PORTING OPENMMC TO STM32 MICROCONTROLLERS FOR FLEXIBLE AMC DEVELOPMENT

M. B. Stubbings[*], E. P. Juarez, L. Stant, Diamond Light Source, Didcot, UK
A. A. Wujek[†], CERN, Geneva, Switzerland

Abstract

Diamond Light Source has chosen the MicroTCA platform for high performance data acquisition and controls as part of the Diamond-II 4th generation light source upgrade. One requirement is the ability to create custom Advanced Mezzanine Cards (AMCs) for signal conditioning and interlock support. To facilitate this, a Module Management Controller (MMC) is required to negotiate payload power and communications between the AMC and MicroTCA shelf. A popular open-source firmware for controlling such a device is OpenMMC, a project from the Brazillian Light Source (LNLS), which employs a modular approach using FreeRTOS on ARM microcontrollers. Initially, openMMC supported the NXP LPC series of devices. However, to make use of Diamond's existing ST Microelectronics (STM32) infrastructure, we have integrated a CERN fork of the project supporting STM32 microcontrollers into openMMC. In this paper, we outline our workflow and experiences introducing a new ARM device into the project.

INTRODUCTION

MicroTCA is an open standard for constructing high performance computer systems in a small form factor [1]. It defines a number of hot-swappable modules, which when connected to a backplane provide power, cooling, management and user functionality. At its core are Advanced Mezzanine Cards (AMCs), which are modules that provide the processing and I/O required to implement an application. Board management and communication with the rest of the system is handled by a Module Management Controller (MMC), which is typically implemented as a low-power Microcontroller Unit (MCU) on top of the AMC.

Electronic Keying

When an AMC is inserted into a MicroTCA shelf, the on-board MMC must pass an Electronic Keying (E-Keying) stage before it is allowed access to payload power and communications [2]. The main management module, known as the MicroTCA Carrier Hub (MCH), leads this process to determine the electronic capabilities of the inserted module and its compatibility with the crate. If the module is found to be incompatible, then it will be rejected from receiving power and unable to communicate with the rest of the system. Through this mechanism, the crate ensures that it protects itself and all Field Replaceable Units (FRUs) from mis-operation and power supply overloading.

Diamond Light Source

At Diamond Light Source [3], we are undergoing significant changes to our infrastructure as part of the Diamond-II 4th generation light source upgrade [4]. One aspect of these changes includes the use of MicroTCA for high performance data acquisition, processing and control. In the majority of cases, AMCs will be purchased off-the-shelf from vendors. However, for a small number of high speed signal processing applications, we would like to be able to produce our own AMCs.

A major challenge presents itself when attempting to recreate the behaviour of an MMC. There are a number of complex processes, such as E-keying, that the MMC must perform in order for the AMC to operate in a MicroTCA system. Additionally, the MMC's firmware is dependant on the target controller being used and the design of the AMC. Therefore, in each application the firmware would need to be revised to account for the change in design.

Together, these issues present a significant amount of work required to support only a small number of use cases. Through this realisation, we looked for an alternative solution.

OPENMMC

openMMC [5] is an open source, hardware independent firmware designed to carry out the operations of an MMC in a MicroTCA system [6]. Its modular architecture allows for flexible configuration of the sensors, communications and controller used for a target board. It employs the FreeRTOS [7] operating system for independent task management and advanced hardware control. Whilst it was initially released with support for the NXP LPC17xx family of chips, there is scope to port the project onto other microcontroller architectures.

STM32 Support

A team at CERN [8] created a fork of the project that provides support for the STM32 family of microcontrollers. However, during development they faced several issues with the portability of the code so made adjustments to the core architecture to resolve this. As a result, the fork was no longer compatible with the LP17 MCUs and the flexibility was lost. At Diamond, we are interested in combining our existing ST Microelectronics infrastructure with openMMC to produce new AMCs. As such, we created this project to complete the integration work that CERN started, thereby introducing a new ARM device into the firmware.

