Building Integrated Remote Control Systems for Electronics Boards

Richard Jacobsson, *Member, IEEE*

Abstract—**This paper addresses several aspects of implementing a remote control system for a large number of electronics boards in order to perform remote Field Programmable Gate Array (FPGA) programming, hardware configuration, data register access, and monitoring, as well as interfacing it to an expert system. The paper presents a common strategy for the representation of the boards in the abstraction layer of the control system, and generic communication protocols for the access to the board resources. In addition, an implementation is proposed in which the mapping between the functional parameters and the physical registers of the different boards is represented by descriptors in the board representation such that the translation can be handled automatically by a generic translation manager.**

Using the Distributed Information Management (DIM) package for the control communication with the boards, and the industry SCADA system PVSS II from ETM, a complete control system has been built for the Timing and Fast Control (TFC) system of the LHCb experiment at CERN. It has been in use during the entire prototyping of the TFC system and the developments of the LHCb sub-detector electronics, and is now installed in the online system of the final experiment.

***Index Terms*—Control communication protocol, control interface, control systems, FPGA, LHCB.**

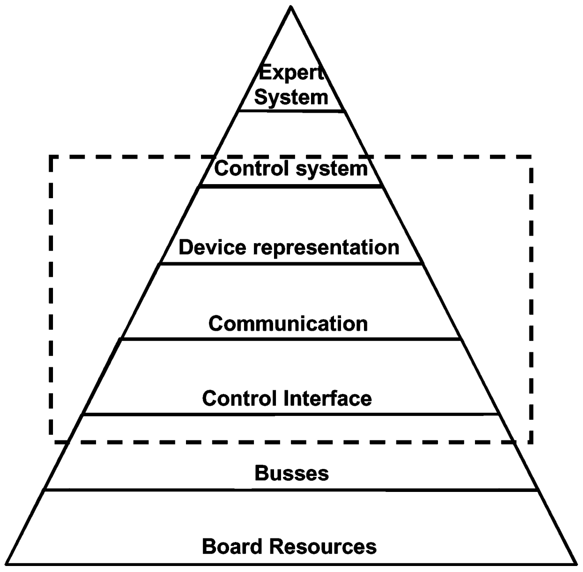


Fig. 1. This paper proposes generic solutions in the implementation of the middle layers of a complete control hierarchy with remote control by an expert system.

I. Introduction

In a system of many electronics boards of many different types which should be controlled from a remote supervisory expert system it is important to devise common and generic strategies all the way down to the board bus drivers which accesses the resources on the electronics boards. Fig. 1 shows a complete control hierarchy with the typical abstraction layers. This paper proposes generic solutions for the implementation of the middle layers between the access to board resources and the control actions of the supervisory control system.

Given that a set of electronics boards are controlled from an on-board controller which has access to a set of board control buses, the aim here is to:

- Define a generic data structure which allows representing the state of all control and monitoring resources of a board of any type, and which has interfaces to the communication protocol and to the supervisory control and expert system.
- Provide a simple and economical control communication protocol for the remote access mechanism to any board resource type independent of the bus type.
- Provide a simple and economical mechanism which allows monitoring counter and status information in the electronics boards by data register subscription.
- Provide a mechanism to verify the integrity of the data register access.
- Provide a mapping between the logical or functional view and the hardware view of the boards.
- Provide a set of functions by which the control system can perform control actions and retrieve status information through the functional view of the system, either directed from a user interface or from an expert system.

The developments have been done in the context of the Timing and Fast Control system (TFC) [1] of the LHCb experiment [2] at CERN. The LHCb experiment is located on the LHC accelerator and is a single-arm spectrometer composed of ten sub-detectors with the aim to study the large number of b-hadrons produced at the LHC.

In the LHCb experiment the electronics boards are controlled using the industrial distributed SCADA system PVSS II from ETM [3]. Fig. 2 shows an overview of the control system architecture of the LHCb Timing and Fast Control system. The TFC system is composed of some 75 boards of five different types. The most complex board, the LHCb Readout Supervisor, has over 300 functional parameters in over 200 physical registers and several data tables.

PVSS is centred on the powerful concept of an Event Manager which acts in between a Data Manager in association with

Manuscript received May 11, 2007; revised November 11, 2007.

