CONTROLLING ELECTRONICS BOARDS WITH PVSS

R. Jacobsson
CERN, 1211 Geneva 23, Switzerland

ABSTRACT

This paper addresses several aspects of implementing a control system for electronics boards in order to perform remote Field Programmable Gate Array (FPGA) programming, hardware configuration, register control, and monitoring, as well as interfacing it to an expert system. The paper presents an implementation, using the Distributed Information Management (DIM) package [1] and the industrial SCADA system PVSS II from ETM [2], in which the access mechanisms to the board resources are completely generic and in which the device description and the handling of mapping between functional parameters and physical registers follow a common structure independent of the board type. The control system also incorporates mechanisms by which it may be controlled from a finite state machine based expert system. Finally the paper suggests an improvement in which the mapping between logical parameters and physical registers is represented by descriptors in the device description such that the translation can be handled by a common method

The implementation has been applied to build a local run control for the Timing and Fast Control (TFC) system [3] of the LHCb experiment at CERN. It has been in use during the entire prototyping of the TFC system and is now installed centrally at CERN for the LHCb sub-detector tests.

INTRODUCTION

In a system of many different electronics boards which should be controlled from a remote supervisory control system it is important to devise a common strategy for the representation of the devices and generic communication protocols for the access to the board resources. Given that a set of electronics boards are controlled from a common type of control interface which has access to set of board control buses, the aim here is to:

- Define a generic data structure which reflects the control configuration of a board and on which the supervisory system can perform control actions, either directed from a user interface or from an expert system, and verify the integrity of the register control.
- Provide a simple and economical 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.

In LHCb the electronics boards will all be controlled from the overall Experiment Control System (ECS) [4]. In order to access the actual board resources, an ECS interface will be located on each board. The ECS interface is connected to the control network and performs directly all device programming, configuration, control and monitoring of each electronics board. For the electronics situated in the LHCb counting rooms behind the shielding wall, the ECS interface is based on a commercial Credit Card PC (CCPC) with Ethernet from Digital Logic, AG, Switzerland [5].

All the board resources are accessed via the PCI bus of the CCPC. In order to facilitate interfacing to different types of board control buses, a small intermediate mezzanine card “Glue Card” has been developed [6]. The glue card is connected to the PCI bus and form a standard set of simple interfaces needed by all the different electronics boards, such as a parallel 32-bit Local Bus, I^2C, JTAG and an 8-bit General Purpose I/O interface.

CONTROL SYSTEM ARCHITECTURE

Figure 1 shows an overview of the local control system architecture.

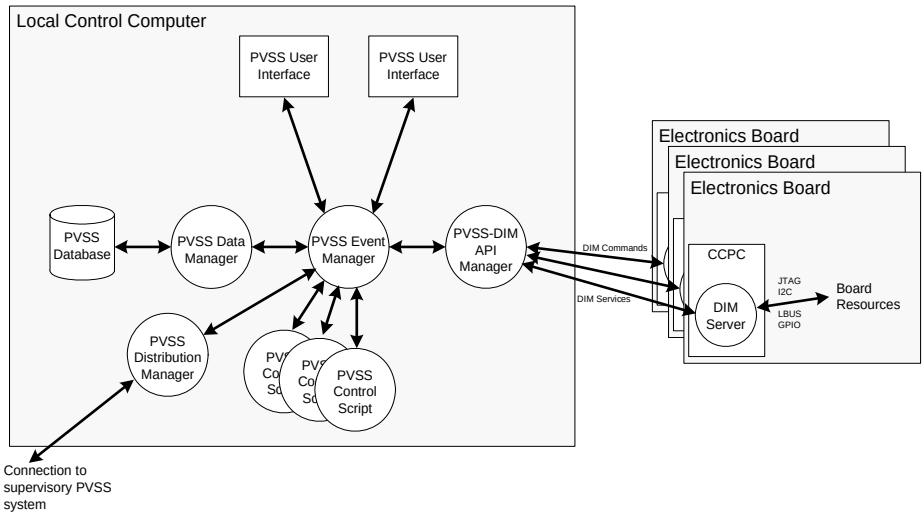

Figure 1: Overview of the local control system architecture.

DIM Server

Using the Distributed Information Management (DIM) package [1], a generic DIM server has been implemented on the CCPC which performs directly all the FPGA programming, hardware configuration, register control, and monitoring of each electronics board. The DIM server is linked with functions to operate the four different interfaces over the PCI bus.

The DIM server is automatically started on each electronics board when the CCPC is booted. At start-up it accesses a board identifier stored according to a common format in an I^2C Serial EEPROM on each board. This allows the server to publish on the control network a set of identifiable DIM commands and DIM services through which all the board resources can be controlled and monitored from the remote control system. The services and commands have a completely general purpose structure and are the same independently of the board type.

