INTEGRATION OF THE LHC POWER CONVERTERS WITHIN THE HIGH-LEVEL LHC CONTROL SYSTEM

S. Page[1]
[1]*CERN, Geneva, Switzerland*

ABSTRACT

The LHC will use some 1700 power converters, each of which will be locally controlled by an embedded Function Generator/Controller (FGC) connected to one of ~70 WorldFIP fieldbuses. To operate the LHC, all power converters must be controlled and co-ordinated in a synchronised manner within the high-level LHC control system.

This paper describes the architecture and systems involved in integrating the control of the power converters into the high-level LHC control system, including: (i) management of the FGCs on the WorldFIP bus, (ii) reception of commands and delivery of status data via the Controls Middleware (CMW), (iii) integration into the LHC Java client API, (iv) monitoring and reporting of failures and warnings via the LHC alarm system and (v) extraction of post-mortem data in the event of beam-loss or equipment failure. The paper also provides an overview of the XML-based development system that supports and documents all parts of the LHC power converter control system.

SYSTEM ARCHITECTURE

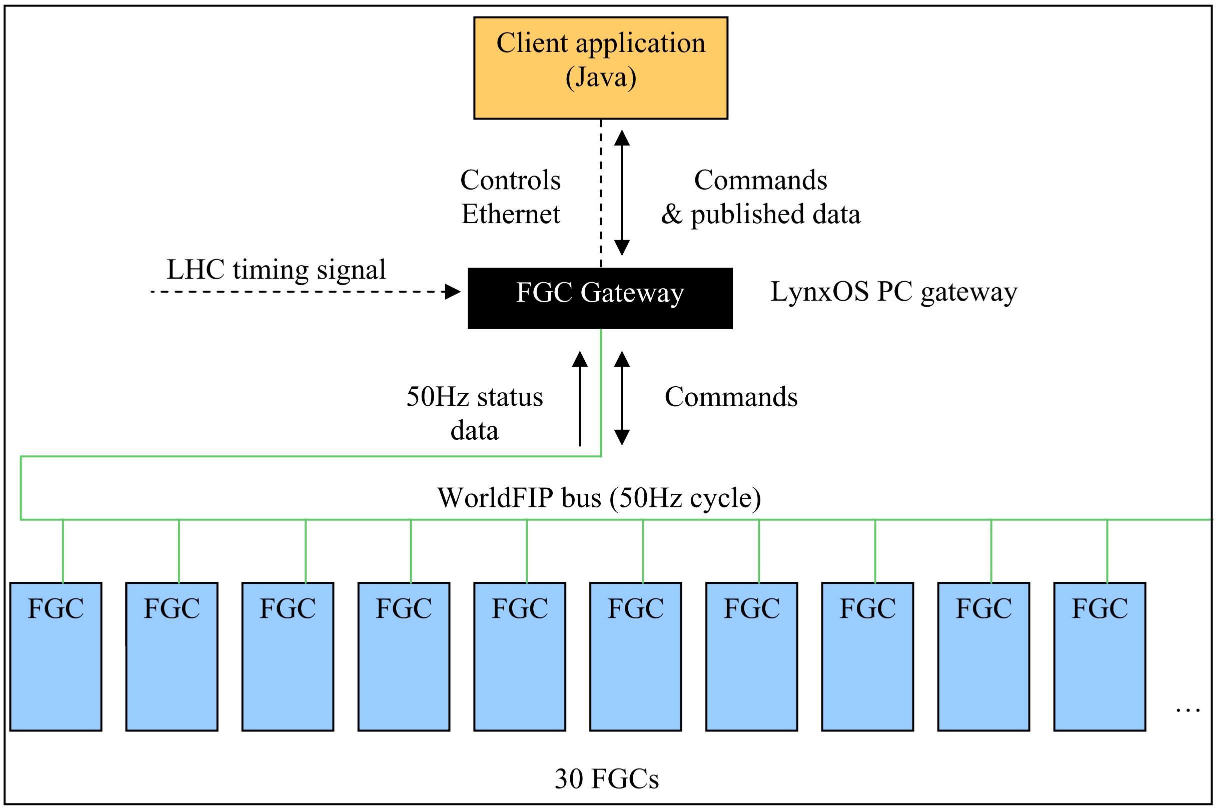

Figure 1: LHC power converter control system architecture

The system for controlling the LHC power converters can be divided into three basic layers as shown in Figure 1. At the bottom layer, every power converter has a Function Generator/Controller (FGC) [1] installed within it that performs the function generation and local control of the converter. An FGC contains all of the logic necessary to have complete local control of an individual power converter and if no coordinated control is needed between converters, a simple VT100 terminal connected to an FGC gives a user complete access to the converter’s functions.