The author is with CERN, CH-1211 Geneva 23, Switzerland (e-mail: richard.jacobsson@cern.ch).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNS.2007.914028

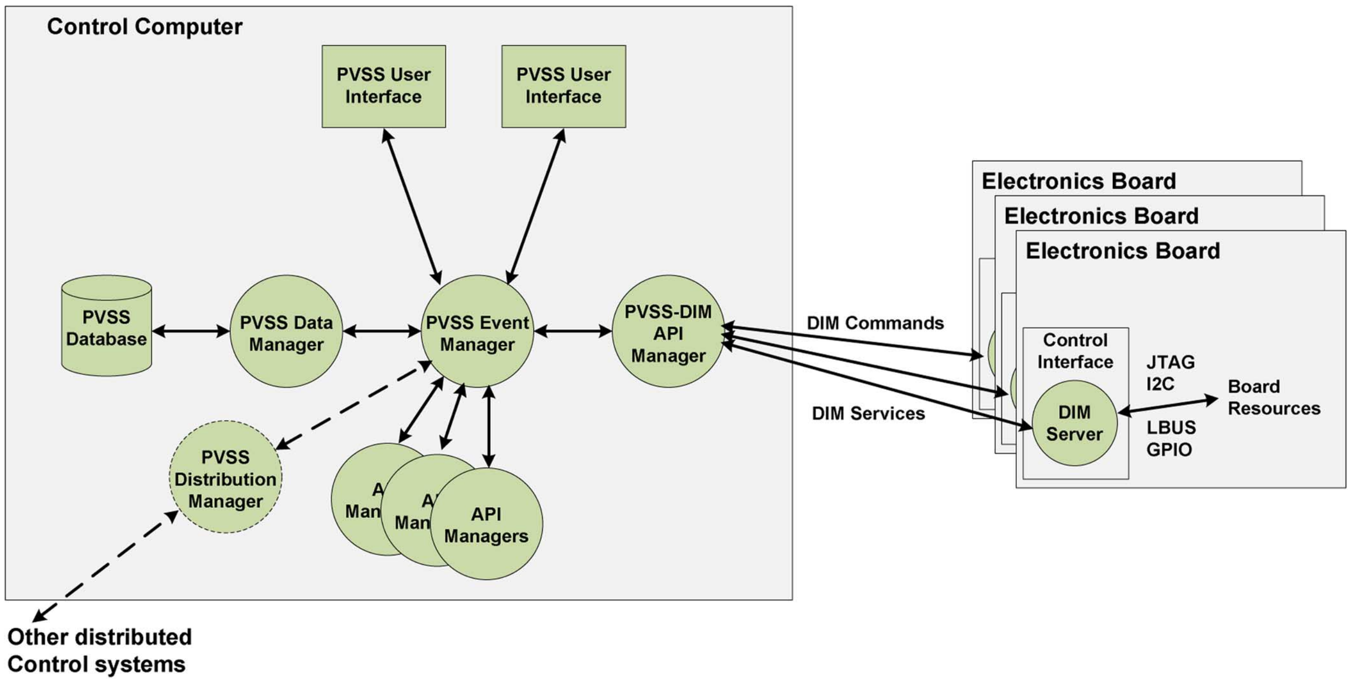


Fig. 2. The control system architecture of the Timing and Fast Control system of the LHCb experiment.

a high-speed database, User Interfaces, Control Scripts, and external systems. The external systems may be connected via API managers or drivers. The data management is based on the concept of data points which is analogous to dynamic structures of data to which event-triggered functions in PVSS scripts and PVSS API managers may be associated.

The concept of data points is very suitable to implement the dynamic representation and storage of the controllable resources of each electronics board.

The communication over the control network between the supervisory control system PVSS and the controller located on each electronics board is based on the Distributed Information Management (DIM) package [4]. As shown in Fig. 2, a special PVSS API manager allows associating DIM commands and DIM services published by a DIM server running in the board controllers to the dynamic data structures in PVSS.

In order to access the actual board resources, a controller referred to as Credit-Card PC (CCPC) [5] is embedded on each of the TFC boards. All the board resources are accessed via the PCI bus of the CCPC. In order to facilitate interfacing to the different types of board control buses, an intermediate glue logic based on an FPGA has been developed [6]. The glue logic FPGA allows accessing via PCI a standard set of simple interfaces needed by all the different electronics boards, such as a parallel 32-bit Local Bus, I^2C, JTAG for device programming and boundary scan, and a General Purpose I/O (GPIO) interface.

