openMMC: AN OPEN SOURCE MODULAR FIRMWARE FOR BOARD MANAGEMENT

H. A. Silva*, G. B. M. Bruno, LNLS, Campinas, Brazil

Abstract

openMMC is an open source firmware designed for board management in MicroTCA systems. It has a modular architecture providing decoupling between application, board and microcontroller-specific routines, making it useful as a base for many different designs, even those using less powerful controllers. Despite being developed in a MicroTCA context, the firmware can be easily adapted to other hardware platforms and communication protocols. The firmware is based on the FreeRTOS operating system, over which each monitoring function (sensors, LEDs, Payload management, etc) runs its own independent task. The OS, despite its reduced footprint, also provides numerous tools for reliable communication among the tasks, controlling the board efficiently.

INTRODUCTION

LNLS Beam Diagnostics team is currently developing the Sirius' Beam Position Monitor (BPM) electronics and has adopted the MicroTCA.4® standard by PICMG [1] in its boards designs [2].

The main board on the electron BPM system is the AMC FMC Carrier (AFC) [3], which is a general purpose FPGA board that hosts up to two mezzanine cards in FMC form factor. Those smaller cards carried by AFC can be implemented to have many different applications (e.g. fast digitizers, SFP modules, data pre-processing modules, RS485 communication). In the BPM application, fast digitizer FMCs will be used in order to read the BPM analog signals.

AMC boards, as the application boards are called in the MicroTCA system, must have implemented a Module Management Controller (MMC), which usually is a microcontroller responsible for monitoring the board health and acting as a communication channel between the system manager and application using Intelligent Platform Management Interface (IPMI) [4].

openMMC was created to be an open source (using GPLv3 license) modular and generic firmware, easily portable to other platforms. It runs over FreeRTOS, which gives the developer a wide set of tools to implement complex monitoring functions or advanced hardware control. Given its modular independent structure, it is possible to use the firmware in applications outside MicroTCA environment with little effort, changing the communication protocol in its lower layers, for example.

The project development is being versioned in a GitHub repository [5], using pull requests and issues tracking as its main collaboration tools.

* henrique.silva@lnls.br

MOTIVATION

The development of openMMC started after some unfruitful tests with the available open source MMC implementations. Those firmwares were developed to run on specific target boards, requiring a substantial effort in order to port them to AFC's hardware and begin a functional evaluation process.

After some attempts, it was clear that porting the code basically meant to rewrite it from scratch, given that its low level driver and application functions were deeply intertwined.

The hardware flexibility offered by MicroTCA was not being accompained by its MMC firmware architecture, since each board implementation had to recreate the managing firmware. openMMC was thought to be the hardware independent firmware that could meet this need.

FreeRTOS

FreeRTOS [6] is a popular open source real time operating system for embedded controllers that has already been ported to a wide range of CPU architectures.

It features a preemptive scheduler that allows the application code to run multiple tasks in parallel with a single core. The scheduler decides which task will run based on its priority. More important tasks are always executed first, whilst blocking the lower priority ones. If one or more tasks are on the same priority level, a round-robin time slice method is applied, ensuring that all of them are executed within its time limits.

Most of OS-native tools used in communication between tasks are also implemented on FreeRTOS (e.g. semaphores, software timers, queues). The developer can also use some unique functions provided, such as direct-to-task notifications, event groups and co-routines.

Except for task creation, all tools on FreeRTOS can be stripped from its compilation, reducing resource usage, thus enabling use of cheaper and lower-power microcontrollers.

FreeRTOS project uses a modified GPLv3 license, which allows the user to implement its application on top of the OS without having to publish proprietary code [7].

FIRMWARE STRUCTURE

The firmware was structured in order to be easy to upgrade and port to different boards and controllers. Therefore four different abstraction layers are implemented: *Application*, *Hardware Abstraction*, *Port* and *Driver*, arranged as in Fig. 1.

