OpenIPMC: a free and open source Intelligent Platform Management Controller Software

Luigi Calligaris, André Cascadan, Luis E. Ardila-Perez, Bruno Casu, Alison França da Costa, Ailton Akira Shinoda, Lucas Arruda Ramalho and Oliver Sander

Abstract—**OpenIPMC is a free and open source software designed to implement the logic of an Intelligent Platform Management Controller (IPMC). An IPMC is a fundamental component of electronic boards conformant to the Advanced Telecommunications Computing Architecture (ATCA) standard, currently being adopted by a number of high energy physics experiments. The IPMC is responsible for monitoring the health parameters of the board, managing its power states, and providing board control, debug and recovery functions to remote clients. OpenIPMC is based on the FreeRTOS real-time operating system and is designed to be architecture-independent, allowing it to be used in firmware designed for a variety of microcontrollers. Having a fully free and open source code is an innovative aspect for this kind of software, enabling full customization by the user. In this work we present the features and structure of OpenIPMC as well as its example implementations on Xilinx Zynq UltraScale+ (ZynqUS+), Espressif ESP32 and ST Microelectronics STM32 architectures.**

Index Terms—**PICMG, ATCA, IPMC, Electronic board management**

I. Introduction

The Advanced Telecommunications Computing Architecture (ATCA) standard [1] is developed by a consortium of leading computer hardware manufacturers known as the PCI Industrial Computer Manufacturing Group (PICMG) [2]. This standard defines mechanical, electrical and functional design rules, connector pin assignments and communication protocols to be used in the design of electronic boards and their housing shelves for industrial computing applications. The rules aim to guarantee a high availability and reliability of the deployed systems, an objective which is achieved with the aid of a sophisticated Hardware Platform Management (HPM) [3] infrastructure. The ATCA standard is widely adopted in the telecommunications industry and its use extends to a broader range of applications, such as medical equipment [4] secure networking, military electronics and large physics experiments [5].

L. Calligaris (corresponding author, email: luigi.calligaris at cern.ch), A. Cascadan and B. Casu are with the Scientific Computing Center (NCC) of São Paulo State University (UNESP), Rua Dr. Bento Teobaldo Ferraz, 271, São Paulo - SP, 01140-070, Brazil.

A. França da Costa and A. A. Shinoda are with the Electrical Engineering Department (FEIS) of São Paulo State University (UNESP), Av. Professor José Carlos Rossi, 1370 Campus III, Ilha Solteira - SP, 15385-000, Brazil.

L. A. Ramalho is with Exact and Earth Sciences Department (FACET) of Mato Grosso State University (UNEMAT), Rua A, S/n, Bairro São Raimundo, 78390-000, Caixa Postal 92, Barra do Bugres, Mato Grosso.

L. E. Ardila-Perez and O. Sander are with the Institute for Data Processing and Electronics (IPE) of Karlsruhe Institute of Technology, Hermann-von-Helmholtz-Platz 1, D-76344 Eggenstein-Leopoldshafen, Germany.

This work is supported by the Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), through grants number 18/18955-0 and 17/16245-3.

The focus on high availability and reliability, the large data bandwidth offered by the shelf backplane, the availability of large electrical power, good thermal dissipation, and the possibility to insert and remove boards and other components in a running system (*hot swap*) make ATCA systems very attractive for use in high energy physics experiments [6]–[12], where the requirements for very large detector read-out rate, low latency, high availability and compact physical size of the back-end systems of the detectors push the limits of current technology. Examples of ATCA boards used in High Energy Physics are the Serenity [13] and Apollo [14] boards, which are going to be used in the back-end of the tracker detector of the CMS experiment [15] following its Phase-2 upgrade.

Each electronic board compliant to the ATCA standard is required to host an Intelligent Platform Management Controller (IPMC), which is typically implemented using a microcontroller running a firmware that implements the IPMC functions. Many IPMC solutions have been proposed over the years, some of which commercial in nature [16], [17] and others non-commercial in nature [18]–[21]. These solutions employ a firmware specifically written for their intended target microcontroller, consequently making it tedious to migrate the firmware in the event those parts become obsolete. Furthermore, vendor tool-chains supporting those older parts tend to be excluded from new software updates, making them rely on the support of legacy operating systems, which can be difficult to operate over the lifetime of the target boards. Lastly, closed-source implementations - including commercial ones - often pose significant bureaucratic barriers to developers by requiring their institutions to sign Non-Disclosure Agreements (NDA). This may cause lengthy approval processes by the home institution of the researchers and prevent non-staff participants (e.g. students) from participating officially to the project. These are some of the reasons that motivated us to look into developing a free and open-source solution.

II. The ATCA Shelf

An ATCA shelf is a standardized form factor chassis accommodating Field Replaceable Units (FRU). These can be various "intelligent" (i.e. capable of mutual coordination) components such as cooling fan trays, power supplies or user-designed electronic boards, which are the focus of our development. FRUs need to be compliant to a set of mechanical, electrical and interface specifications, as defined in the PICMG standard, to ensure their proper inter-operation. From the point of view of an ATCA electronic board, the resources made available by a shelf are:

Power: Each board is powered by a two-channel, redundant, -48 V rail.

Cooling: Redundant fan trays drive an air stream to remove the heat generated by the electronics.

Data and clock bus: ATCA specifies a number of backplane topologies, supporting the transmission of synchronization clock signals and high speed links between the boards.