II. Control Communication Protocol

A server, based on DIM for the communication with the remote control system, has been implemented on the CCPC and performs directly all the programming of the FPGAs and the configuration devices for the FPGAs, the board hardware configuration, the data register access, and the monitoring of each electronics board. It is independent of the board type and the organization of its resources. The DIM server is linked with functions to operate the JTAG bus, the Local Bus, the I^2C bus and the General Purpose I/O over the PCI bus.

In order for the programming of the programmable devices on the boards to be independent of the device types, the programming data is handled as byte-code Standard Test and Programming Language (STAPL) files [7]. The STAPL files contain both the programming data and the programming algorithm and are normally executed by the STAPL player from Altera. The DIM server has been linked with an adapted library version of the STAPL player which drives the JTAG bus over PCI.

The DIM server publishes a set of four pairs of generic DIM commands and services, described in detail in Sections II-A–D below, on the control network per board through which all the board resources can be controlled and monitored from the remote control system. The data structures are all dynamic in length in order to pack the control actions and thereby save on the network overhead and the call-back overhead in the server and in PVSS. In all cases except data subscription, a control command is always followed by an update of the corresponding service to pass back the result of the command to the remote control system.

A. Remote FPGA Programming

The command $\boldsymbol{DownloadFPGA}(struct[]\{id, code\})$ allows the remote control system to send a set of programming files. The identifiers are used by the server to direct the JTAG bus. After the programming the server sends back the status using the service $\boldsymbol{FPGALoadStatus}(struct[]\{id, status\})$.

B. Register Reading and Writing

The command $\boldsymbol{ReadWriteRegisters}(\ struct[]\ \{method, address, data, mask, write\})$ allows the remote control system to read or write a dynamic set of registers, each of which is located on the bus type specified by the access method. Currently there are three access methods implemented: Local Bus, I^2C or GPIO. The mask allows modifying only a subset of the bits. Each register write action in the server always consists of a sequence of a read, modify bits, write and a final read in order to pass back the updated value to the remote control system. This allows the control system to verify the integrity of the write action. The result of the write actions is passed back using the service $\boldsymbol{UpdateRegisters}(struct[]\{method, address, data\})$.

C. Reading and Writing Tables

The command ***ReadWriteTable****(struct {method, address1, address2, addr_start, nvalues, write, data[]})* allows the remote control system to read or write a table of data located on any of the bus types. By convention all tables are accessed through an address register and a data register with addresses on the control bus specified by *address1* and *address2*. The command allows starting the write anywhere in the memory by specifying the start address. The service ***UpdateTable****(struct {method, address1, address2, addr_start, nvalues, data[]}* is used by the server to return the table contents to the control system.

D. Data Subscription

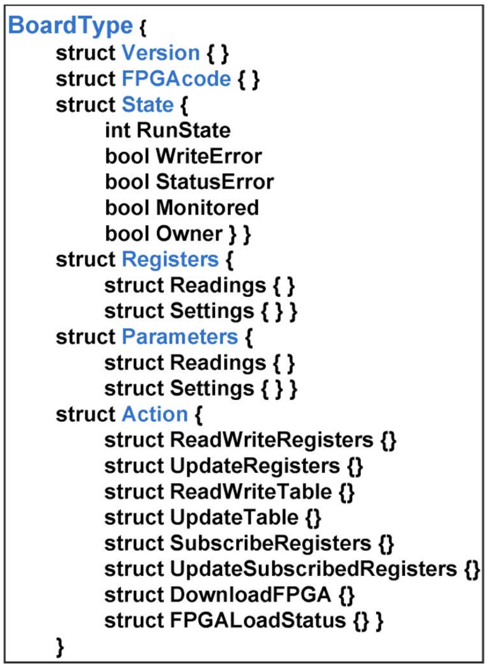
To avoid the traffic cost of polling counter and status registers, the command $\boldsymbol{SubscribeRegisters}(struct[]\{method, address, interval\})$ allows the remote control system to subscribe to a set of registers located on any of the bus types with a specified update interval. The DIM server creates dynamically a list of the subscribed registers and the timers, and automatically updates the values according to the specified intervals by returning them to the control system using the service $\boldsymbol{UpdateSubscribedRegisters}(struct\ []\ \{method, address, data\})$.