[*] michael.stubbings@diamond.ac.uk
[†] formerly associated with

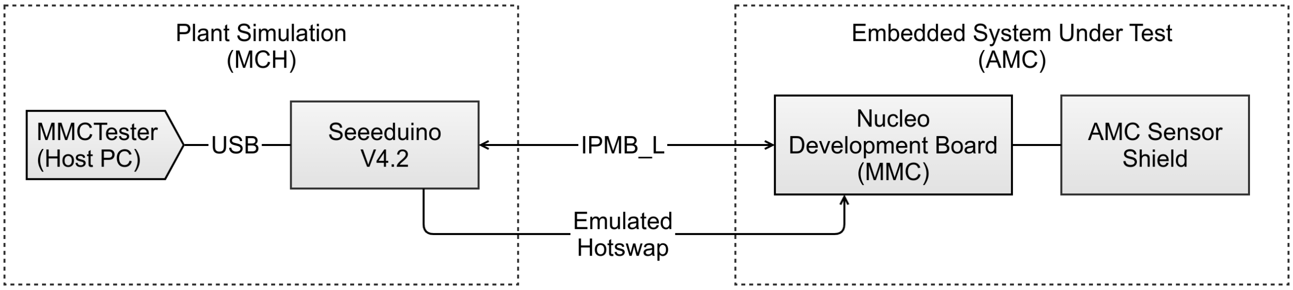

Figure 1: Block diagram outlining the hardware-in-the-loop test environment

WORKFLOW

The project was broken up into two primary workflows: The first outlined the creation of a test environment to rapidly deploy the firmware and test the integration work as it was completed. This was based around an STM32 Nucleo development board, which had its own board implementation created in openMMC. The second captured the integration work required to port the code to openMMC. An initial approach was laid out, which involved an investigation of the CERN fork, followed by a more detailed breakdown and reform of the commits. To conclude the project, a meeting was organised with the maintainers of openMMC to discuss the changes and seek approval before submitting the pull request.

TEST ENVIRONMENT

A harware-in-the-loop test environment was created to allow for rapid deployment of the openMMC code during the reform. This was realised as a hardware stack that simulates an AMC connecting to the MicroTCA shelf and communicating with the MCH. Low-cost development boards were used to reduce workload and provide a widely accessible testing solution.

Figure 1 grants an overview of the test environment, with the hardware stack being shown in Fig. 2. The MMC is represented by a Nucleo-F303RE development board [9] where openMMC will be deployed. Alongside this, a custom sensor shield provides interfaces to all of the modules that openMMC supports (e.g. I2C, LEDs, ADCs, Hot Swap handle etc.). Four breakout headers allow an engineer to connect and test each of the supported I2C sensor modules (e.g. LM75, ina220, max6642 etc.). Using this design, an engineer can create several board implementations in openMMC and verify their operation without ever touching a real MicroTCA system. Together, the Nucleo development board and the sensor shield represent a complete AMC and the embedded system under test.

Mounted between these is a Seeeduino V4.2 [10], which is an Arduino-compatible board based on the ATmega328P MCU. A switch allows the system supply voltage to be adjusted between 3.3 V and 5 V, which is required to match the 3.3 V operating voltage of the Nucleo's I/O pins. The board serves as the primary power supply for the system, receiving power from a microUSB port and distributing it to the stack via the interconnected power pins. Its primary function is to represent the physical component of the MCH by providing an I2C interface between a connected computer and the Nucleo development board. Requests and control signals can be sent to the Seeduino via a host PC, which will either be processed or forwarded on to the Nucleo MMC. Responses can then be sent back to the host PC for display or further processing.

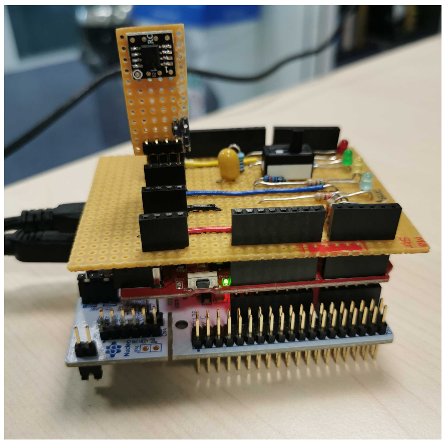

Figure 2: Harware Stack with TMP75 temperature sensor.

MMCTester

MCH commands are simulated using an in-house utility package called MMCTester. This utilises two communication interfaces: A USB UART connection between the host PC and Seeduino, and an I2C connection between the Seeduino and Nucleo. In a MicroTCA system, the MMC and MCH communicate using the Intelligent Platform Management Interface (IPMI). This is a set of standardized specifications that provide management and monitoring capabilities irrespective of the host's operating system. In MMCTester, a python package called pyIPMI provides the libraries required to send and receive commands over the IPMI protocol.