IPMB: The backplane exposes to all FRUs a dual-redundant two-wire bus, compatible with the I^2C protocol with a signalling level of 3.3V. The two buses are named Intelligent Platform Management Bus A and B (IPMB-A and IPMB-B), also referenced collectively as IPMB-0. The two buses are used by the FRUs and the Shelf Management Controller (ShMC) to relay messages in multi-master mode, that is, listening as I^2C slaves for messages addressed to them and taking control of one of the available buses as I^2C masters to send a message, managing failures and collisions in case they take place.

Other management signals: A set of pins in the backplane electrically encodes the address of the physical slot a board is inserted in.

A. Hardware Platform Management

The main task of the HPM system is monitoring the health of the hardware by collecting sensor data (voltages, current draws, temperatures, fan speeds, etc.) and taking corrective actions (increasing the fan speed, switching off power, trigger alarms, etc.) in case the measurements lie outside the nominal range. The system is also tasked with orchestrating the power consumption of the FRUs in a shelf, such that the overall parameters do not depart from the allowed operational envelope.
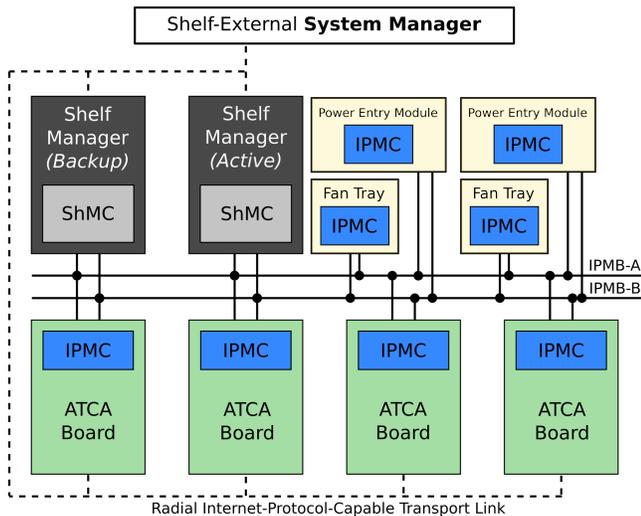

Fig. 1. ATCA HPM Architecture. Adapted from [1].

As shown in Fig. 1, the HPM system is composed of IPMCs that manage one or more FRUs, one or two Shelf Management Controllers (ShMC) for each shelf, and an optional external System Manager. The System Manager is a global high-level controller that manages ShMCs in multiple shelves connected by a network; the ShMC is a device that orchestrates the behavior of all the IPMCs running inside the FRUs hosted on its shelf; and the IPMC is a controller local to each FRU that is responsible for controlling all aspects specific to the FRU operational state and providing real-time hardware status and sensor information to the ShMC. The communication between FRUs and ShMCs takes the form of messages based on the Intelligent Platform Management Interface (IPMI) [23] [22] protocol. In the specific application to ATCA, the IPMI protocol is extended through the addition of remote board control, fault detection and fault management functions.

B. IPMC: Hot-swap and other functions

One of the main roles of the IPMC is the management of the hot-swap operation, where the board is activated or deactivated in a graceful way, assisted by the ShMC. The procedure starts with the insertion of a board into a shelf, the IPMC is powered immediately, informing the ShMC about its presence. The board activation request from the user is signalled by locking the mechanical handle on the front plate of the board, which triggers a switch. The IPMC then sends to the ShMC information about its sensors (like name, units, conversion constants, thresholds and many others), board identification and power requirements. The ShMC then evaluates the power budget in the shelf and may or may not authorize to power ON the board. If power-ON permission is granted, the IPMC follows the specific steps needed to bring the electronics on the board to the active state, for example booting an operating system on a processor. In a similar fashion, when the front handle is unlocked the IPMC begins the board-specific procedure to shut down gracefully the electronics in the board, coordinating with the ShMC in doing so.

Considering the hot-swap as an example, it is clear that the main reason for the IPMC to exist in the standard, is that it exposes a standard abstract interface for the management of the board, hiding the board-specific details from the ShMC. Thanks to this standard interface, the ShMC can be designed generically and without the need for prior knowledge of the details of boards installed into the shelf. This also means that the IPMC needs to be specifically customized for the board it is designed to run in, either by configuring its generic firmware through scripts, or by designing a firmware tailored for this purpose. Other functions of the IPMC include declaring the list of sensors available on the board to the ShMC, reading them out and transmitting their readings to the ShMC, which is the device tasked with the ultimate decision on whether to command the shut down a FRU in the shelf.

ATCA boards can host expansion boards (see Fig. 2), which can be Advanced Mezzanine Cards (AMCs) [24] - meant to be inserted into front-facing slots of the main board - and Rear Transition Modules (RTMs) [1], [25], [26] - hosted in an optional slot on the rear of the backplane. When expansion boards are used, the IPMC operates as the managing controller for their operation, in a similar way as the ShMC manages the FRUs in a shelf. The communication between IPMC and the expansion boards takes place via a local IPMI bus (IPMB-L) - with electrical characteristics similar to a single IPMB-0 channel - and a number of status and control signals.
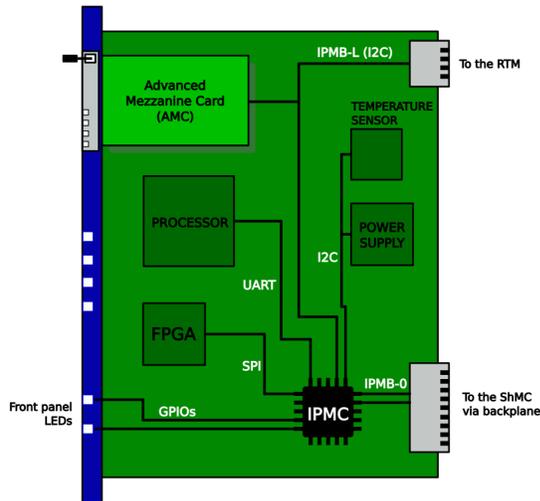
Fig. 2. Example scheme of an ATCA electronic card hosting expansion boards.