Application

The Application layer holds high-level tasks responsible for deciding which action will be taken based on the in-

ISBN 978-3-95450-189-2

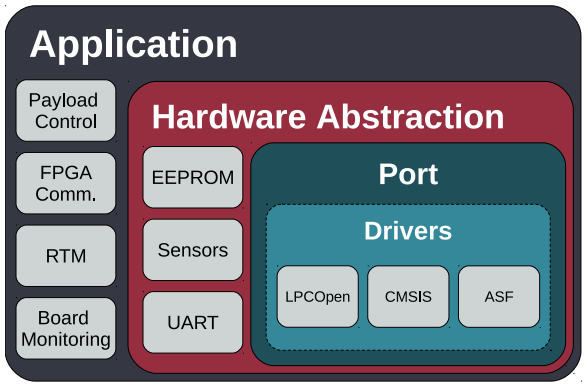

Figure 1: openMMC firmware structure.

formation provided by the lower level layers. Most of the functions implemented on this level are state-machine based, regurlarly checking and actuating on the board status.

Hardware Abstraction

In the Hardware Abstraction Layer (HAL) lies all the functions that interfaces with the board peripheral hardware, making the internal calls to the IC registers transparent to the caller for example. Since any board can benefit from a module in this layer, they are all stored together inside "modules" folder.

An exception in the HAL is the IPMI Protocol module, that does not target any specific peripheral hardware. Instead, it is responsible to manage IPMI protocol messages internally, decoding packets, asserting checksums and building responses accordingly.

The following modules were already implemented and tested in LNLS' AFC board:

- IPMI Protocol
- Watchdog Timer
- SCANSTA111 JTAG Switch
- ADN4604 Clock Crossbar Switch
- AD84XX DAC
- AT24MAC EEPROM
- 24xx64 EEPROM
- Hotswap Handle
- LM75 Temperature Sensor
- MAX6642 Temperature Sensor
- INA220 Voltage and Current Sensor
- HPM Upgrade
- PCA9554 I/O Expander
- Si57X Oscillator
- UART Debug interface

Port

The Port layer serves as a connection between *Driver* and the upper layers. Its purpose is to mask all functions provided by the controller drivers so that the upper layers only call generic functions defined in this level, as demonstrated in Fig. 2.

This layer presents a few restrictions regarding its structure. Developers in charge of implementing new hardware

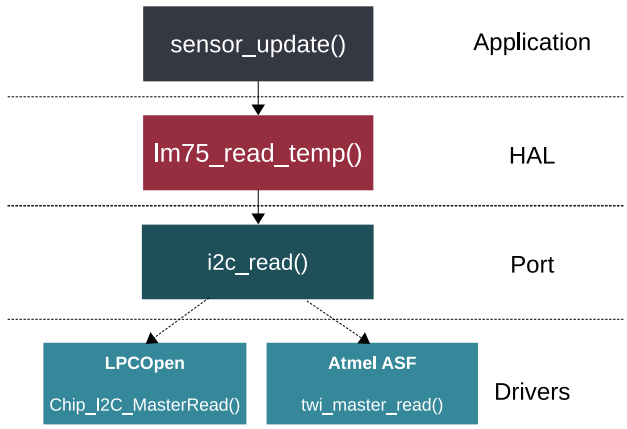

Figure 2: openMMC sensor update function call sequence.

drivers must follow the port layer functions' signature in order to maintain compatibility throughout the firmware.

Currently, there is only one port maintaned in mainline openMMC, which targets LPC17 family chips. Development of a port to ATxMega128 CPU has been initiated by GSI [8].

Drivers

The lowest layer is where the microcontroller peripheral drivers are implemented, accessing directly the hardware to control the outputs.

BOOTLOADER

Having a small footprint makes it possible to store more than one image of the running firmware, easing the system's upgrade process. Taking advantage of this property, a Bootloader scheme was developed in order to manage the controller's memory sections and reserve a portion of the ROM to store an updated firmware. An example of the internal memory divison in a LPC1764 controller can be seen in Fig. 3.

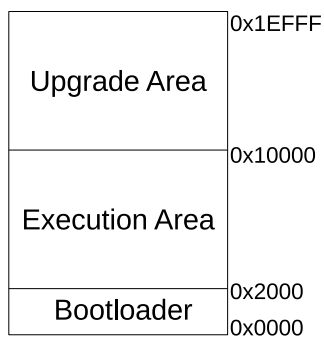

Figure 3: LPC1764 ROM organization with openMMC.

The new firmware is sent to the MMC broken in small packages using HPM update protocol [9], defined by PICMG. The controller then writes page by page the new firmware in the reserved ROM section and also adds an update flag to the bootlader, indicating that this firmware is the more recent version and should be moved to the proper section in order to be executed.