III. Representation of the Board Resources

Each electronics board type is represented by a dynamic database structure, i.e., a PVSS data point type, with an overall structure common to all board types (Fig. 3). Each individual board is stored as an instantiation of the corresponding data point type. The data point reflects the entire state of all the control and monitoring resources on each board.

In addition to the storage of the control and monitoring resources, the data point structure contains a set of substructures for the communication with the electronics board and with the expert system. The *Action* data structures are associated with the DIM commands and the DIM services described in the previous section and are the dynamic objects through which control data are exchanged with the server on the CCPC. The *State* structure contains global status information and is used to interface the board to an expert system for automated control.

The data point also contains a structure with the file pointers to the FPGA and configuration device programming files, and a *Version* and a *BoardIdentifier* structure which allows cross-checking that the control connection has been established with



```
BoardType {
        struct Version { }
        struct FPGAcode { }
        struct State {
                int RunState
                bool WriteError
                bool StatusError
                bool Monitored
                bool Owner } }
        struct Registers {
                struct Readings { }
                struct Settings { } }
        struct Parameters {
                struct Readings { }
                struct Settings { } }
        struct Action {
                struct ReadWriteRegisters {}
                struct UpdateRegisters {}
                struct ReadWriteTable {}
                struct UpdateTable {}
                struct SubscribeRegisters {}
                struct UpdateSubscribedRegisters {}
                struct DownloadFPGA {}
                struct FPGALoadStatus {} }
        }
```

Fig. 3. The representation of the board resources in the abstraction layer of the control system.

the correct board and that the firmware version matches the current board description.

The data structure of a board represents the board both from a hardware view and a functional view. The hardware view reflects the organization of the resources in terms of the onboard devices and their internal physical registers. As shown in the example in Fig. 4, the data registers are grouped in superstructures according to the access method and the base address of the device in which they are located. In this notation, "Q" corresponds to devices on the Local Bus such as FPGAs, and "I2C_" to I^2C devices, followed by the base address. As an example, the full physical address of register R004 in FPGA Q2 is 0×2004. To verify register write actions automatically in the control system, the data registers exist in two copies: *Register Settings* and *Register Readings*.

In order to save on logic resources, there is not always a one-to-one relation between physical registers and the functional parameters. In addition, the configuration and the state of a board is best expressed in terms of the functional parameters, and ultimately, the supervisory control system, the expert system and the graphics user interfaces will always act upon the functional parameters. The data structure therefore contains a functional view of the board as shown in Fig. 5. Like the hardware registers, the functional parameters are grouped into superstructures, however this time according to the function to which they belong. The functional parameters are also stored as *Parameter Settings* and *Parameters Readings* in order to display both the new requested settings and the current values in the graphics user interfaces.

The parameters of a functional group are always written together in one single DIM command. Hence this naturally defines the set of configuration and control actions of a specific board which may be called from the supervisory control system.

```
struct Registers {
        struct Readings {
                struct Q1 {
                        int R000
                        int R004
                        ...}
                struct Q2 {
                        int R000
                        int R004
                        ...}
                struct I2C_40 {
                        int R00
                        ...}
                }
        struct Settings {
                struct Q1 {
                        int R000
                        int R004
                        ...}
                struct Q2 {
                        int R000
                        int R004
                        ...}
                struct I2C_40 {
                        int R00
                        ...}
                }
```

Fig. 4. The hardware view in the board representation.

```
struct Parameters {
    struct Readings {
            struct HW {
                int H_CLK_EXT
                ...}
            struct System {
                int P_L0_LATENCY
                ...}
            struct Status {
                int S_ERR_PWR
                ...}
            struct Enable {
                bool R_L0_EXT_ENB
                ...}
            ...}
    struct Settings {
            struct HW {
                int H_CLK_EXT
                ...}
            struct System {
                int P_L0_LATENCY
                ...}
            struct Enable {
                bool R_L0_EXT_ENB
                ...}
            ...}
```

Fig. 5. The functional view in the board representation.

The hardware and the functional data structures are created from a Board Resource Descriptor which is described in Section IV.