III. OPENIPMC

OpenIPMC is a piece of portable software implementing the behavior of an IPMC. It stems from a collaboration between São Paulo Research and Analysis Center (SPRACE) and the Karlsruhe Institute of Technology (KIT) on development of electronics for the Phase-2 upgrade of the CMS experiment. The development of OpenIPMC is the evolution of a previous project led by SPRACE researchers in collaboration with Fermilab, in which the collaborators successfully developed the IPMC [20] [21] for the Pulsar2b ATCA board [27]. The free and open-source nature of OpenIPMC helps in the customization and debugging of the firmware running on prototype ATCA boards during their development and commissioning, and allows adapting it to the operational needs as they evolve over the expected long period of operation.

The software is written in C language, based upon the FreeRTOS free and open-source real-time operating system [28] and targeting embedded Microcontroller Units (MCUs) and systems-on-a-chip (SoCs). Thanks to the very wide support of FreeRTOS across different hardware manufacturers, OpenIPMC can easily be ported to any MCU/SoC supported by FreeRTOS, provided that the device is equipped with sufficient amount of resources to run the code and with enough I/O peripherals to interface the microcontroller to the ATCA backplane and local board functions. In our tests we estimated the size taken by OpenIPMC by observing the size increment experienced by a basic firmware when OpenIPMC was included (table I). From the table we can also observe that, while OpenIPMC requires just a few tens of kiB, there are large variations in the overall firmware size across different microcontrollers and SoCs, likely due to differences in the board support package implementations. We have not yet optimized the stack size for the OpenIPMC real-time tasks, which will be described in the next paragraphs. The sizes presented in the table include instructions, core data and the RAM reserved for the heap working space.

In the case of MCUs which do not ship with enough hard-

TABLE I
ESTIMATED SIZE OF OPENIPMC IN FIRMWARE AND TOTAL SIZE OF FIRMWARE FOR DIFFERENT ARCHITECTURES.

Device	OpenIPMC size (kiB)	Total size (kiB)
ZynqUS+	55	218
ESP32	33	93
STM32	40	75

ware peripherals to cover all the needed I^2C channels, communication can be established by pairs of software-driven GPIOs to emulate such peripherals. Still, this operation can be rather CPU-intensive and therefore should be avoided on very busy channels when possible. When choosing a microcontroller or SoC for IPMC applications, we recommend the use of devices with at least three I^2C dedicated peripherals (either hard in-silica cores or soft cores in the programmable logic of an FPGA), with two to be to used in IPMB-0 communication and at least one to control local devices such as temperature and current sensors. In cases where the board can host AMCs or RTMs, an additional hardware peripheral should be dedicated to the operation of the IPMB-L interface. OpenIPMC achieves portability thanks to a stringent separation between its core behavioral code and the accessory interface to the underlying microcontroller drivers and hardware. This is accomplished through a *Hardware Abstraction Layer* (HAL) and *board-specific control* functions (as shown in Fig. 3).
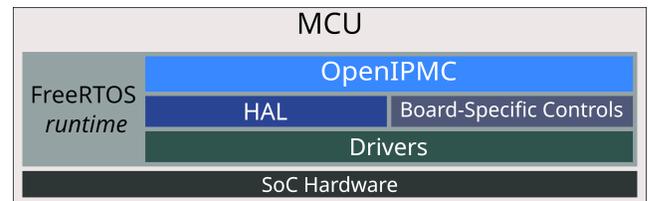

Fig. 3. Schematic of the relationships between the core of OpenIPMC, its HAL and the hardware-specific drivers in the context of the FreeRTOS runtime, and the hardware peripherals of the SoC/microcontroller.

A. Running OpenIPMC in FreeRTOS

OpenIPMC was designed to be integrated in a wider micro-controller firmware according to the needs of the developer and, therefore, it was a natural choice to adopt a multi-task real time operating system such as FreeRTOS as underlying infrastructure. The core of OpenIPMC runs as a collection of FreeRTOS tasks (described in the following paragraphs) running in parallel and interacting via thread-safe queues and semaphores, with no requirement for exclusive access to the processor. Hence, the developer has the freedom to extend the functions of the firmware by adding independent tasks excluded from the OpenIPMC execution flow. The FreeRTOS scheduler [28] allows to set the priority of each task, such that the critical ones can be guaranteed to execute with low latency, unimpeded by low-priority tasks. OpenIPMC executes in parallel the tasks listed below.