PVSS Control System

The supervisory control system is based on the industrial distributed SCADA system PVSS II from ETM [2]. PVSS is centred on an Event Manager which acts in between a Data Manager in association with 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 may be associated. The data points are used to build up complete device descriptions of the electronics boards. A special PVSS API manager allows associating DIM commands and DIM services to data points [1].

DEVICE DESCRIPTION

Each electronics board type is described by a PVSS data point type with an overall structure common to all board types (Figure 2). Each individual board is stored as an instantiation of the corresponding data point type. The data point reflects the entire state of all the controllable resources on the board.

To verify register write actions automatically in PVSS, the data registers exist in two copies: *Register Settings* and *Register Readings*. The data registers are divided into groups according to the base addresses of the devices in which they are located. The notation here is such that "Q" corresponds to 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 0x2004.

Since there is not always a one-to-one relation between physical registers and the functional parameters, the data point also stores the values of the parameters, again as both *Parameter Settings* and *Parameter Readings*. These are the values acted upon and displayed in the user interfaces. The physical parameters are divided into logical groups according to the board function to which they belong. The set of configuration actions for a particular board type corresponds one-to-one to the

logical groups, which means that a configuration request from a user interface or from a supervisory control system updates the entire set of parameters in one logical group.

PVSS allows associating event-triggered functions with data points. Whenever the content of a data point element changes a function may be called in a PVSS script or API manager to perform a set of actions associated with the change. This mechanism allows setting up functions which perform the configuration actions and other global actions, and which make the automatic translation between physical registers and functional parameters and vice-versa.

The data point element *Apply* is associated with a function in which all global actions of a particular board type are decoded. Examples of global actions are SystemReset, Initialize, ResetAllCounters, SubscribeAllCounters, StartRun, StopRun, AcknowError, ApplyL0Settings, ApplyHWSettings etc.

In order to perform the control of an electronics board and receive information from it, the data point for each board contains a set of common *Action* structures which are associated with DIM commands and DIM services. These will be discussed in the sections which follow.

The data point also contains a structure with the file pointers to the FPGA programming files and a *State* structure used to interface the control system to a supervisory expert system for automated control. And finally the data point of each board also holds the *BoardIdentifier* of the particular board.

```
BoardType {
        int BoardID
        struct FPGAcode { string Q1
                        …}
        struct State {
                int RunState
                …}
        struct Registers {
                struct Readings {
                        struct Q1 { int R000
                                …}
                        struct I2C_40 { int R0C
                                …}
                        …}
                struct Settings {
                        struct Q1 { int R000
                                …}
                        struct I2C_40 { int R0C
                                …}
                        …}
        struct Parameters {
                struct Readings {
                        struct HW { int H_CLK_EXT
                                …}
                        struct Status { int S_ERR_PWR
                                …}
                        …}
                struct Settings {
                        struct HW { int H_CLK_EXT
                                …}
                        struct Status { int S_ERR_PWR
                                …}
                        …}
                int Apply }
        struct Actions {
                struct ReadWriteRegisters {}
                struct UpdateRegisters {}
                struct SubscribeRegisters {}
                struct UpdateSubscribedRegisters {}
                struct DownloadFPGA {}
                struct FPGALoadStatus {} }
        }
```

Figure 2: Common data point structure describing an electronics board.

FPGA PROGRAMMING

The programming of FPGAs and configuration devices is done using JTAG directly from the CCPC with byte-code Standard Test and Programming Language (STAPL) files [7]. The STAPL files contain both the programming data and the programming algorithm, and are executed by a STAPL player.

The DIM server is linked with the STAPL player software and publishes one DIM command Download*FPGA(struct{id, data})* by which PVSS sends the programming file and the identifiers of the devices to be programmed. Instead of storing the programming files in PVSS, the data point holds the file pointers to the files. PVSS reads the files upon receiving a programming request and writes the data to the *DownloadFPGA* action structure in the data point, which triggers the sending via DIM.

In order to inform the control system about the status of the programming, the DIM server publishes a service *FPGALoadStatus* with the identifier of the programmed devices and a status flag.

The entire programming sequence is shown in Figure 3. The programming request may come from a user interface as well as from a supervisory control system.

