Abstract:

This document describes the CERN Module Management Controller (MMC) features, its use and integration on specific AMC designs. It specifies both Hardware design rules to use the MMC mezzanine and Software customization to fit to AMC requirements.
Table of contents

1 Introduction .................................................................................................................................... 3
  1.1 MMC role and environment ................................................................................................. 3
2 Setup ............................................................................................................................................... 6
  2.1 SVN repository architecture ................................................................................................. 6
  2.2 Installing the Integrated Development Environment (IDE) ............................................... 7
  2.3 Downloading the HPM software into the microcontroller .................................................. 8
  2.4 Downloading/upgrading the MMC application (JTAG) ....................................................... 10
  2.5 Downloading/upgrading the MMC application (HPM.1) .................................................... 10
3 Hardware ...................................................................................................................................... 12
  3.1 Mezzanine pinout .............................................................................................................. 12
4 HPM.1 ........................................................................................................................................... 13
5 MMC Application software ......................................................................................................... 14
  5.1 Commands implemented .................................................................................................... 14
  5.2 User code architecture ...................................................................................................... 15
  5.3 User GPIO .......................................................................................................................... 15
  5.4 Analog to Digital Converter (ADC) ..................................................................................... 16
  5.5 FRU information ................................................................................................................ 16
  5.6 SDR repository ................................................................................................................... 18
  5.7 Power management ............................................................................................................ 23
  5.8 E-keying ............................................................................................................................. 24
  5.9 Sensors ............................................................................................................................... 25
  5.10 Custom LEDs .................................................................................................................. 26
  5.11 OEM commands ............................................................................................................... 28
  5.12 Controller specific commands ......................................................................................... 29
  5.13 Bench top .......................................................................................................................... 30
6 Appendix A: Cygwin installation on Windows ........................................................................ 31
7 Appendix B: MMC microcontroller signals .......................................................................... 33
8 References ................................................................................................................................. 34
9 Table of figures ......................................................................................................................... 35
10 Table of tables ......................................................................................................................... 35
1 Introduction

The CERN MMC (Module Management Controller) is compliant with the Intelligent Platform Management Interface (IPMI) v1.5 standard [1] as well as the PICMG extension for xTCA. The module is implemented as a small mezzanine card to be mounted on AMC modules (Advanced Mezzanine Card). The MMC mezzanine design can also be used as a reference design for direct integration into an AMC project. Development of the MMC software was started by DESY. The original MMC source code was given to CPPM and CERN by Kay Rehlich and Vahan Petrosyan from DESY. The MMC project has then evolved in a joint collaboration between CERN and CPPM and is currently supported by the PH-ESE-BE group at CERN (https://ph-dep-ese.web.cern.ch/ph-dep-ese/).

1.1 MMC role and environment

The MMC communicates with the carrier manager through the IPMB-L bus (I2C based) via the IPMI standard. The module when implemented on an AMC board can be used in both μTCA and/or ATCA environments. Figure 1 below shows the Hardware Platform Management components and their connectivity as well as the MMC role and location in an ATCA architecture.

![Figure 1: ATCA Hardware Platform Management](image1)

In μTCA, the Hardware Platform Management, shown below (Figure 2), is mostly the same. In this architecture, the carrier and shelf managers are both implemented in the MCH (MicroTCA Carrier Hub).
Upon AMC insertion the MMC provides information like power consumption, ports connectivity, clock configuration, etc. to the carrier IPMC. Based on this information, the carrier IPMC, in association with the Shelf manager, decides whether the AMC can be activated or not. The AMC.0 standard [2] defines an activation/de-activation state machine that handles the AMC hot swapping features. The Figure 3 describes the different states and actions performed during the change of states (AMC activation/de-activation).
Figure 3: Activation / De-activation state machine

The state machine is managed by the Carrier IPMC. To perform this management task, the carrier manager controller receives handle switch events from the MMC.

The software for the CERN MMC supported features are listed here:

- FRU information (board information, product information, connectivity, ...)
- FRU management (hot swap, power, reset, ...)
- E-keying
- Clock configuration
- Sensors monitoring (temperatures, voltages, ...)
- Hooks for the addition of user functions (OEM support)
- Remote upgrade with HPM.1
2 Setup

This section describes how to install the required development software and tools.