ipmb0_msg_receiver_task Listens for incoming messages on IPMB-A and IPMB-B. It performs basic veri-
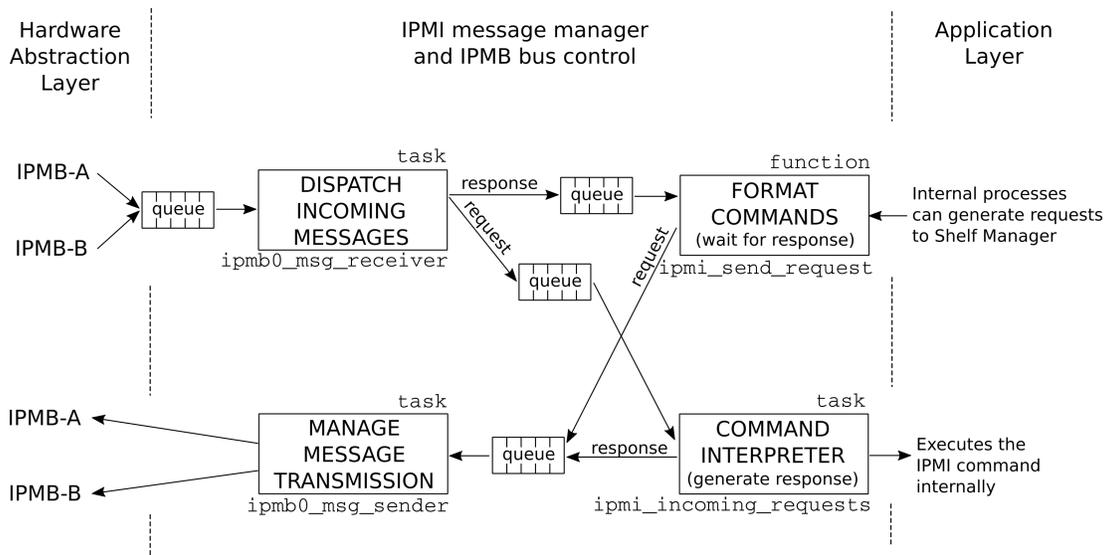
Fig. 4. IPMI message transactions in OpenIPMC. "Hardware Abstraction Layer" represents the collection of adapter functions between OpenIPMC and the I²C driver available for the desired platform. "Application" Layer represents the set of functions responsible to execute the IPMI commands coming from ShMC. Processes in this layer also can generate requests to the ShMC by calling the proper API.

fication on the message, checking its checksum and the type of the message (request or response). The task forwards the message to the proper queue according to its type.

ipmb0_msg_sender_task Processes messages outbound to the IPMB-A and IPMB-B channels. The task pops messages from a specific output queue, chooses the IPMB channel to use, sends the message and manages retries in case of transmission failures.

ipmi_incoming_requests_task Is responsible for generating responses for received requests, triggering the proper processes to do so. This task interacts with both previously described `ipmb_0_msg` tasks.

fru_state_machine_task Drives the FRU state transitions, such as responding to the *hot-swap* events and triggering the board-specific activation/deactivation routines.

ipmc_handle_switch_task Periodically samples the state of the front-plate handle and triggers *hot-swap* events. This task interacts with `fru_state_machine_task`.

ipmc_blue_led_blink_task Controls the blinking time of the blue-LED, a device mandated by the PICMG standard in the front panel to indicate the power status of the board. This task polls the current state in `fru_state_machine_task`.

The interplay between tasks responsible for the IPMI message transactions described above and the data flow between them are schematized in Fig. 4.

We chose to use FreeRTOS as a base for our software because it is a mature and widely-used real-time operating system designed to be robust, with a tiny footprint, and a wide range of supported devices [28]. Among the safety features offered by this Real-Time Operating System (RTOS), we employ the stack overflow detection feature to evaluate the stability of our application, and we have the possibility to trigger a global reset of the MCU upon such an event. Furthermore, OpenIPMC has been designed to avoid the use

of dynamic memory allocation after its initialization. As a good practice, all tasks and other FreeRTOS objects (queues, semaphores, etc) are allocated just once, at startup, and heap over-run events are set to be logged if they occur. The few globally-accessible symbols are protected through the use of mutexes against racing conditions.

B. OpenIPMC Hardware Abstraction Layer

```
1  // The pointer to this function will be
2  // registered into the OpenIPMC HAL as the
3  // implementation of the blue-LED state change.
4  void ipmc_ios_blue_led_set(int blue_led_state)
5  {
6    if( blue_led_state == 1 )
7      gpio_driver( BLUE_LED_PIN_NUMBER, SET_TO_HIGH );
8    else
9      gpio_driver( BLUE_LED_PIN_NUMBER, SET_TO_LOW );
10 }
```

Listing 1. A function implementing the blue-LED state change on a specific hardware. This `ipmc_ios_blue_led` prototype is available as an example in the OpenIPMC code.

The HAL provides OpenIPMC with an interface to the hardware (IMPB-A, IMPB-B, hardware address pins, status LEDs, handle switch, etc.) that is prescribed in the PICMG standard to be present in every ATCA board and is fundamental for OpenIPMC operation. This HAL is composed of a set of functions that call the drivers used to access the relevant peripherals in the microcontroller. These functions take the form of declared - but undefined - functions in the base software release of OpenIPMC, and must be implemented by the developer of a new board to fit the specific hardware interface. As an example, turning ON the blue-LED, could be accomplished through a GPIO being flipped to the HIGH or LOW status, or be controlled by an on-board device like an I/O Expander attached to a local I²C bus. The idea behind the OpenIPMC HAL is to give the board developer the freedom to choose how to turn ON the blue-LED in hardware and

implement the corresponding function in software to do so. In Listing 1 we show as an example the code needed to operate a GPIO-driven blue status LED in the front panel.

C. Board-Specific Controls

The interaction between OpenIPMC and the electronics in the payload (that is, the electronics responsible for the main functions of the ATCA board) takes place through an API, which accepts the registering of callbacks to functions managing the various operations to be performed on the payload. The choice of using an implementation based on function callbacks is justified by the fact that boards are designed with great variety in terms of functions and components, additionally the PICMG standard makes no prescription on this aspect of the board design. Through the API the user can implement payload control routines that fit the hardware design of his choice. We chose to call this layer, made of callable hardware interface functions, the "board-specific controls" (see Fig. 3).

