CHEBURASHKA: A TOOL FOR CONSISTENT MEMORY MAP CONFIGURATION ACROSS HARDWARE AND SOFTWARE

A. Rey*, A. Butterworth, F. Dubouchet, M. Jaussi, T. Levens, J. Molendijk, A. Pashnin
CERN Geneva, Switzerland

Abstract

The memory map of a hardware module is defined by the designer at the moment when the firmware is specified. This memory map is then used by the software developers to define device drivers and front-end software classes. Maintaining consistency between the hardware and the software is critical. In addition, the manual process of writing the VHDL firmware on one side and the C++ software on the other is very labour-intensive and error-prone. Cheburashka is a software tool developed in the Radio Frequency group at CERN which eases this process. From a unique declaration of the memory map, created using the tools graphical editor, it allows us to generate the memory map VHDL package, the Linux device driver configuration for the front-end computer, and a FESA (Front End Software Architecture) class for debugging. An additional tool, Gena, is being used to automatically create all required VHDL code to build the associated register control block. These tools are now used by the hardware and software teams for the design of all new interfaces from FPGAs to VME or onboard DSPs in the context of the extensive programme of development and renovation being undertaken in the CERN injector chain during Long Shutdown 1 (2013-14). Several VME hardware modules and their associated software have already been deployed and used in the SPS RF system.

INTRODUCTION AND BRIEF HISTORY

The Radio Frequency (RF) group is responsible for the accelerating and damping systems at CERN. The high-speed digital electronics for the cavity and beam feedbacks are implemented mainly in the VME form factor. The team of hardware designers works closely with a team of software developers who are responsible for developing the front-end control software for the VME systems using the FESA [1] framework under Linux. The interface between the hardware and software worlds is through the memory map of the device. Historically, memory maps were described in a Microsoft Excel worksheet, which allowed easy editing and sharing of the file via CERN central folders in a format familiar to the hardware designers. The memory map was then entered by hand into the Controls Configuration Database (CCDB) [2] and standard tools from the CERN Controls group were used to generate device drivers. However, this process was fastidious and error prone. A simple copy-paste error or a typo could cause several long hours of software debugging. The need of an automated system to parse memory maps, compute block addresses, generate drivers and software variables has been expressed for several years already.

In 2008 an initial tool was written in C++, running from the command line, which extracted the memory map description from the CCDB and generated a driver wrapper library for use in the FESA class. It was susequently extended to generate parts of the XML design document of a FESA class mapping the registers of the device to datastore fields and interface properties. This XML could then be inserted by hand into the FESA design document. It also generated the C++ code necessary to access the registers, which could be inserted into the FESA class code. Although its use was quite expert-oriented, this application proved to be very time-saving for software developers.

In order to find a global solution, including editing of the memory map in the same application, the software developers proposed the idea of a memory-map template that hardware designers could fill in and where variables would be defined once in a single place. From this master repository, all code relating to the memory map would be generated: firmware VHDL fragments, DSP header files, device driver description files, FESA class design and code and so on. From these wishes several solutions came by themselves:

- An XML file with an XSD template for memory-map formatting
- A user interface based on Java so that it could run on any operating system
- A central repository with versioning support for XML file storage
- An attractive user-friendly interface that would encourage users to move to this new product

The concept of Cheburashka was born in 2009, but due to lack of resources its development progressed only slowly during several years. In summer 2011 a production series of new RF cards for the PS Booster synchrotron awoke the project and work started, focused on VHDL code generation. Early in 2012, driver and FESA code generation followed.