User interaction is provided by a terminal user interface that displays the FRU data and current state of the attached sensors. As with a real system, the application will obtain the FRU data of the Nucleo MMC and display some of its properties to the user to verify this process. Afterwards, the user can control the state of the hot swap handle and confirm the operation of the sensors defined within the FRU data.

Board Implementation

Initially, the board was implemented using CERN's fork of openMMC. The idea was to establish an initial working state, so that any errors made during the integration would be flagged up during testing. CERN's branch contained their own board implementation, an enhanced Rear Transition Module (eRTM), which included an STM32 chip. Since it was from the same family as the chip on the Nucleo board, we were able to use their implementation as a model for our own.

In this sense, the pin mapping and I2C configuration files were setup with the correct ports, but required modifications to the function of each pin/channel. The drivers for any payloads were removed and a dummy 'power good' function was written in place for testing. This used the sensor boards ADC channel to monitor when power had been applied to that pin. The Sensor Data Record (SDR) list, used during runtime, was cleared and again replaced with information on our power good sensor. Finally, the CMakeLists.txt file that contained the boards configuration of openMMC was modified to include our desired list of modules and driver code.

CODE INTEGRATION

The first task was to identify at which point in the commit history did the work that CERN carried out first take place. With this information, we could pinpoint all of the changes that were related to providing STM32 support (ignoring the rest) and review them subsequently. A clone of the CERN's fork was created internally and new branch was produced from that to perform the integration. The upstream openMMC repository was added as a new remote and by using the 'git merge-base' command, we identified the common ancestor between our local branch and the upstream master.

154 commits were shown to be made on top of the common ancestor providing the new support. By inspecting the modified files, we could quickly identify those that were not required, which if we could discard would significantly reduce our workload. The git project recommends git-filter-repo as a tool for rewriting history, which provided us with this capability. Using the common ancestor as our starting point, we removed all commits relating to two directories: port/board and boot/. The port/board directory contained CERN's eRTM board implementation, which was not required for the support and was likely to fail after the code had been generalised. The boot directory similarly contained modifications for the eRTM, which were board specific and overrode the original bootloader code.

The remaining 84 commits were analysed and those suspected of requiring modification or removal were documented. Those that were not documented, included the core STM32 library code and drivers written for the modules in openMMC. These were kept by default.

Board Specific

The majority of outstanding commits were based around code supporting the eRTM board. The design included an FPGA, which required a number of interfaces and drivers to run. On review, we found additions to the link drivers, UART, SPI, and hotswap modules to support this. Typically, we found that any board specific additions were isolated to a single commit, so could be dropped without affecting any of the base code. However, in some cases we discovered that core functions had been removed or bypassed entirely to prevent tasks from blocking or producing errors at run-time. In one such instance, the LED module had been bypassed using a series of return statements in its functions. Unlike the other modules, the LEDs are a forced component in openMMC so could not simply be omitted via a change in the configuration. The eRTM board did not include any LEDs in its design, so would have likely caused errors when attempting to run the software. In cases such as these, the commits were dropped as supporting the board was outside of the scope of this project.

IPMB Configuration

The IPMB configuration was found to be hard coded, forcing the user to use a specific I2C port for IPMI communications. This was traced back to the STM32 I2C driver where we found the source in the initialisation code. On review, there were a number of opportunities for the code to be optimised, so we made the decision to rewrite it.

Each I2C interface was defined in a structure that contained a number of properties associated with it. One of these was an ID number, which was represented by an enumerator. In the initialisation routine, a unique input was used to determine which port was to be initialised using a case statement. By replacing this input with the ID, the readability of the code improved and we found that we were able to reduce some of the duplicated lines within the function.

In the case of I2C2, we found additional code for setting it up as a slave for IPMB. This required a flag to be set in the configuration, specifying that the port should be used in this way. Despite being the only port with this option, there was a number of slave specific routines that ran after the case statement for any port flagged in this way. To generalise this, we extracted all of the slave code and created a second routine for slave initialisation. We then used a constant defined in the board configuration to allow flexibility on which port would be set up in this way. As a result, any port that was defined as an IPMB channel in the board configuration was reinitialised as a slave in the I2C init.

At the end of the routine, there was some initialisation code that was common to all I2C ports. As the function had now been broken up into two initialisation routines, the code was extracted into its own common init. The final architecture contained three routines: one for setting up an I2C master, one for setting up an I2C slave, and one common to both to complete the initialisation.