IV. Dynamic Register-Parameter Translation

In the scheme described in the previous section in which the supervisory control system is acting on the functional view of the electronics boards, a translation stage is required which is based on a mapping between on the physical registers and the functional parameters. On the one hand, the mapping is needed when receiving data to decode the physical registers in order to extract the functional parameters. On the other hand, the mapping is needed when performing a control action to encode the functional parameters of a functional group into a set of physical registers with the appropriate masks to be written to the hardware.

```
Device1 {
    struct Version { }
    struct Registers {
        struct Q1 { }
        struct Q2 { }
        struct Q3 { }
        struct Q4 {
            . . .
            string R01C [ ] ={ {0x4034, 1, 1, P_IP_HDR_LEN, FrontEnd, 4, 0, 1}
                               {0x4034, 1, 1, P_IP_VERSION, FrontEnd, 4, 4, 1}
                               {0x4034, 1, 1, P_IP_SERVICE, FrontEnd, 8, 8, 1}
                               {0x4034, 1, 1, P_IP_TTL, FrontEnd, 8, 16, 1}
                               {0x4034, 1, 1, P_IP_PROTOCOL, FrontEnd, 8, 24, 1} }
            . . . } }
    struct FuncBlocks { }
    struct DataSubscribe { }
        }
```

Fig. 6. Board resource descriptor containing the mapping between the functional view and the hardware view of the board.

Using the PVSS API the translation stage has been implemented as a generic translation manager which is independent of the board type. The mapping between the hardware view and the functional view is not hard-coded but is stored in a separate board descriptor structure shown in Fig. 6. The board descriptor contains all the information needed to translate the registers into functional parameters and the vice versa to perform the configuration actions. As shown in Fig. 6, for each register the descriptor contains the list of all the parameters, each with the address of the register in which it is located, the access method, the data type, the name of the parameter, the name of the functional group to which it belongs, the position and size of the bit field, and whether a check should be made between the requested value and the value read after writing. In addition the Board Descriptor contains the list of data registers for the data subscription.

The translation manager is written in $C++$. As shown in the example in Fig. 7, it creates dynamically the translation objects as $C++$ objects run-time upon reading the Board Resource Descriptor at start-up. Thus it is sufficient to start one instance of the translation manager per board type, or possibly a few instances for performance reasons if there are a very large number of boards of the same type. The translation manager performs automatically the translation by associating the translation objects to the data elements in the functional and the hardware representation of the board as call-back functions whenever a value changes. Fig. 7 shows an example of the processing in the case of the register to parameter translations. The 'reception object' is associated to the DIM service through which the register data is received. It decodes the address and sets the value in the hardware part of the device representation. An automatic comparison is made between the received value and the expected value in order to verify the integrity of the control action. Based on the Board Resource Descriptor the translation manager calls