For the LHC, control of all of the power converters must be synchronised. This is achieved by using WorldFIP real-time fieldbuses running at 2.5MB/s to connect all of the FGCs to x86 PCs running LynxOS that form gateways to the Controls Ethernet network. A maximum of 30 FGCs may be connected to a single fieldbus segment, however due to their geographical distribution, between 70 and 80 WorldFIP segments are required in order to connect all of the LHC power converters to the Controls network. Each gateway acts as the bus arbitrator for its WorldFIP segment and is equipped with an LHC timing receiver that is used to maintain synchronisation between the WorldFIP networks. The WorldFIP fieldbus is used to transmit both control commands and timing to the power converters [2].

The top-level consists of controls applications written in Java, that communicate with the power converters and other LHC subsystems using a CORBA based middleware.

WorldFIP communications

Communications across the WorldFIP fieldbus are based around a repeating cycle of 20 milliseconds in length, a simplified version of which is shown in Figure 2. Each cycle is started by a trigger pulse from the timing receiver in the gateway, ensuring that the cycles of all gateways are aligned. The cycle is driven by the gateway which acts as the bus arbitrator and therefore dictates which device may transmit data on the bus at any given time.

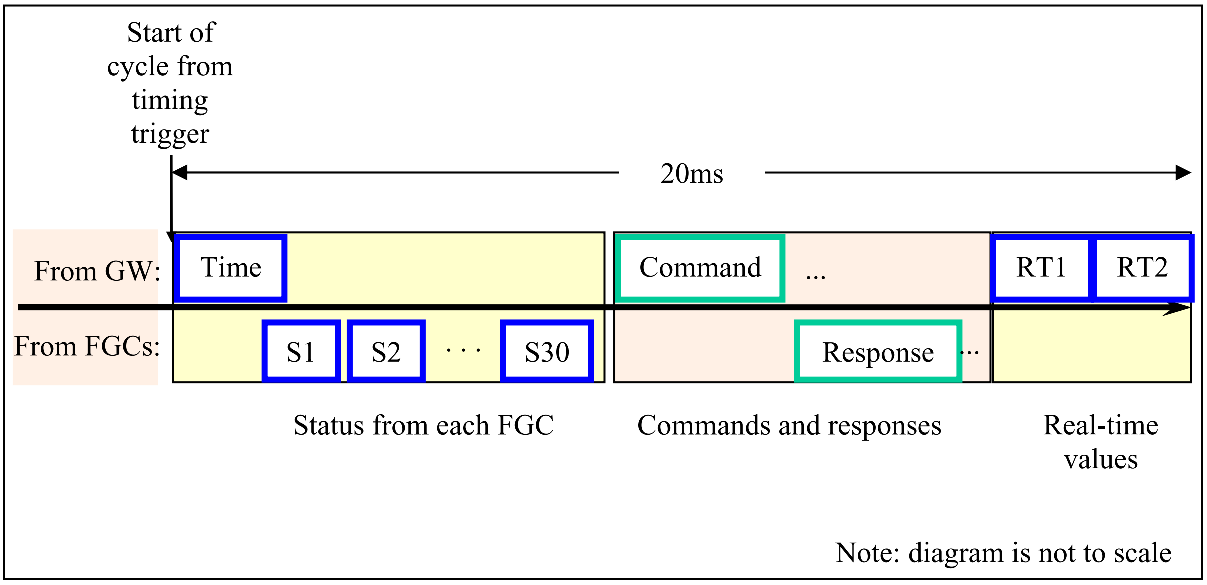

Figure 2: Simplified view of the WorldFIP cycle

At the start of each cycle, the gateway transmits a variable containing the current UTC time as well as some other data used by the FGCs. The timing of the cycles is critical as the arrival of this variable is used to regulate a clock in each FGC using a phase-locked loop [2].

After the time variable from the gateway, each FGC on the bus in turn transmits a block of status data. Each status block includes the main states of the power converter in question and the most important analogue values (including the reference and measured currents and voltages). The FGCs send this status block in every cycle, so each gateway has access to all of the values contained within the status blocks for its power converters at 50Hz. Any FGCs not reporting status blocks for two consecutive WorldFIP cycles are deemed to be offline and the gateway will return an error to any client attempting to send commands to them.