HISTORY

Having reviewed all of the commits, a consideration was made for the outstanding history. With commits being modified, dropped, extracted or broken down, the history was disjointed and the messages were no longer reflective of their contents. To resolve this, the list of modified files were reviewed and a new set of commits were drafted. The intention was to compress the history into a single commit, then break it up into a small number of commits that highlighted the core components of the support work. These were as follows:

- Add support for STM32F30x microcontroller
- Add I2C support
- Add ADC support
- Add OpenOCD configuration

TESTING

Frequent testing was carried out over the course of the code review using the hardware-in-the-loop test environment. This helped to verify that the work done to the I2C drivers and other modules were functioning as commits were reduced. A notable issue was discovered early on, where the firmware would freeze after initialisation with no clear indication as to the cause. It was later found to be a problem in the IPMI module, where the FreeRTOS stack size had been hard coded for the original LPC17 controllers. As openMMC receives more controller support, the stack sizes will need to be adjusted on a case-by-case basis. To resolve this, it was made into as a constant and added to the IPMB header file to make it easier to adjust and a single source of truth.

Once all of the modifications had been made, a final push was made to test all of the components in openMMC. The LEDs, IPMB, UART debugging and hotswap moules were all active during the code review, so were tested by default. This left the sensor I2C port, ADC module, and each of the supported sensors themselves. Small breakout boards were made for the sensors, which were plugged into the sensor shield and tested alongside the sensor I2C port. The ADC was testing using the power good signal, which was set up as a payload when the board was first implemented. By adding each module consecutively to the list of target modules, we were able to build up a complete profile of openMMC covering the majority of features and proving them complete.

LICENSING

openMMC is an open source project, so it is important that the code being used comes under a compatible licence. When porting the code, we came across a number of instances where the licences were not suitable to be used in this application.

The first instance was within the STM32 linker script. When CERN had generated the code, it had been done so using Atollic TrueSTUDIO, which is an enhanced C/C++ IDE. Code generated in this way comes with a licence that restricts it from being used outside of an Atollic TrueSTUDIO project. Fortunately, as the linker is a core component of an STM32 project it can be reproduced using the STM32CubeIDE. Files generated from this use a GPL compatible license, which is required to be used in openMMC.

After the code had been tested, it underwent a review from the maintainers of openMMC. During this, they discovered that the core standard peripheral libraries contained a licence that was not compatible with the project. The standard peripheral libraries provide the core drivers for STM32 devices, but have since been deprecated. Instead, a more modern approach is to use the Common Microcontroller Software Interface Standard (CMSIS) from Arm, with the STM32 Hardware Abstraction Libraries (HAL) for device specific control [11]. As with the linker, these libraries can be generated from the STM32CubeIDE and contain a GPL compatible license. However, in replacing the core libraries, the names and inputs of the back-end functions have all been updated. Therefore, additional effort is now required to adapt the existing code to use the new driver functions.

CONCLUSION

openMMC is an effective non-commercial solution for designing in-house AMCs and replicating MMC behaviour. In this project, we integrated a CERN fork of openMMC supporting STM32 microcontrollers, and verfied it using a low-cost hardware-in-the-loop test bed. Introducing a new Arm device into the project has created new opportunities for Diamond to produce a number of specialist AMCs. As part of the Diamond II program, openMMC will be adopted to create a number of AMCs for high speed signal processing applications.

REFERENCES

[1] PICMG MicroTCA Overview, `https://www.picmg.org/openstandards/microtca/`

[2] VadaTech MicroTCA Overview, `https://www.vadatech.com/media/pdf_MicroTCA_Overview.pdf`

[3] Diamond Light Source, `https://www.diamond.ac.uk/Home.html`

[4] Diamond II programme, `https://www.diamond.ac.uk/Home/About/Vision/Diamond-II.html`

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

[6] H. A. Silva and G. B. M. Bruno, “openMMC: An Open Source Modular Firmware for Board Management”, in *Proc. PCaPAC’16*, Campinas, Brazil, Oct. 2016, pp. 94–96. `doi:10.18429/JACoW-PCAPAC2016-THPOPRPO04`

[7] FreeRTOS, `https://www.freertos.org/`

[8] CERN, `https://home.cern/`

[9] Nucleo-F303RE, `https://www.st.com/en/evaluation-tools/nucleo-f303re.html`

[10] Seeeduino V4.2, `https://www.seeedstudio.com/Seeeduino-V4-2-p-2517.html#`

[11] CMSIS documentation, `https://arm-software.github.io/CMSIS_5/General/html/index.html`