AN EXTENSIBLE EQUIPMENT CONTROL LIBRARY FOR HARDWARE INTERFACING IN THE FAIR CONTROL SYSTEM

M. Wiebel, GSI, Darmstadt, Germany

Abstract

In the FAIR control system the SCU (Scalable Control Unit, an industry PC with a bus system for interfacing electronics) is the standard front-end controller for power supplies. The FESA-framework is used to implement front-end software in a standardized way, to give the user a unified look on the installed equipment. As we were dealing with different power converters and thus with different SCU slave card configurations, we had two main things in mind: First, we wanted to be able to use common FESA classes for different types of power supplies, regardless of how they are operated or which interfacing hardware they use. Second, code dealing with the equipment specifics should not be buried in the FESA-classes but instead be reusable for the implementation of other programs. To achieve this we built up a set of libraries which interface the whole SCU functionality as well as the different types of power supplies in the field. Thus it is now possible to easily integrate new power converters and the SCU slave cards controlling them in the existing equipment software and to build up test programs quickly.

INTRODUCTION

As GSI is building up the FAIR [1] project and thus doing renovations all over the facility, it was decided to build up a new control system for the accelerator. This is done as a collaboration project with CERN. As part of the new system, the FESA (Frontend Software Architecture [2]) framework deals with all frontend related tasks.

To integrate our numerous hardware designs with the framework, we decided to build up the FESL (Front-End Support Library) as a lightweight approach to implement flexible equipment interfacing.

This paper sketches the workflow developing a FESA class and describes the challenges resulting from it. As a consequence the requirements to the FESL are depicted, followed by the description of the resulting structure of our library. As a last part we discuss the usage of FESL in the context of the FESA framework und give a short outlook to the future development of the library.

USING THE FRONTEND CONTROLLER

As described in [2] the development of a FESA class follows a specific workflow leading to a ready to use equipment software. After designing the class in an XML based document, one can automatically generate a set of C++ source code frames. These frames are filled with specific implementation and are compiled to a ready to use FESA class. In the deploy unit one or more FESA classes are linked with the run-time core to build an x86-Linux executable. This executable can then be delivered to a front-end computer of choice.

The FESA framework is designed to be flexibly tailored to the broad range of equipment in the accelerator, but due to its rather long development cycles, it lacks the flexibility needed during an early development phase. Especially in our case, as we often have several variants of the devices. Writing test software for different power supplies forced us to walk through the aforementioned development process over and over again. Several only slight differences in the equipment behavior, led to an unwanted overhead of work and time.

REQUIREMENTS TO FESL

To cope with the above described limitations, we tried to decouple the equipment specifics from the implementation of the control system specific parts. Thus we came up with several requirements we had for our Front-end Support Library.

First of all it should unify and simplify the usage of the different power converters we are addressing through our front-end computers, namely the SCU (Scalable Control Unit [3]). Being an industry PC with a bus system for interfacing electronics, the SCU is used as the standard equipment interfacing in the FAIR project. Power supplies and other equipment are connected to it using slaves of an internal bus.

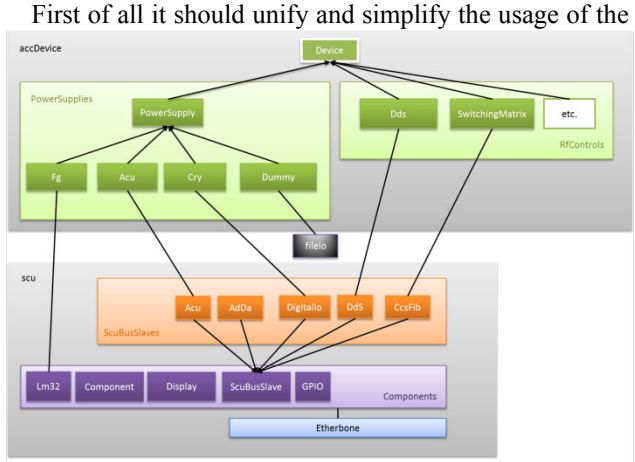

Figure 1: Overall structure of the FESL.

ISBN 978-3-95450-146-5

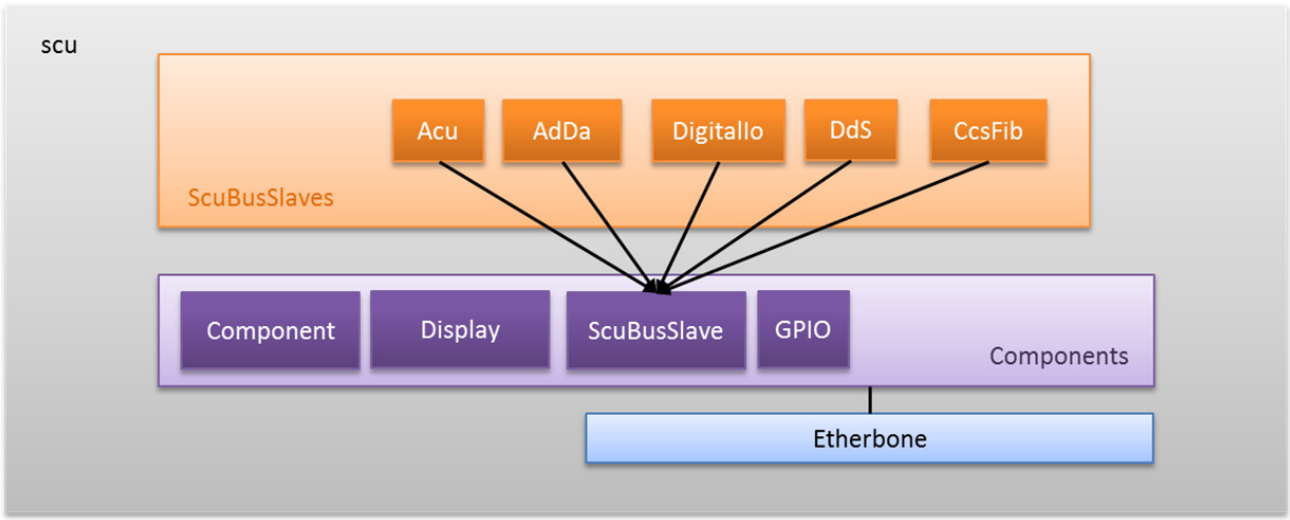

Figure 2: The SCU part of the FESL.

