpga, pit_driver_OptinBoard, pit_driver_Optical_link and pit_driver_FoChannel are classes that were designed to model the existing hardware devices with the same name. They provide high level methods to access functionalities of the devices they represent hiding the hardware related implementation. They are self contained: they include inside them the means for communicating with the hardware and all memory addresses and information needed to keep track of the status of the device. They can publish status information via DIM services if needed. They are safe, they verify the data and throw exceptions if needed.

For instance the pit_driver_ProcFPGA class offers methods to fully configure the output algorithms of the system, the pit_driver_Optical_link class displays when a link is locked, is required or if it is in error.

PIT driver hardware classes summary:

- they model existing hardware devices with the same name
- they provide high level functionalities hiding memory addresses, registers and hardware related issues
- they are self contained
- they are included inside the pit_driver class
- they are safe: they verify the data and throw exceptions

B. Extension of Existing Frameworks

There was an extension of existing frameworks namely in Log4Cpp::DimAppender and run_time_error::pit_comm_error classes.

In the Log4Cpp framework a logger class can have several appenders and in this way it can write logging information to several destinations: log files, databases or even system specific logs ex: windows event log.

Log4Cpp::DimAppender is an extension of the generic Log4cpp::Appender. It was created through inheritance from the Log4cpp::FormatedAppender class. It publishes logging information to the higher instances that connect to the driver layer through DIM. This framework provides several priority levels for each log message (fatal, error, warning, info, debug etc.) and each appender can be configured to write messages only above a certain level. In that way the driver layer can be configured to be in debug level for the FileAppender object. Then all log messages are written to file, or to be in warning

level for the DimAppender object, displaying only warnings or errors to the operator in the supervision layer.

C. Command Parsing

The pit_driver class was designed to have a common path for the parsing of all commands from any source: the command line interface, DIM commands from the SPD FED server, DIM commands from the supervision layer etc. Like that a standard way of managing all commands with logging information and error handling can be created.

It uses only C++ formatted data input/output mechanisms relying in the istream and ostream inherited classes operators ">>" and "<<" so there is no possibility of buffer overruns while using sscanf or sprintf, increasing the stability and safety of the system.

The commands are seen as "white space" separated strings with a command name and a variable number or arguments. Furthermore the way of identifying instances in the hardware is done only by SPD coordinates: sector, side, channel and not by hardware coordinates, Optin number, channel. This makes the commands more easy to remember and more user friendly.

D. Error Handling

The error handling is a critical part of the development of any project. This software makes an extensive use of C++ exceptions for error handling. A special pit_comm_error class which is a extension of the run_time_error exception was developed. It is used by the system for special error handling of communication errors through the DDL. It allows to automatically reset the SIU on certain types of errors.

The verification of the data and the throwing of exceptions in case of errors was left to the lower level classes.

This increases:

- encapsulation: permits a bigger level of abstraction, the higher level instances classes do not need to be aware of the low level implementation of the system
- software flexibility: is easier to replace the lower level classes by other classes
- software scalability: is easy to add new kinds of errors after the initial design of the system
- safety: all errors will be reported and saved the same way. Even some lower level system errors included automatically by the compiler can be caught this way

E. Documentation Generating Tools

During the development of this software an automatic documentation generating tool called Doxygen [20] was used.

Doxygen is a documentation system for several languages including: C++, C, PVSS control scripts, VHDL, Python, Ruby and many more. It is used by a large number of projects, it can easily be integrated in many Integrated Development Environments (IDE) for instance eclipse. It can generate on-line documentation in HTML, MS-Word rich text format documents, Latex, compressed HTML, PostScript, PDF and Unix man pages. The documentation is extracted directly from the source files through specially defined comments and also by the code structure displaying the relationship between the various elements, dependency graphs, inheritance diagrams and collaboration diagrams which are all generated automatically.

This proved to be very useful to quickly find the way in large source distributions and increased good practices in commenting code and consistence in documentation.

VIII. Conclusions

The PIT FED server was integrated into ALICE DCS and is in stable production phase. It has been running stably without any crash for several months and it performes already many of the high level functions required. In particular the bit error rate measurements on the hardware and the automatic finding of noisy FastOR channels were extremely useful during the early commissioning phase as well as the command line interface.

The highly modular high level class structure proved to increase scalability and robustness, to reduce maintenance issues and does not hinder the performance of the system.

Automatic documentation tools and modern integrated development environments were used during the development phase.

The PIT FED successfully maps all PIT hardware features while publishing the status information for FastOR tuning. It calculates the trigger quality and provides the higher level FSM information to recognize errors conditions.

IX. References

[1] The ALICE Collaboration, K. Aamodt et al., The ALICE Experiment at the CERN LHC:
2008_JINST_3_S08002.

[2] P. Riedler et al., Overview and status of the ALICE silicon pixel detector:
Proceedings of the Pixel 2005 Conference, Bonn, Germany.

[3] G. Aglieri Rinella et al., The Level 0 Pixel Trigger system for the ALICE experiment:
2007 JINST 2 P0100

[4]Aglieri Rinella Gianluca et al., The Level 0 Pixel Trigger System for the ALICE Silicon Pixel Detector: implementation, testing and commissioning.
TWEPP-08 16 Sep 2008

[5] ALICE Collaboration, Technical design report, trigger, data acquisition, high level trigger, control system:
CERN-LHCC-2003-062, ALICE TDR 010, 7 January 2004.

[6] CERN ECP/ALD, RMKI RFFO, ALICE detector data link user requirements document:
Alice Internal Note, ALICE-INT-1996-042.

[7] Peter Chochula, Proposal for ALICE Front-end and Readout Electronics Monitoring and Configuration:
Alice DCS Development Note, Jan 14, 2003
http://alicedcs.web.cern.ch/alicedcs/Documents/FEEConfig.pdf

[8] ETM, PVSS:
http://www.etm.at/index_e.asp?id=2&m0id=6

[9] ALICE DCS:
http://alicedcs.web.cern.ch/alicedcs/

[10] ALICE Experiment Control System – ECS:
http://alice-ecs.web.cern.ch/alice-ecs/

[11] DCS Integration - FSM
http://alicedcs.web.cern.ch/AliceDCS/IntegrationDCS/IntegrationDCS.html

[12]Clara Gaspar, DIM;
http://dim.web.cern.ch/dim/

[13] OCCI, Oracle C++ Call Interface:
http://www.oracle.com/technology/tech/oci/occi/index.html

[14] Log4Cpp:
http://log4cpp.sourceforge.net/

[15] Rob Aberystwyth, Fec2Rorc:
http://www.brokenpipe.co.uk/fec2rorc.html

[16]Ivan Amos Cali,The ALICE Silicon Pixel Detector Control and Calibration Systems:
CERN-THESIS-2008-038

[17]Franco Carena, TINDET Partition:
Private Communication Dec-2006

[18]Standard Template Library Programmer's Guide:
http://www.sgi.com/tech/stl/

[19] Addison-Wesley,Design Patterns: Elements of Reusable Object-Oriented Software

[20]Doxygen:
http://www.stack.nl/~dimitri/doxygen/