2.1 SVN repository architecture

The MMC project is stored on the following CERN SVN repository server
(https://svnweb.cern.ch/cern/wsvn/ph-ese/be/mmc/?#a1c8473bc0300afaf74b5ab8f1b9502da).

Note: this directory is restricted to members of the phse-mmc-users e-group (https://e-groups.cern.ch/e-groups). Please contact Markus Joos (markus.joos@cern.ch) to request membership.

Note: On Linux systems, the code can be checked out with the command “svn co svn+ssh://svn.cern.ch/reps/ph-ese/be/mmc/”

These is the trunk architecture of the SVN project is given on (Figure 4) below:

![SVN repository architecture diagram](image)

Figure 4: SVN repository architecture

The **Docs directory** contains this document as well as a short README file.

The **HPM directory** contains the HPM project described in chapter 4 of this document.

The **MMC directory** contains the MMC application source code composed of the application core and the user directory. The customization of this program is detailed in chapter 5.
The **Tools directory** contains 3 tools in relation with the CERN MMC:

- **HPMDownloader**: Allows programming the MMC application using the HPM.1 standard [3]. The procedure is described in section 2.5.
- **FRU Editor**: Generates an FRU information binary from an .h file that can also be used in the MMC application project. One option allows downloading the generated binary into the MMC EEPROM.
- **MTCALib**: C library that allows sending IPMI commands via Ethernet (RMCP) to μTCA modules (MCH, AMC, Cooling units, Power modules).

### 2.2 Installing the Integrated Development Environment (IDE)

The CERN MMC mezzanine is based on the ATMEGA128 microcontroller from ATMEL. For the compilation of the application, it is recommended to use the Atmel Studio IDE (Integrated Development Environment). This environment provides advanced programming and debugging features for MCUs, including the ability to capture data trace information. The latest version of Atmel Studio can be downloaded from the Atmel website (current version: [http://www.atmel.com/microsite/atmel_studio6/](http://www.atmel.com/microsite/atmel_studio6/)).

![USB driver installation](image1.png)

**Figure 5: USB driver installation**

The USB driver installation must be accepted to use the programmer as described in the next section. After the driver installation, the setup program install the Atmel Studio environment.

![Atmel Studio installation](image2.png)

**Figure 6: Atmel Studio installation**

After successful IDE installation, the CERN_HPM and/or CERN_MMC projects can be opened.
2.3 Downloading the HPM software into the microcontroller

The first programming step consists of downloading the HPM program into the MMC microcontroller flash memory. The CERN_HPM project should be opened (File > Open > Project/Solution and select cern_hpm.atsln) and built before being downloaded into the MMC. The figure below shows the Atmel Studio interface after opening the project:

**Note:** Before compilation, the IANA and product ID must be changed to match with the AMC card it will be installed on. This information is set in the config.h file: the IANA_MANUFACTURER_ID field, which corresponds to the IANA manufacturer identification number ([http://www.iana.org/assignments/enterprise-numbers](http://www.iana.org/assignments/enterprise-numbers)) or, and the PRODUCT_ID field, which corresponds to the product identification number. Once these parameters are correctly registered, the project can be built (Build > Build solution)

Before downloading the binary, the microcontroller fuses have to be programmed with the device programming tool (Tools > Device programming) as shown in Figure 8 (Tool selected and Fuses tab):
Finally the HPM binary can be downloaded into the microcontroller flash memory (click the “program” button in the Flash box):

Figure 8: MMC microcontroller fuse configuration

Figure 9: MMC bootloader programming using JTAG
2.4 Downloading/upgrading the MMC application (JTAG)

The MMC application must first be built using Atmel Studio. The cern_mmc project must be opened and can be adapted to fit the specific AMC requirements as detailed in section 5. The binary can be generated (Build > Build solution) from the user code template (default) and downloaded into the MMC (Tools > Device programming) as shown on Figure 10 (Important: The “Erase device before programming” box must not be checked):

The original user code (template) allows starting the AMC with the default FRU information and power ON/OFF sequence. This template does not implement any custom sensors.