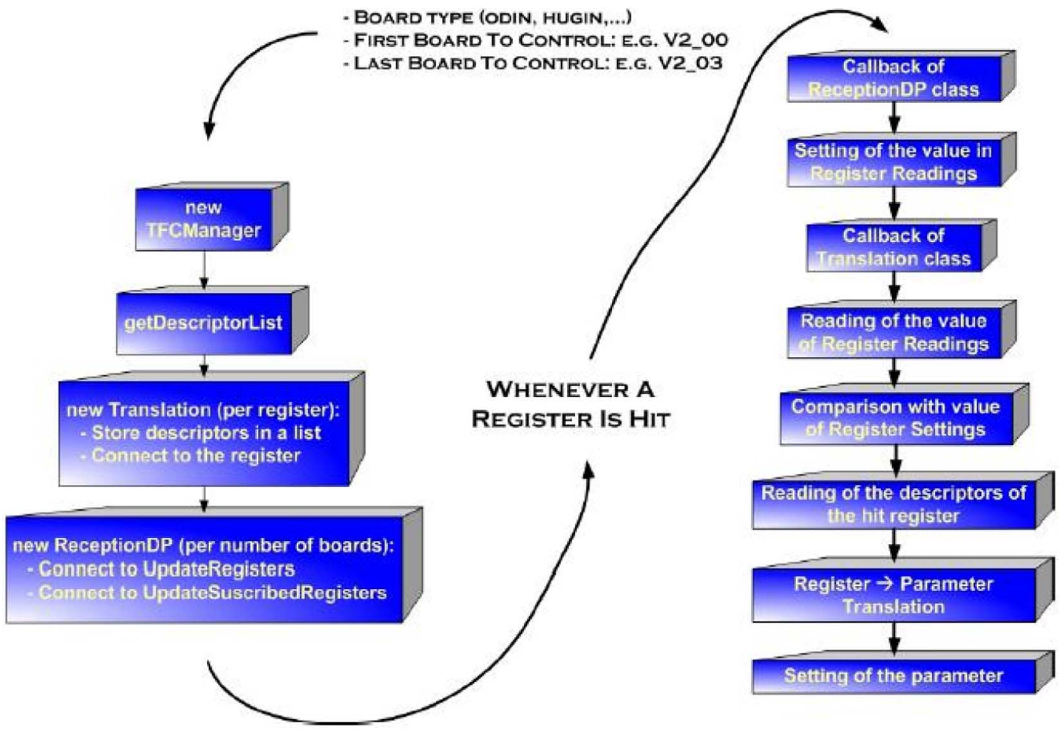
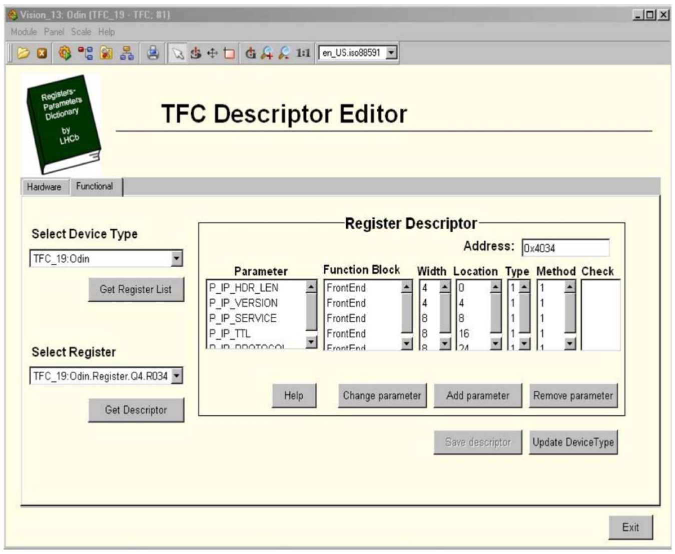

- Board type (ODIN, HUGIN,...)
- First Board To Control: e.g. V2_00
- Last Board To Control: e.g. V2_03
new TFCManager
getDescriptorList
new Translation (per register):
- Store descriptors in a list
- Connect to the register
new ReceptionDP (per number of boards):
- Connect to UpdateRegisters
- Connect to UpdateSuscribedRegisters
Whenever a Register Is Hit
Callback of ReceptionDP class
Setting of the value in Register Readings
Callback of Translation class
Reading of the value of Register Readings
Comparison with value of Register Settings
Reading of the descriptors of the hit register
Register → Parameter Translation
Setting of the parameter

Fig. 7. Process flow in the translation API manager for the case of the register to parameter translation.



TFC Descriptor Editor

Select Device Type: TFC_19:Odin

Select Register: TFC_19:Odin.Register.Q4.R034

Register Descriptor

Address: 0x4034

Parameter	Function Block	Width	Location	Type	Method	Check
P_IP_HDR_LEN	FrontEnd	4	0	1	1	
P_IP_VERSION	FrontEnd	4	4	1	1	
P_IP_SERVICE	FrontEnd	8	8	1	1	
P_IP_TTL	FrontEnd	8	16	1	1	
P_IP_PROTOCOL	FrontEnd	8	24	1	1	

Fig. 8. The board resource descriptor editor.

the corresponding 'translation object' which decodes the physical register data and sets the functional parameters.

The Board Resource Descriptor is edited using a Resource Descriptor Editor shown in Fig. 8. It also allows creating and updating the actual board representation described in Section III for each type of electronics board.

V. Data Register Access Flow

As an example of the scheme outlined in this paper, the entire register access sequence is shown in Fig. 9. A register configuration request may come from a user interface as well as from the supervisor control system. In Fig. 9, a user requests a change of a parameter in the graphics user interface (GUI). Upon applying the value, the value is written to the appropriate functional parameter under *Parameter Settings* in the board data structure. The translation object in the translation manager associated with the configuration action of this particular functional block writes the value of the parameter in the physical register in which it is located under *Register Settings* and writes the new register value, the access method, address and mask to the action structure *ReadWriteRegisters* which is associated with the *ReadWriteRegisters* DIM command. The other parameters belonging to the same functional block will also be sent together.