Easy Testing

To ease the testing of the connected devices and to decouple it from the need of embedding the equipment into the control system, we split up the library into two different parts (figure 1): One part to be used from inside the FESA source code frame (*accDevice*) and the other part to be used for dealing with the internals and the slaves of the SCU directly (*scu*). The accDevice part builds upon the scu part in a way, that it combines parts of it to provide a more unified look on it from the FESA class.

No Need to Know About Initialization Details

As a second point, we wanted to hide all the SCU internal communication mechanics from the user of the FESL. As a consequence, we decided to do all the initialization of the different SCU bus slaves as far as possible in an automized way.

Follow the Structure of the Hardware

Furthermore it was important for us, that the library could be used in a very intuitive way. Thus the overall design of the FESL reflects the structure of the hardware found in the context of the SCU.

STRUCTURE OF FESL

Guided by our requirements, we designed the library along the hardware to be represented. This led to an intuitive and straight-forward model of our front-ends. As mentioned before, the library is split up into two parts, the *scu* part and a more FESA oriented *accDevice* part.

SCU Part of the Library

Figure 2 shows the SCU part of the FESL. The blue box represents the etherbone bus [4], which is used to communicate with all components of a SCU. On top of that comes the part of all SCU internal components, starting with the base class for all of them (*Components*). By creating a Components-derived object all needed initializations are done. This makes it easy to create new component classes without deeper knowledge of the SCU. Components which can be created so far are

- Component (base class)
- Display (internal display of the SCU)
- ScuBusSlave (base class for all SCU bus slaves)
- GPIO (general purpose IO)

Inheriting from *Component*, *ScuBusSlave* is the base class for all cards which can be connected to the SCU via the backplane. Again, the base class does all the initialization needed to establish the connection.

AccDev Part of the Library

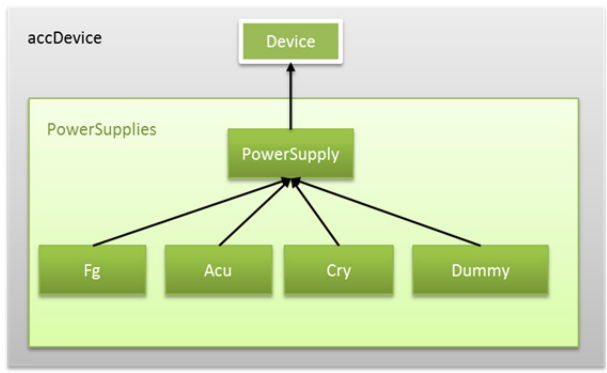

Figure 3: Device part of the FESL.

The *accDev* part builds on the SCU part and combines the different bus slaves to real devices located in the control system. Up to now, two types are supported:

- PowerSupplies
- RFcontrol

ISBN 978-3-95450-146-5

According to the idea that we wanted to be able to control different variants of power supplies with one FESA class (figure 3), they are all derived from the same base class *PowerSupply*. This base class can be used throughout the whole equipment software as an abstract interface. During FESA instantiation it is decided which power supply is actually connected and which implementation should be used.

Fg implements the interface for power supplies driven by a function generator, while *Acu* interfaces the so called Adaptive Control Unit for ramped power supplies. *Cry* again handles a different variant.

The *Dummy* class is used when there is no real power supply to control. It reads from and writes to the hard disk.

INTEGRATION WITH FESA

With the *accDev* part of the FESL at hand, we are now able to easily exchange the different device variants used in a FESA class. This allows us to operate multiple variants of power supplies from within one and the same equipment class.

Instantiating the Power Supply

During start-up, the FESA class reads the type of the power supply from the so called instance file (an XML file containing all the configuration details for a specific device instance). With this information it instantiates the according power supply from the FESL, which is then used in the whole class via the *PowerSupply* interface. So, what we did is injecting different equipment variants using the configuration file.

CONCLUSION

With the introduction of the FESL we were able to enhance our way of developing new equipment software to a level that is now as flexible and lightweight as we wanted. FESA as a framework for equipment interfacing and integration into the control system, is still the way to go, but can now be used with the presented flexibility in the development process.

It's now much easier to quickly write programs to interact with the hardware. Furthermore we can extend our library and thus can use new power supply variants with one and the same existing FESA classes, which saves a lot of time.

REFERENCES

[1] FAIR website: http://www.fair-center.de
[2] Solveigh Matthies et al., "FESA 3 Integration in GSI for FAIR", WPO006, Proc. PCaPAC'14, http://jacow.org/.
[3] S. Rauch et al., "Facility-wide Synchronization of Standard FAIR Equipment Controllers", WEPD48, Proc. PCaPAC'12, http://jacow.org/.
[4] M. Kreider et al., "Etherbone – A Network Layer for the Wishbone SoC Bus", WEBHMULT03, Proc. ICALEPCS'11, http://jacow.org/.

ISBN 978-3-95450-146-5