Following the status blocks is what is known as the “aperiodic window”. During this window, command messages (of varying length) may be sent by the gateway to the FGCs and responses (also of varying length) may be read back. It is through these messages that commands are transmitted to the power converters, with the gateway receiving them over the Controls Ethernet and retransmitting them over the WorldFIP bus.

At the end of each cycle, the gateway transmits two real-time variables containing reference values to be applied by the FGCs during the next cycle. These are used to allow real-time feedback of beam parameters for the LHC.

XML DEFINITION SYSTEM

The control system for the LHC power converters is complex and involves many elements, both as parts of the online control system and as components to support the development process. For example, in order to ensure that every FGC (and each of the individual components within it) is in working order on arrival from the manufacturer, a dedicated tester platform has been designed. The software for the tester platform is tightly bound to the software that runs in the FGC itself so changes in the FGC software must be taken into account when the tester is compiled.

The solution to such interdependencies – which run throughout the project – is to define all of the common data in one place only, from which it may propagate. In the case of the FGC project, it was decided that all aspects of the system should be described in the XML mark-up language. Details of the more important aspects defined in this way follow:

Device platforms

The basis of every device is a device platform. The device platform describes the broad environment for the device and may include information about the hardware and software that supports the device. The three most significant platforms within the FGC project are the FGC itself, a tester platform for testing the FGC hardware and the gateway platform.

Memory maps

A memory map may optionally be defined for each platform, describing the layout of its memory to be used later for both documentation and code. The memory maps are hierarchical in nature and allow memory zones of any size to be defined with an unlimited number of children. This allows definitions to be made at an appropriate resolution depending upon requirements of the system involved.

The memory map definition system is applied extensively in the case of the FGC to define everything from the location of the memory pages accessible to the processors down to bits in individual registers.

Device classes

Each platform may have several variants which are termed “classes”. A class is typically an individual variant of the software for a given platform. For example there are two classes for the FGC, one for its role in the LHC power converters and a second for function generation for the LHC RF system. Device classes allow system variants to be made whilst retaining the common attributes of the device platform.

Properties

Each device class may have a number of properties that are used to read (in the form of a ‘get’ method) or write (through a ‘set’ method) a value or setting for the device. It is through these properties that the devices – and therefore the power converters – are controlled. A full description of every property, including its type, size and documentation is described in XML.

Published data

Each device class has a block of status data associated with it that is known as its “published data”. This data is a 56 byte block of binary data, of which 16 bytes are common to all devices and the remaining 40 bytes are specific to the particular class of device. The contents of the published data are described in an XML file such that it may later be decoded. All fields within the published data represent the values of properties for the device and are therefore also available through ‘set’ and ‘get’ commands.

PARSER

Once the XML descriptions for the various project elements have been written, the data within them must be validated, and then propagated to its various targets. This is performed by a parser program written in Perl. The parser operates in two stages. First during its input stage, it reads all of the XML files for the entire project, validating each in turn and populating a tree structure in memory with the data contained in the files and their many interrelationships. Once all of the files have been read and the tree fully populated the parser progresses to its output stage. In the output stage, the parser extracts data from the tree in memory and uses it to generate files in the many areas of the project where the data must be used. A subset of them is described below.

Documentation

Every aspect of the FGC project is documented on the web in HTML. The vast majority of this documentation is automatically generated, based upon the data defined in the XML files. The different sections of the generated documentation are tightly integrated with each other, and it is simple for a developer or user to navigate through the information available for a particular device platform or class.

By generating a large proportion of the documentation from the same source as is used to build much of the source code for the running system, it is ensured that the documentation is always consistent with the actual system in use. Static documentation is limited to user guides and procedures that change rarely.

Device source code

As mentioned previously, the hierarchical property tree for a device class is entirely defined within the XML files. The parser produces a C header file containing structures representing the property tree, which is then compiled into the software for the device. Due to the size and complexity of the property trees, it would be difficult to manage them by editing the header file manually. By integrating this process in the parser, the addition of a property to a device can be made simply by modifying an XML file, running the parser, then recompiling the source code.

C header files including constants describing the memory maps that were defined in the XML for the platforms are also generated. Once the memory map of a platform has been defined in the XML, developers can work with the constants that are generated from it without needing to know the memory addresses of the hardware components that they are using.

Client source code

Several programming languages are supported for the control of the LHC power converters, including Java, C and Perl. API libraries are provided to client developers in each of these languages and these must be maintained so as to retain coherency with the actual running version of the device software in the LHC. This is achieved by the parser automatically generating the sections of code that are dependent upon the device software based upon the definitions in the XML files. Typically, this means generating code to resolve available property names and also to extract values from the binary published data block for a device class.