Among the classes of operations that need to be implemented in the IPMC according to the PICMG standard are the ones relative to Power Management and Sensor Readings. Power Management refers to the activation, deactivation, reset and the regulation of the power draw of the different circuits present on the payload. In OpenIPMC this management is performed by the state machine implemented in `fru_state_machine_task`, which calls a number of user-defined board-specific controls. Different states (M0, M1, ...) of the state machine represent the different power states of the payload during activation and deactivation operations. Since the ShMC centralizes the management of power allocation for all the ATCA boards in the shelf, the activation/deactivation process involves negotiations between OpenIPMC and the ShMC through the IPMB bus. The other operations on the payload specified by the PICMG standard, like *Cold Reset* and *Warm Reset*, can be also implemented through board-specific controls.

The IPMC must be able to collect sensor readings from the payload and send them to the ShMC, formatted in accordance with the IPMI specifications. Similarly to the case of Power Management described above, it is the responsibility of the user to provide a function callback such that OpenIPMC can read a sensor value using the correct procedure and protocol (for example, by accessing an SPI or I^{2}C register, writing and reading a GPIO, using a lookup table to interpret data, etc.). A sensor reading is generally triggered by a request sent by the ShMC. OpenIPMC receives this request on the IPMB bus, interprets it and executes the callback associated with the reading of that specific sensor, and finally sends the value and sensor status to the ShMC.

According to the standard, the IPMC is also responsible for sending to the ShMC information about each sensor such as sensor type, measurement units, linearization parameters, threshold values, accuracy, a string containing its name, and other information, which is collected into a data structure called the *Sensor Data Record* (SDR) [23]. This is generated by the IPMC at startup and transmitted to ShMC during the activation process. OpenIPMC provides an API to create

SDRs for the board sensors and automatically manages their transmission to the ShMC.

The implementation of the Board-Specific controls and of the HAL layers strongly depend on the specific hardware being targeted and on its driver interface, as designed by the hardware manufacturer. Due to the wide variety of microcontrollers supporting FreeRTOS from different manufacturers, the implementation and debugging of these layers should be tailored to the target hardware on a device-by-device basis. We believe that the design of this layer should be left in control of the user to match his specific needs.

D. Licensing and code distribution

OpenIPMC is released under Mozilla Public License 2.0 [29], a license which allows of the open-source code to be statically linked, as it is common when building microcontroller firmware binaries. The code is currently publicly available on Gitlab.com [30].

IV. DEVELOPMENT AND TESTING PLATFORMS

We began the development of OpenIPMC on the AVNET Ultra96 board [31], establishing early the separation between behavioral code, HAL and the board-specific controls described before. This separation was of great help to port the code to a number of different architectures. With the exception of the Trenz-Serenity one, all tests were performed at São Paulo State University using an ATCA horizontal shelf manufactured by Comtel (model CO6B-6U-FM40X) with 6 slots and two redundant ShMM-700R Shelf Management Controllers by PigeonPoint (running firmware version 3.6.1.3).

Fig. 5. Development setup with the Ultra96 sitting on top of the Pulsar-2b.

A. Ultra96-Pulsar2b Development Platform

In this setup the Ultra96 acts as an IPMC for a Pulsar-2b board. The ZU3EG SoC on the Ultra96 board belongs to the ZynqUS+ EG family [32], [33], containing four ARM A53 high-performance application processors (APU) and two ARM R5 real-time processors (RPU). APU and RPU are independent processing units within the same SoC package which can run

different operating systems. The FreeRTOS instance hosting the OpenIPMC tasks runs on the RPU, leaving the APU free to run a Linux-based operating system. The communication between the Ultra96 and the ATCA backplane takes place through an adapter board, which fits into the Mini-DIMM slot normally used by the Pulsar-2b to host its IPMC board. This adapter board exposes on a 2mm header the IPMB-A and IPMB-B buses, the hardware address lines, the blue-LED control and the handle switch state line, such that they can be connected to the pins of the Ultra96 through a flat cable (Fig. 5).

In tests performed on this setup, OpenIPMC correctly executes its management tasks, such as the activation and deactivation triggered by the handle switch, and the declaration and read-out of a dummy sensor to the ShMC. In this setup, both power and sensor management are simulated, since the current version of the DIMM adapter does not provide access to the sensors I^2C bus and the power supply controls of the Pulsar-2b. Furthermore, using this platform, OpenIPMC was used to trigger the boot sequence of a CentOS Linux distribution [42] on the APU of the ZynqUS+.

B. Trenz-Serenity Development Platform

The Trenz-Serenity Platform is the platform currently being used at KIT for the development of a centralized management architecture based on ZynqUS+ [34]. Its main components are the Serenity ATCA Carrier Card [13] and a Trenz TE0803 module [35] hosted on a custom adapter card (*Trenz Adapter*), which allows the TE0803 to fit into the slot - originally designed to accommodate a COMExpress industrial computer [36] - present on the Serenity board (Fig. 6). Additionally, the Trenz Adapter interconnects the TE0803 module to the IPMC slot on the Serenity board through a ribbon cable and an adapter Mini-DIMM board, allowing the ZynqUS+ on the TE0803 module to perform the role of an IPMC.

Fig. 6.   Trenz-Serenity setup in a vertical shelf at KIT.