* anthony.rey@cern.ch

ISBN 978-3-95450-139-7

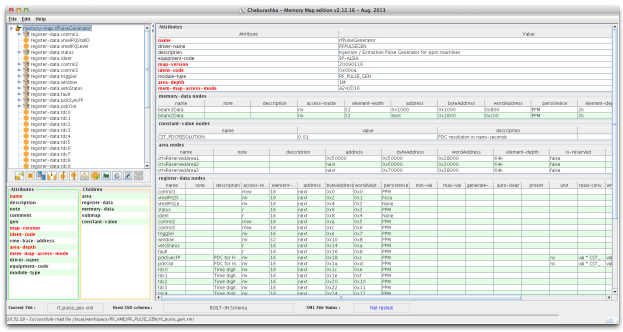

Figure 1: Main interface for Cheburashka application

FUNCTIONALITY

Memory Map Editing

The first challenge was to create a user-friendly memory map editor that could offer, at least, similar features to the hardware designers as Excel. A big effort has been made to create a new application that would be accepted by users.

Creating a new memory map from scratch is quite straightforward. Once you have understood the terminology, you can rapidly add registers and memory blocks to your memory map, and once you have specified their size, their addresses are computed automatically. The memory map is defined as the root element of our design. Each element can have children, i.e. other elements, and attributes, properties describing particular elements. For a given element, some attributes are mandatory and allow identification of the element, others are optional. Once a block is selected, several actions can be performed:

- Duplicate : The selected block and all its children will be cloned and added below.
- Move Up or Down : Blocks can be rapidly shifted up or down depending on the requirements.
- Deletion : Blocks can be removed by a simple click on a button.

For each of these actions, block addresses are automatically re-computed in the background.

Graphical User Interface

Most of the graphical layout had already been thought out and provided before 2011. In order for it to be accepted and used by a majority of the hardware designers, a considerable amount of time has been invested in developing practical functions such as

- A complete ”File” menu (Fig. 2)
- A history of recently used memory-maps
- Undo/Redo functionality (Fig. 3)
- A search tool with extended filters (Fig. 4)
- Keyboard shortcuts for frequently used functions

Copy-Paste commands could not be programmed as such; instead a ”clone element” button is available in the center of the main window.

The colour theme has also been carefully thought out and tested so that the contrast between rows in tables is highlighted without being agressive.

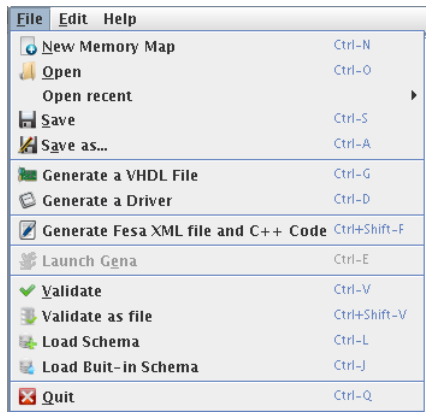

Figure 2: A ”File” menu designed to be familiar from many other software applications.

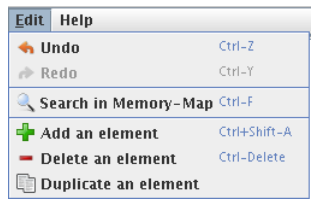

Figure 3: ”Edit” menu, Undo/Redo and Search commands have been often requested and are appreciated.

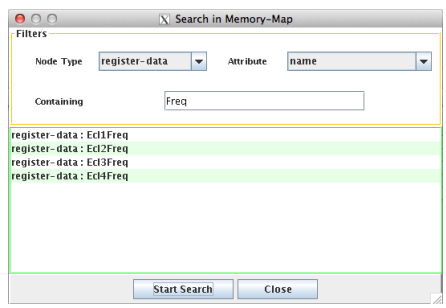

Figure 4: Search Tool, with three optional filter fields: the search can be restricted to a name, an element type and or an attribute

Submaps

A memory map can include another memory map, as an external file. This is known as a submap. Memory block addresses of this submap are then shifted by the submap’s

ISBN 978-3-95450-139-7

address in the parent memory map. It is possible to include the same child memory map several times as long as the name and address are different. It is also possible to import submaps recursively; the only limit is that submap's depth should be smaller or equal to parent's depth.