The DIM server will react by reading the register in the electronics board, modify the value according to the mask and write the new register value. The value is then read again and returned via the *UpdateRegisters* DIM service. The data is received by the *UpdateRegisters* structure in the board data structure. The translation manager decodes the addresses and writes the values to the appropriate register under *Register Readings*. The translation objects associated with the registers decodes the contents and updates the functional parameters under *Parameter Readings*. In addition the translation manager will automatically verify that the return values of the registers match the requested values of those parameters which should be checked. Since the user interfaces can subscribe to the functional parameters, the display is automatically updated with the latest settings.

VI. Conclusion

A general purpose strategy to building a control system for a large number of different electronics boards has been outlined. It allows a common and generic approach to interfacing the supervisory control system to all the resources for control and monitoring. The resource organization in the hardware is completely

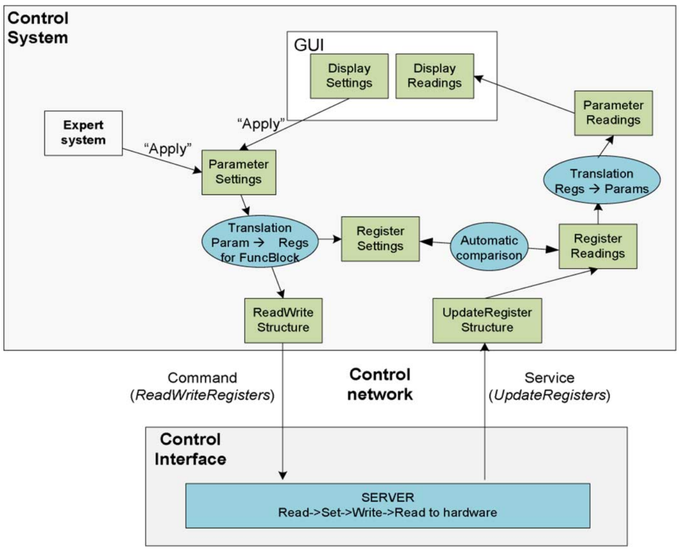


Fig. 9. Complete view of the data flow during a data register access.

hidden and the supervisory control handles the resources from the logical point of view, that is, in terms of the functional parameters. All the specificities of the different boards are entered into the control system via Board Resource Descriptors which hold the hardware structure and the functional structure of the board and the mapping between the two, and which are created and edited using a graphics user interface.

Based on this a complete run control system has been implemented for the LHCb Timing and Fast Control system. The control system has been in use since 2002 during the entire prototyping of the TFC system and the developments of the LHCb sub-detector electronics, and is now installed for the commissioning and operation of the LHCb experiment.

ACKNOWLEDGMENT

The author would like to thank Zbigniew Guzik, Soltan Institute for Nuclear Studies, Poland, for many fruitful discussions and suggestions, and Daniel Alonso Alvarez, University of Oviedo, Spain, who worked as a CERN Summer Student on the translation manager and the descriptor editor. The author would also like to thank Clara Gaspar, CERN, for many useful suggestions and help with PVSS and the CERN JCOP components, and Niko Neufeld, CERN, for setting up and maintaining the Credit Card PCs and providing the PCI driver.

REFERENCES

[1] Z. Guzik, R. Jacobsson, and B. Jost, "Driving the LHCb front-end readout," *IEEE Trans. Nucl. Sci.*, vol. 51, no. 3, pp. 508–512, Jun. 2004.
[2] LHCb Tech. Proposal LHCb Collaboration, CERN-LHCC-98–04.
[3] ETM Professional Control [Online]. Available: http://www.pvss.com
[4] C. Gaspar, Distributed Information Management System [Online]. Available: http://cern.ch/dim
[5] F. Fontanelli *et al.*, "Embedded controllers for local board control," *IEEE Trans. Nucl. Sci.*, vol. 53, no. 3, pp. 936–940, Jun. 2006.
[6] Z. Guzik and R. Jacobsson, "Glue Light—A simple programmable iinterface between the credit card PC and board electronics," LHCb Tech. Note 2003–56, 2003.
[7] Altera, ISP and the Jam STAPL [Online]. Available: www.altera.com/support/devices/tools/jam/tls-jam.html