Similarly to the case of the Ultra96-Pulsar2b platform, the Trenz TE0803 module hosts a ZynqUS+ EG device. OpenIPMC runs on the ARM Cortex-R5 cores and the IPMB channels are implemented using both I^2C hard peripherals available on the ZynqUS+ *Processing System* (PS). However, this test setup presents a number of significant differences compared to the Ultra96 case and the OpenIPMC HAL subsystem has been modified accordingly. For example, due to the different signal routing on the Trenz module some critical signals (Hardware Address, blue-LED, Handle_Switch and 12V_Enable) have been routed to PCA9557 IO expanders controlled by an I^2C master in the ZynqUS+. Since both I^2C channels available on the ZynqUS+ PS are already allocated to the IPMB buses, to drive the expander we use a software-emulated I^2C master, where two GPIO signals are controlled by a driver to behave as the SDA and SCL lines of an I^2C peripheral.

On this setup, OpenIPMC has shown to correctly perform its management tasks, including turning ON the main board power supply, managing sensors and triggering the boot of a CentOS Linux distribution running on the APU of the ZynqUS+.

C. ESP32-Pulsar2b Development platform

To demonstrate the hardware independence of the core behavioral code of OpenIPMC we performed an exercise, porting OpenIPMC to an architecture significantly different to the ZynqUS+ SoC used in the two development setups described earlier, and measuring the human effort needed to complete the porting. We chose as platform the Espressif ESP32 microcontroller, which is based on a Harvard-architecture Tensilica Xtensa LX6 processor, significantly different from the ARM R5 cores on the ZynqUS+. The ESP32 microcontroller is an affordable yet powerful device designed for wireless IoT applications [37] [38].
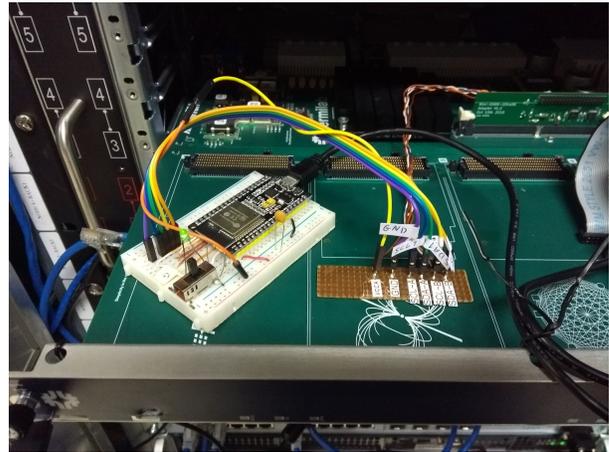

Fig. 7.   ESP32-Pulsar2b development platform at SPRACE in São Paulo.

We assembled the platform for this demonstration starting from the Ultra96-Pulsar-2b setup and swapping the Ultra96 with a breadboard on which we installed an ESP32 development board, as shown in Fig. 7. In this setup, the only signals connected are the IPMB buses (allowing communication to the ShMC), the handle switch (which was emulated by a simple bi-stable switch), and the blue-LED (a simple through-hole LED), the last two being installed on a breadboard. For simplicity the Hardware Address, being a set of static signals

bound to simple GPIO pins, was not read out and its expected values were trivially hard-coded into the HAL.

Porting OpenIPMC to this platform required just 3 person-weeks, which we identify as a success. Most of the time was spent in circumventing an inflexible implementation of the I^{2}C multi-master mode in the ESP32 IDF board support package, while porting of the core OpenIPMC code required very little effort. The difficulty arises from the I^{2}C driver when the device is in slave mode listening for messages: in the ESP32 the driver expects the developer to know in advance the size of each future incoming message, which is not the case for IPMB communication, where messages from the master can have variable length. We tested the behavior of the IPMC and its interaction with the ShMC and found it to work as expected.

D. STM32-Pulsar2b Development platform

Aiming for deployment into a mass-produced, reliable and affordable microcontroller, we chose to port OpenIPMC to the STM32 family [39], which is one of the most widely used families of microcontrollers worldwide. They are available in a large variety of types optimized for a large number of application classes. Specifically, the STM32H7 family is composed of high-performance microcontrollers [41], characterized by powerful processors and a large number of IO peripherals, which makes them interesting for use in a feature-rich IPMC. On this test platform, we replaced the ESP32 board in the ESP32-Pulsar2b Development platform with the NUCLEO-H745ZI-Q development board manufactured by ST Microelectronics, which hosts a STM32H745ZIT6U microcontroller [40]. This device is characterized by a dual Cortex-M7/Cortex-M4 core, four I^{2}C, four USART and four UART peripherals, and a large number of GPIO channels.
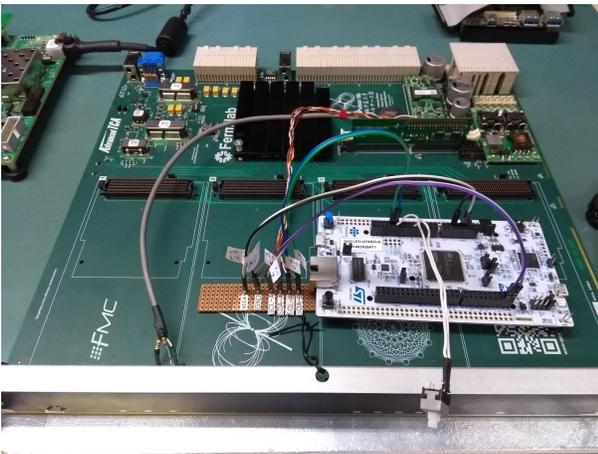

Fig. 8.   STM32-Pulsar2b development platform at SPRACE in São Paulo.