ISBN 978-3-95450-189-2

The bootlader is always executed first in power-up sequence, checking whether there is a new firmware to write into execution section. If not, it just moves the program execution flow to the starting point of the user code area.

BUILD AUTOMATION

CMake is a software designed to automate build systems [10], widely used in projects where the compilation chain has more than one build target and needs to manipulate multiple sources. It implements its own scripting language that allows the user to fully customize each build without having to worry about sources dependencies.

Using this tool with openMMC allows developers to fit the firmware to the board hardware just by adding new modules in the compilation list present on each board port folder. The user is also able to set new compilation flags or link new libraries almost effortless.

BOARD PORTING

openMMC also allows the user to easily port the firmware to a different board with other peripheral hardware configuration while keeping the target controller.

Since all peripheral hardware drivers are implemented as modules in the *Hardware Abstraction Layer*, only the `CMakeLists.txt` file inside the board folder have to be changed. In this file, all modules needed in each hardware configuration are selected based on a list parameter. When CMake is called to generate the automatic Makefiles, it parses this argument and includes all selected modules sources and dependecies in the compilation chain.

CODE SIZE

openMMC has a small memory footprint, which makes it perfect for embedded environments that traditionally have limited resources. Table 1 presents openMMC's average RAM usage and code size in ROM after compiling in both minimal and standard configurations for AFC BPM board, using GCC's optimization flag -Os.

Table 1: openMMC Resources Usage

Configuration	RAM [kB]	ROM [kB]
Minimal	5.54	20.91
Standard	8.07	29.69

CONTINUOUS INTEGRATION

Every modification in the application layer must be tested across all boards and controllers ports to assert that the firmware remains compatible.

Validating that the firmware compiles with every variant option becomes very difficult as the ports count increases. Using a Continuous Integration tool allows the maintainer to run dedicated tests after each commit on the main repository and automatically reports if the lattest change has made any port unusable. Travis C.I. [11] uses a simple script written in its own language to install the needed toolchains. Tracking the latest commits on the repository branches, allows it to perform build tests on them, ensuring that the latest modification is compatible with all board and controller ports.

DOCUMENTATION

The firmware documentation is written using Doxygen style [12]. This way, the documentation format is completely automated and can have many different output formats such as HTML, LaTeX, XML, RTF, etc.

By hosting the firmware code in GitHub repository, one is able to use a feature called GitHub Pages, in which a small website for generic use can be hosted simply by adding a new branch named *gh-pages* in the main repository. openMMC documentation has been uploaded to GitHub pages, but it is still under development [13].

Using this tool simplifies code documentation update, since it just needs to be regenerated by Doxygen after each change and pulled to its respective GitHub Pages branch.

ACKNOWLEDGMENTS

The authors would like to thank K. Macias from Creotech Instruments SA for the extensive firmware debug and P. Miedzik from GSI for the initial discussions in the firmware structure definition and by suggesting FreeRTOS as a base for the project.

REFERENCES

[1] *PICMG MicroTCA Specification*, https://www.picmg.org/openstandards/microtca

[2] D. O. Tavares *et al.*, "Development of an Open-Source Hardware Platform for Sirius BPM and Orbit Feedback", in *Proc. ICALEPCS'13*, San Francisco, October 2013, p. 1039, 2013.

[3] AMC FMC Carrier Project, http://www.ohwr.org/projects/afc

[4] *Inteligent Platform Management Interface*, http://www.intel.com/content/www/us/en/servers/ipmi/ipmi-home.html

[5] openMMC GitHub Repository, https://github.com/lnls-dig/openMMC/

[6] FreeRTOS Project Page, http://www.freertos.org/

[7] FreeRTOS modified GPLv3 license, http://www.freertos.org/license.txt

[8] GSI MMC implementation repository, https://github.com/qermit/JAMMCI/

[9] Hardware Platform Management Overview, https://www.picmg.org/openstandards/hardware-platform-management/

[10] CMake Project, https://cmake.org/

[11] *Travis Countinous Integration tool*, https://travis-ci.org/

[12] Doxygen Project, http://www.stack.nl/~dimitri/doxygen/

[13] openMMC Documentation at GitHub Pages, http://lnls-dig.github.io/openMMC/

ISBN 978-3-95450-189-2