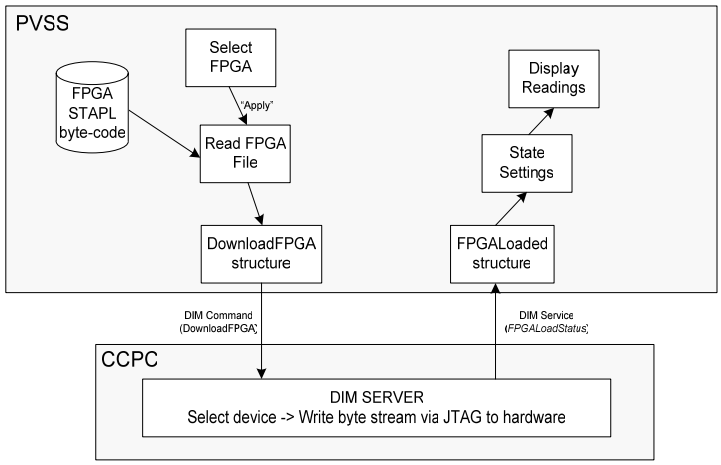

Figure 3: Flow chart of the FPGA programming

REGISTER CONTROL

The DIM server publishes one DIM command and one DIM service for the control of any data register on the electronics board:

- *ReadWriteRegisters(struct[] {method, address, data, mask, write}):* Command to read or write registers from PVSS.
- *UpdateRegisters(struct[] {method, address, data}):* Service to update in PVSS a register value which was read by the DIM server.

Both have a dynamic length meaning that any number and types of data register may be accessed in a single action. The parameter *method* indicates the access method for each register which in the current implementation is Local Bus (0), I^2C (1) or GPIO (2). The *address* is the full physical address, the *mask* allows modifying only specific bit fields of a wider register, and the parameter *write* indicates if the action is a write (1) or only a read action (0). A write action always consists of a read of the register, followed by a write of the new data using the mask and last another read in order to send back the data that was actually written.

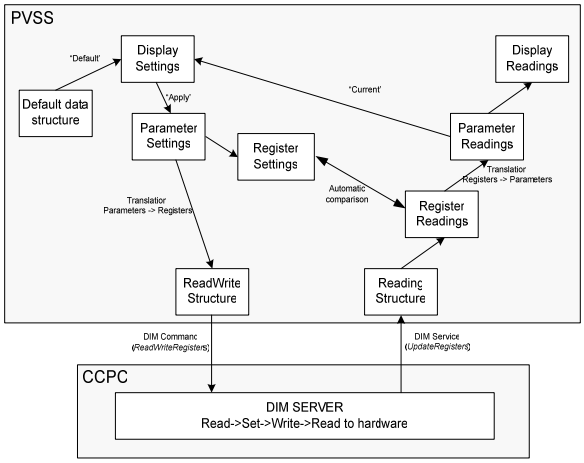

Figure 4: Flow chart of the register control.

The entire register control sequence is shown in Error! Reference source not found.. A register configuration request may come from a user interface as well as from a supervisor control system. In Error! Reference source not found. a user requests a change of a parameter in the user interface. Upon applying the value, the value is written to the appropriate functional parameter under *Parameter Settings* in the data point. The function associated with the configuration action changes 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 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 data point and is written to the appropriate register under *Register Readings* according to the received address. In addition to updating the physical parameters of the register under *Parameter Readings*, a function also verifies that the return value of the register matches the requested value. Since the user interfaces can subscribe to the parameters, the display is automatically updated with the latest settings.

DATA SUBSCRIPTION

In order to avoid the traffic cost of polling counter and status registers, the DIM server publishes a command and a service to subscribe to registers and have them updated in PVSS regularly. The command *SubscribeRegisters(struct[] {address, interval})* has a dynamic length which allows subscribing to a set of registers with a specified update interval. The DIM server creates dynamically a list of the subscribed registers and the timers. Additional registers may be subscribed to any time.

Upon expiration of a timer the associated registers are read and updated in PVSS via the DIM service *UpdateSubscribedRegisters(struct [] {address, data})*.

The data is received by the *UpdateSubscribedRegisters* structure in the data point corresponding to the board and is written to the appropriate register under *Register Readings* according to the received addresses. As with the register control the values may be updated automatically in the user interfaces. Figure 5 shows a flow chart of the register subscription. Currently only Local Bus registers are subscribed to but by adding access method to the subscription it would be possible to subscribe to a register on any of the bus types.