We ported OpenIPMC on the STM32 successfully and with relatively little effort, taking around 5 person-weeks. Most of the effort was focused in circumventing an inflexible implementation of I^{2}C multi-mastering, similar to the case of ESP32. We could verify that OpenIPMC behaves as expected in its communication with the ShMC and in the activation process. In Fig. 9 an oscilloscope trace of the IPMB-A and IPMB-B SDA buses shows the exchange of messages between

the IPMC and the ShMC. The transaction takes place in around 300 μs, well below the IPMI latency requirement.
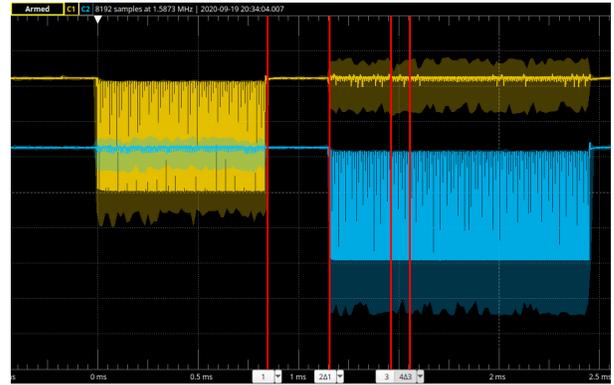

Fig. 9.   Example of IPMB communication: OpenIPMC is running on the NUCLEO-H745ZI-Q board, receiving a message from the ShMC on IPMB-A (activity on the left) and replying on IPMB-B (activity on the right) in around 300 μs. The apparent cross-talk is an artifact of the measurement setup.

E. Comparison between different test platforms

In table II we summarize the differences in the use of different peripherals to read and operate the various I/O needed by OpenIPMC. Note that *Soft-Reset* and *12V_Enable* are board-specific controls, which were only used in the Trenz-Serenity setup.

V. SUMMARY AND OUTLOOK

In this document we presented OpenIPMC, a software written in C for embedded microcontrollers implementing an IPMC as defined by the PICMG ATCA standard. The operation of the software has been demonstrated successfully on different hardware architectures such as ZynqUS+, ESP32 and STM32. We plan to continue our development on new hardware designs, where OpenIPMC will be given full control and monitoring duties over its hosting ATCA carrier board. We also plan to introduce support for local add-on boards, such as AMCs and RTMs. While OpenIPMC has been conceived in a context of academic research, its full customizability make it attractive for many other applications, such as innovative industrial designs and board prototyping.

ACKNOWLEDGMENT

TABLE II
COMPARISON OF PERIPHERAL USAGE BETWEEN THE DIFFERENT TEST SETUPS. PS = PROCESSING SYSTEM AND PL = PROGRAMMABLE LOGIC IN ZYNQUS+.

	Ultra96-Pulsar2b	Trenz-Serenity	ESP32-Pulsar2b	STM32-Pulsar2b
SoC/microcontroller	ZynqUS+ ZU3EG	ZynqUS+ ZU4EG	ESP32	STM32H745
Architecture	ARM Cortex-R5	ARM Cortex-R5	Tensilica Xtensa Lx6	ARM Cortex-M7
IPMB-A	I^2C-PS	I^2C-PS	I^2C	I^2C
IPMB-B	I^2C-PS	I^2C-PS	I^2C	I^2C
Hardware Address (bit 0)	GPIO-PL	PCA9557 using software-emulated I^2C on GPIO-PS	GPIO	GPIO
Hardware Address (bits 1-7)	GPIO-PS	PCA9557 using software-emulated I^2C on GPIO-PS	GPIO	GPIO
Blue_LED	GPIO-PL	PCA9557 using software-emulated I^2C on GPIO-PS	GPIO	GPIO
Handle_Switch	GPIO-PL	PCA9557 using software-emulated I^2C on GPIO-PS	GPIO	GPIO
Soft_Reset	-	GPIO-PS	-	-
12V_Enable	-	PCA9557 using software-emulated I^2C on GPIO-PS	-	-