INTEGRATION WITH LHC CONTROLS SERVICES

Controls Middleware

The gateway software architecture is designed in such a way as to allow communications protocols to be easily added and to co-exist with each other. A CORBA based middleware known as the Controls Middleware (CMW) [3] has been adopted as the standard protocol for the control of all LHC equipment and as such is integrated into the FGC gateways to allow control of the power converters.

The gateways receive commands from CMW over the Controls Ethernet network, and then reformat them before transmitting them to the FGCs over the WorldFIP fieldbus. The status blocks sent by the FGCs in every WorldFIP cycle are also available through CMW in the form of a subscription which clients may monitor at a rate of 2Hz.

Client Application Interface

The LHC controls applications will have to interface with a wide range of equipment provided by many different groups within CERN. This presents a problem for applications programmers who do not wish to contend with the intricacies of different equipment interfaces. The solution to this is an abstraction layer known as the Java API for Parameter Control (JAPC) [4]. Equipment providers must also provide a series of Java classes that comply with the specification of the JAPC project, allowing application developers to interface with all types of equipment in a standard way.

The LHC power converters are no exception to this. The XML parser uses the data that it has accumulated to automatically generate JAPC java classes for all of the classes of device in the project. As a result of this, manual changes to the JAPC classes are seldom needed with any changes to the properties for the FGCs in the LHC power converters being automatically reflected in the interface used by developers of the controls applications.

LHC Alarm System

The LHC alarm system is known as LASER [5]. It is used to report equipment problems to consoles in the LHC control room.

For the LHC power converters, it was decided to divide the alarms into two categories: “faults” that will trip off a power converter (or prevent it from turning on if already off), and “warnings” that indicate a loss of redundancy or degradation in performance of a converter, whilst allowing it to continue operating. The faults and warnings are two bitmask fields in the status block of every device, therefore the FGC gateways receive them from all of the FGCs connected to their WorldFIP segments at 50Hz. The gateways report alarms appearing as bits set in the masks to the LASER alarm system at a sub-sampled rate of once every 5 seconds.

In order that the alarms appear as meaningful messages on the LASER console in the LHC control room, data to describe all of the possible alarms must be entered into a database for the alarm system beforehand. The parser extracts the data required by the LASER database from the tree formed from the XML definition files and puts into a form that allows the database to be updated with a single command.

LHC Post-Mortem System

The post-mortem system for the LHC is currently in the design and prototyping phase. In the event of the loss of an LHC beam or the quench of a superconducting magnet, post-mortem log buffers recording data around the moment of the incident must be retrieved from the FGCs and sent into the post-mortem system for further analysis to locate the source of the problem. The gateways will also submit a buffer to the post-mortem system containing the status data for each power converter that is received at 50Hz over the WorldFIP. The prototype post-mortem system allows the submission of binary buffers described in the BinX XML schema. Following the prototype stage, this will be integrated into the main body of XML files handled by the parser.

CONCLUSION

The control system for the LHC power converters is complex and requires the integration of many elements both internal and external to the project. Managing the integration of these elements and maintaining coherency between them presents a significant challenge. By basing the project around an XML-based definition system that forms a single source for all shared data, this challenge was overcome. The LHC power converter control system has proved itself through its successful use in the LHC Magnet Test Benches and also for the commissioning of power converters installed in the LHC tunnel. This experience with the control system has been very good and the development system behind it has greatly simplified the interrelationships between all control system elements involved as well as allowing for their future evolution.

REFERENCES

[1] J.C.L. Brazier, et al, “The All Digital Approach to LHC Power Converter Current Control”, ICALEPCS’2001, San Jose, USA, November 2005.
[2] Q. King, “Advanced uses of the WorldFIP fieldbus for diverse communications applications within the LHC power converter control system”, ICALEPCS’2005, Geneva, Switzerland, October 2005.
[3] K. Kostro, et al, “The Controls Middleware (CMW) at CERN – Status and Usage”, ICALEPCS’2003, Gyeongiu, Korea, October 2003.
[4] V. Baggiolini, et al, “JAPC – the Java API for Parameter Control”, ICALEPCS’2005, Geneva, Switzerland, October 2005.
[5] K. Sigerud, et al, “First Operational Experience with LASER”, ICALEPCS’2005, Geneva, Switzerland, October 2005.