Code Generation

This important part for both hardware and software designers has been facilitated by the object oriented approach of the data structure. For hardware designers, Cheburashka generates automatically a VHDL register declaration file for their firmware. Thanks to a plugin application called Gena, they can also generate a full VHDL implementation of the memory decoder block. Gena is an external application, written in Python, which takes as input a memory map saved by Cheburashka and adds a number of code generation features. If the hardware design uses a dedicated digital signal processor (DSP), the user can define its memory map in Cheburashka and generate the corresponding C header file for inclusion in the DSP code using Gena.

For software developers, Cheburashka generates

- a definition file for use with the CERN Controls group's device driver generation tool Encore [3],
- a C++ driver wrapper library with a class providing Get-Set methods for each memory block,
- the design, in XML format, of a FESA class, sorting registers and memories depending on their Access mode (Read-Only, Read-Write, Write-Only) and their persistence,
- the C++ classes associated with each FESA class property and their makefiles.

The FESA infrastructure offers tools for importing and integrating external designs and classes.

Persistence

When designing a memory map, the user can specify for each memory block the kind of persistence desired:

- PPM (Pulse to Pulse Modulation): PPM logic is used at CERN on all accelerators in the LHC injector complex, and multiplexes settings by "user" (beam purpose and client). In a PPM system, settings and acquisition make sense only when the associated user is played by the timing system. For each user timeslot, the control system should then load respective settings into hardware and, at the end of the slot, collect acquisitions.
- FESA: The last entered setting will be stored and restored when the front-end computer is rebooted.
- None: No setting persistence will be kept and the hardware value will not be reloaded on reboot.

Before the advent of Cheburashka, software developers needed several days to write the driver and its wrapper, the FESA design and the C++ classes, plus long debugging sessions for tracking typos. With Cheburashka, the process of creating a running test environment from a memory map takes less than one hour.

Memory Map File Management

Cheburashka would not be a complete and successful project if it was not well integrated into CERN's working environment. This means that, once a memory map is ready, it should be made available to other control system tiers working on different operating systems. Generally, hardware and firmware design are performed with tools running on a Windows computer, while software development is done on a Linux computer. A dedicated repository has been created on CERN's SVN [4] service, which adds versioning control and allows multi-platform access.

A memory map created and saved with Cheburashka is an XML file, difficult to decypher when not opened with the software that created it. An XSLT (XSL Translations) file has been written which renders the memory map accessible and easy to read back when opened in a web browser, generating a complete snapshot of the firmware (Fig. 5).

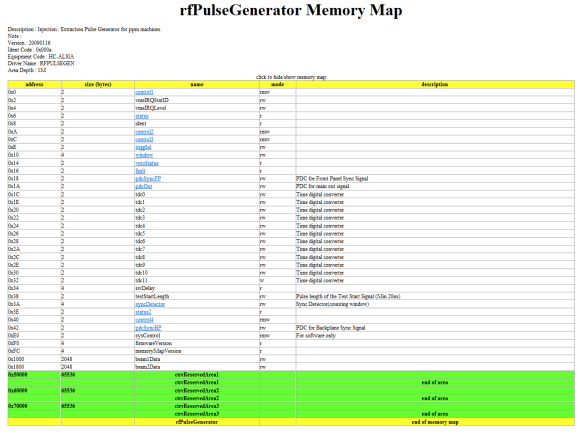

Figure 5: XML Memory map opened in a web browser

DEVELOPMENT AND IMPROVEMENTS

Although Cheburashka is already in operational use and has successfully contributed to optimising resources, there is still room for improvement, and some new features are currently being implemented. Version 3 will be the next big release, and other upgrades will come later on. A complete mechanism for bug reporting and improvement requests, based on Jira [5], was put in place from the beginning. Priorities are assigned as soon as a new item arrives and a development or bug fix is started accordingly.