REFERENCES

[1] PICMG, "PICMG 3.0 - AdvancedTCA Base Specification", Revision 3.0, March 24, 2008.
[2] PICMG, "Open Modular Computer Standards", Available: https://www.picmg.org/. Accessed May 26, 2021.
[3] PICMG, "Hardware Platform Management", Available: https://www.picmg.org/product-category/hardware-platform-management/. Accessed May 28, 2021.
[4] A. B. Mann et al., "A flexible advancedTCA based sampling ADC system for multimodality positron emission tomography," 2007 IEEE Nuclear Science Symposium Conference Record, 2007, pp. 1729-1732, doi: 10.1109/NSSMIC.2007.4436494.
[5] PICMG, "AdvancedTCA® Overview", Available: https://www.picmg.org/openstandards/advancedtca/. Accessed May 26, 2021.
[6] CMS Collaboration, "The Phase-2 Upgrade of the CMS Tracker", Technical Design Report, CERN-LHCC-2017-009, Jul. 1, 2017.
[7] CMS Collaboration, "The Phase-2 Upgrade of the CMS DAQ Interim Technical Design Report", CERN-LHCC-2017-014, Sep. 12, 2017.
[8] ATLAS Collaboration, "Technical Design Report for the ATLAS Inner Tracker Pixel Detector", CERN-LHCC-2017-021, Sep. 23, 2017.
[9] ATLAS Collaboration, "Technical Design Report for the ATLAS Inner Tracker Strip Detector", CERN-LHCC-2017-005, Apr. 01, 2017.
[10] ATLAS Collaboration, "Technical Design Report for the Phase-II Upgrade of the ATLAS TDAQ System", CERN-LHCC-2017-020, Sep. 23, 2017.
[11] LHCb Collaboration, "LHCb Tracker Upgrade Technical Design Report", CERN-LHCC-2014-001, Feb. 21, 2014.
[12] LHCb Collaboration, "LHCb Trigger and Online Upgrade Technical Design Report", CERN-LHCC-2014-016, May. 14, 2014.
[13] A. Rose et al. "Serenity: An ATCA prototyping platform for CMS Phase-2," PoS TWEPP2018 (2019), 115 doi:10.22323/1.343.0115
[14] A. Albert, J. Butler, Z. Demiragli, K. Finelli, D. Gastler, E. Hazen, J. Rohlf, S. Yuan, T. Costa De Paiva and V. Martinez Outschoorn, et al. PoS TWEPP2019 (2020), 120 doi:10.22323/1.370.0120 [arXiv:1911.06452 [physics.ins-det]].
[15] L. Calligaris on behalf of the CMS collaboration, J. Phys. Conf. Ser. 1690 (2020) no.1, 012039 doi:10.1088/1742-6596/1690/1/012039
[16] J. Mendez et al. "CERN-IPMC solution for AdvancedTCA blades", PoS, vol. TWEPP2017, Santa Cruz, CA, USA, September 11-15, 2017, vol.TWEPP-17, 2018, p. 053. DOI 10.22323/1.313.0053
[17] Nvent Schroff, "Pigeon Point Shelf Management Mezzanine Solutions" https://schroff.nvent.com/solutions/schroff/applications/picmg-products. Accessed May 26, 2021.
[18] S. Lafrasse. "LAPP IPMC Overview", IPMC Workshop - ATLAS upgrade, 2018 https://indico.cern.ch/event/737733/contributions/3077000/attachments/1730606/2796822/LAPP_IPMC_Overview.pdf. Accessed May 26, 2021.
[19] P. Perek et al., "ATCA carrier board with dedicated IPMI controller", Proceedings of the 17th International Conference Mixed Design of Integrated Circuits and Systems - MIXDES 2010, 2010, pp. 139-143.

[20] L. A. Ramalho et al., "Development of an Intelligent Platform Management Controller for the PulsarIIb", in IEEE Nuclear Science Symposium and Medical Imaging Conference, San Diego, CA, USA, 2015.
[21] T. C. Paiva, "Remote development environment with reconfigurable components in the advanced telecom computing architecture context", M.S. thesis, FEIS UNESP, Ilha Solteira, SP, Brazil, 2016.
[22] Intel, Hewlett-Packard, NEC, Dell, "Intelligent Platform Management Bus Communications Protocol Specification", v1.0 R1.0, 2002.
[23] Intel, Hewlett-Packard, NEC, Dell, "Intelligent Platform Management Specification", Version 1.5, Revision 1.1, 2002.
[24] PICMG, AMC.0 R2.0, "Advanced Mezzanine Card Base Specification," 2006
[25] PICMG, IRTM.0 Revision 1.1, "AdvancedTCA Intelligent Rear Transition Module (IRTM) Base Specification ," 2011
[26] PICMG, 3.8 R1.0, "AdvancedTCA Rear Transition Module Zone 3A," 2011
[27] S. Ahuja et al., "A Full Mesh ATCA-based General Purpose Data Processing Board (Pulsar II)," doi:10.2172/1431570
[28] FreeRTOS Reference Manual. Accessed May 26, 2021.
[29] "Mozilla Public License Version 2.0", https://www.mozilla.org/en-US/MPL/2.0/. Accessed May 26, 2021.
[30] "OpenIPMC repository", https://gitlab.com/openipmc/openipmc
[31] Avnet, "Ultra96", Available: http://zedboard.org/product/ultra96. Accessed May 28, 2021.
[32] Xilinx DS891, Zynq Ultrascale+ Datasheet v1.8, 2019.
[33] Xilinx UG1085, Zynq Ultrascale+ Tech. Ref. Manual v2.2, 2020.
[34] L. Ardila-Perez et al., A novel centralized slow control and board management solution for ATCA blades based on the Zynq UltraScale+ System-on-Chip, 24th International Conference on Computing in High Energy & Nuclear Physics CHEP2019, EPJ Web Conf.245 (2020), p. 01015. DOI 10.1051/epjconf/202024501015.
[35] Trenz Electronic GmbH, "TE0803 - Zynq UltraScale+", Available: https://shop.trenz-electronic.de/en/Products/Trenz-Electronic/TE08XX-Zynq-UltraScale/TE0803-Zynq-UltraScale/. Accessed May 28, 2021.
[36] PICMG, "PICMG COM.0 - COM Express Module Base Specification", Revision 3.0, Mar. 31, 2017.
[37] Espressif Systems, ESP32 Datasheet v3.6, 2020.
[38] Espressif Systems, ESP32 Tech. Ref. Manual v4.4 , 2020.
[39] STMicroelectronics, "STM32 32-bit Arm Cortex MCUs", Available: https://www.st.com/content/st_com/en/products/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html. Accessed May 26, 2021.
[40] STMicroelectronics DS12923, STM32H745xI/G Product Datasheet Rev 1, 2019.
[41] STMicroelectronics RM0399, STM32H745/755 and STM32H7474/757 Reference Manual v3.0, 2020.
[42] CERN SoC working group, "CentOS for ZynqMP" https://twiki.cern.ch/twiki/bin/view/SystemOnChip/CentOSForZynqMP. Accessed May 26, 2021.