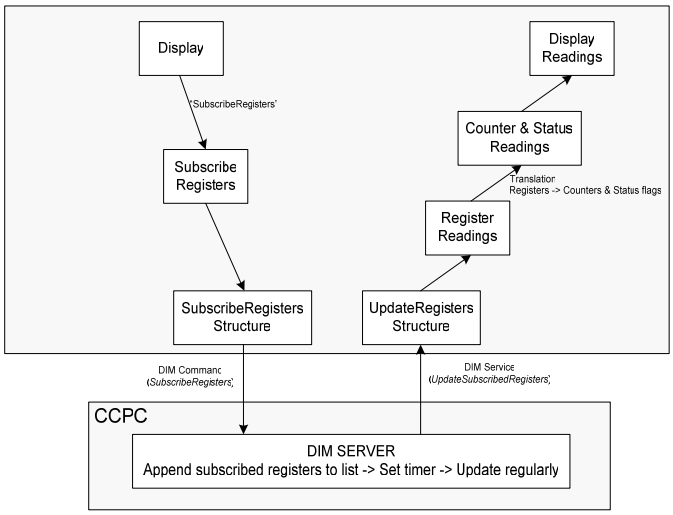

Figure 5: Flow chart of data register subscription.

GRAPHICS USER INTERFACE

As mentioned earlier the register control is handled in terms of the logical blocks of functional parameters, which is also reflected in the user interfaces. Figure 6 shows a part of a user panel. The functional parameters have a field in which the current setting is displayed and a separate field for requesting a new value. The entire set of values is submitted with the “Apply” which corresponds to the same action which would be applied by a supervisor control system.

In order to avoid filling in all values it is possible to load all current values or a specific default setting onto all the requested values and modify them individually before submitting them. The default settings are currently loaded from special instantiations of the data points with different defaults. However, the mechanism also allows loading complete default configurations from a configuration database upon a command from a supervisory control system.

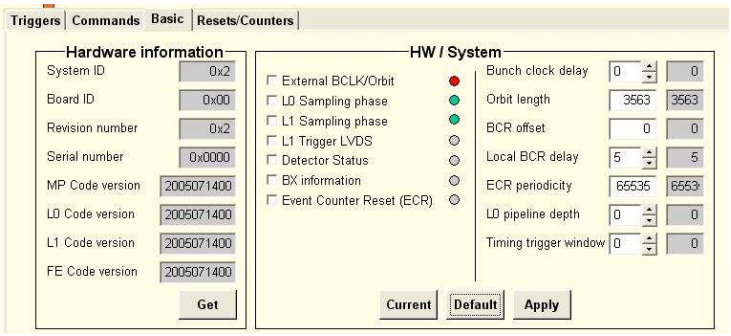

Figure 6: Example of a logical block of parameters in the user interface.

IMPROVED HANDLING OF REGISTER-PARAMETER TRANSLATION

Currently the functions to perform the register-parameter translation and the global control actions are hard coded in PVSS scripts. This has two disadvantages. Although it doesn't have a prohibitive effect on the current implementation, the performance of the PVSS script is inherently rather low. Secondly, the fact that the translation between registers and parameters is hard coded implies that there is a lack of generality in that there is a need for a special script for each board type. Also, any modification to the register organization in the board requires modifying not only the data point type but also the script.

An improvement is underway in which the data point contains the mapping between physical registers and functional parameters in a structure of descriptors. For each parameter the descriptors contain the name of the parameter, the address of the register in which it is located, and the position and size of the bit field. This makes it possible for a generic function to read the descriptors at start up and set up the translation runtime. In order to improve on the performance the aim is to implement this as a PVSS API manager.

CONCLUSIONS

Based on the ideas described in this paper a complete local run control system with user interfaces and low level control of the electronics boards has been implemented for the LHCb Timing and Fast Control system. The TFC system consists of some 75 boards of five different types. The most complex board has over 250 functional parameters in 150 physical registers. The control system has been in use during the entire prototyping of the TFC system and is now in use for the testing and commissioning of the detector Front-End electronics. The integration with the overall control system of LHCb, in particular with the configuration database and with the control automation based on the State Management Interface++ (SMI++) [8], is already underway.

ACKNOWLEDGEMENT

I would like to thank Clara Gaspar for help and many fruitful discussions and Niko Neufeld for maintaining the Credit Card PCs and providing the PCI driver.

REFERENCES

[1] DIM [online] http://cern.ch/dim/
[2] PVSS [online] http://www.pvss.com/english/index.htm.
[3] Z. Guzik, R. Jacobsson, B. Jost, *Driving the LHCb Front-End Readout*, IEEE TNS Vol. 51, pp 508-512, 2004.
[4] The LHCb Collaboration, *LHCb Online System TDR*, CERN/LHCC 2001-040.
[5] C. Gaspar et al.,The Use of Credit Card-sized PCs for interfacing electronics boards to the LHCb ECS, LHCb 2001-147.
[6] Z. Guzik and R. Jacobsson, *"Glue Light"- A Simple ProgrammableIinterface between the Credit Card PC and Board Electronics*, LHCb 2003-56.
[7] Altera [online] http://www.altera.com/support/devices/programming/jam/dev-isp_jam.html
[8] SMI++ [online] http://cern.ch/smi/