ISBN 978-3-95450-139-7

Work in Progress for Version 3

As Cheburashka is integrated in a complex software environment, it needs to keep up with the evolution of other tools. Currently, the FESA project has moved from version 2.10 to 3; this is a major upgrade that introduced several changes in design and code. In its current release, 2.12.17, Cheburashka offers an operational FESA 2.10 class generation, and a beta version for FESA 3 generation, soon to become operational.

Keeping up with the latest FESA version will not be the only improvement. In parallel to code and design generation, Cheburashka is also testing a new feature, the possibility to merge an existing version of a FESA design with the latest one generated from the memory map. This will leave all the "non-generated" properties and code untouched and replace only the properties and classes generated by Cheburashka. This option, once fully debugged, should greatly help improve productivity in software development.

Database Declaration

In order to have a fully running test environment, the new hardware device and its driver must be declared in the Controls Configuration Database. Access to this service is limited to trained software developers via a web client. As all the necessary information is already available in Cheburashka for performing such a declaration, the CERN Controls group have agreed to provide write access to the necessary views allowing Cheburashka to automatically declare hardware modules and drivers. This will make the process of creating a test environment from a new memory map even faster.

Acquisition Buffers

Cheburashka is now expected to perform more complex tasks and processes. A concept which has been standardised in the digital hardware of the RF systems is that of acquisition memory buffers, used to store internal digital signals as a sort of virtual oscilloscope. The acquisition memory is split into channels that represent a physical memory resource in the hardware and each of these is split into multiple buffers that represent different data types that are acquired simultaneously, such as I/Q data. This hierarchy gives the hardware designer full flexibility in defining different memory resources, but ensures that they can be read out with a standardised interface. This is a big improvement over previous implementations that were manually implemented on a card-by-card basis with differing interfaces. Acquisition freeze and release can be performed on a channel-by-channel basis allowing an operator to control which channels will be written to at a particular time. The design structure for this functionality is ready and the C++ code for handling this complex mechanism is currently being debugged.

Documentation Generation

The possibility to generate a firmware documentation based on the memory map structure has been recognised as a useful improvement. The structure can easily be put in the table and many options can be provided to allow a very detailed description. It will be possible to generate a printable document, whose output format would probably be in LaTeX.

CONCLUSION

Cheburashka is first and foremost a team effort and an example of what a good collaboration between software and hardware developers can produce. The software team took charge of the design and software development and hardware team invested a lot of effort in requirements definition and debugging sessions. They also made the effort of moving to a new, promising, yet incomplete, tool. Software developers have gained a substantial amount of time thanks to this tool, now that repetitive simple tasks such as driver generation, FESA class design and C++ code generation are performed automatically. Incoming upgrades will bring more automated processes, improving overall efficiency and allowing them to focus on other tasks.

The project has been integrated in a complete environment, maintained by the CERN Controls group and IT department, including SVN repositories, Jira issues tracking tool and Eclipse [6] plugins for project development, build and release configuration.

Most importantly, users have migrated to this new solution, causing them to change their working habits: with a reasonable effort, most of them have adopted Cheburashka rapidly and have been quite active in reporting bugs and desired improvements. These requests have been fulfilled within short delays; our reactivity helped gaining their trust. All these facts put together allow us to conclude that Cheburashka is on the right track for becoming a successful project.

REFERENCES

[1] M. Arruat et al, "Front-End Software Architecture", ICALEPCS07, Knoxville, Tennessee, USA, 2007

[2] Z. Zaharieva & M. Martin Marquez, "Database foundation for the configuration management of the CERN accelerator controls systems", ICALEPCS2011, Grenoble, France, 2011, MOMAU004

[3] J. D. Gonzalez Cobas, "Status and how-to for encore as replacement for dgII", http://indico.cern.ch/conferenceOtherViews.py?view=standard&confId=167545, CERN, 2012

[4] http://subversion.apache.org

[5] https://www.atlassian.com/en/software/jira

[6] www.eclipse.org

ISBN 978-3-95450-139-7