Note: If an uRTM is used with the AMC module, the files present in the user_code directory must be replaced with those present under “user_code (with uRTM)” before compiling. The next section describes how to perform the same action without JTGA (using the HPM.1 feature).

2.5 Downloading/upgrading the MMC application (HPM.1)

The MMC application can be downloaded/upgraded via HPM.1 (download performed via IPMI commands; no JTAG programmer required) with the HPMDownloader program (included in the SVN repository). This software was originally written for Linux and can be used with Cygwin on Windows (The installation procedure is given in Appendix A).

The MMC application must first be built as described in section 2.4. The default/cern_mmc.hex file generated at compilation time will be used to program the MMC. In parallel, the HPMDownloader software must be compiled by executing the make command (using Cygwin or a Linux terminal –
libcrypto and libssl are required) from the tool directory. Finally, the MMC Application program can then be downloaded by running the "./hpmdownloader <.hex file>" command, where <.hex file> is the binary generated previously. During execution, some information is requested to create the HPM image used to check the target compatibility. Thus the importance to register the appropriate IANA and product ID numbers as explained in section 2.3 of this document. Below (Figure 11) is an example of MMC programming using the HPMDownloader software:

Developpé by Julian Mendez <julian.mendez@cern.ch>

MMC Information
 ******************
IANA Manufacturer ID: 0x80
Product ID: 0x1235
Earliest major firmware rev. compatible: 1
Earliest minor firmware rev. compatible: 0
New major firmware rev. compatible: 1
New minor firmware rev. compatible: 1

Download Information
 **********************
MCH IP: 137.138.63.10
Username: admin
Password: admin
Slot(s) (should be separated by comma - all for all slots): 1

Downloading firmware
 ***********************
[INFO] {main}
[INFO] {Action detected}
[INFO] {Upgrade action detected}
[INFO] {Upgrade action detected}
[INFO] {Upgrade action detected}
[INFO] {get_img_information}
[INFO] {check_hpm_info}
[INFO] {check_hpm_info}
[INFO] {Upgrade in progress}
[INFO] {Upgrade action}

Programming MMC slot 1
Upload firmware image
Upgrade for component 0
Upgrade to version 10.1
"CERN MMC" firmware
HPM image check successful
version 1.0 will be replace by 1.1
HPM image check successful
19770 / 19776
Upgrade success

Failure report
 *****************
No failure

Figure 11: MMC programming using HPM.1
## 3 Hardware

### 3.1 Mezzanine pinout

<table>
<thead>
<tr>
<th>Pin</th>
<th>Pin name</th>
<th>µC</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12 Volts</td>
<td>PF0 / ADC[0]</td>
<td>12 volts monitoring</td>
</tr>
<tr>
<td>2</td>
<td>GPIO[0] / FPGA nReset</td>
<td>PC2</td>
<td>User / FPGA nReset IO</td>
</tr>
<tr>
<td>3</td>
<td>GPIO[1] / FPGA2 Init done</td>
<td>PC4</td>
<td>User / FPGA2 Init done IO</td>
</tr>
<tr>
<td>4</td>
<td>GPIO[2] / FPGA nReload</td>
<td>PC3</td>
<td>User / FPGA nReload IO</td>
</tr>
<tr>
<td>5</td>
<td>GPIO[3] / FPGA1 Init done</td>
<td>PC5</td>
<td>User / FPGA1 Init done IO</td>
</tr>
<tr>
<td>6</td>
<td>Green nLED</td>
<td>PB6</td>
<td>Green LED (AMC front)</td>
</tr>
<tr>
<td>7</td>
<td>MMC SCL</td>
<td>PD4</td>
<td>Local I2C bus (SCL)</td>
</tr>
<tr>
<td>8</td>
<td>Blue nLED</td>
<td>PB7</td>
<td>Blue LED (Mandatory, AMC front)</td>
</tr>
<tr>
<td>9</td>
<td>MMC SDA</td>
<td>PD5</td>
<td>Local I2C bus (SDA)</td>
</tr>
<tr>
<td>10</td>
<td>Red nLED</td>
<td>PB5</td>
<td>Red LED (AMC front)</td>
</tr>
<tr>
<td>11</td>
<td>Handle switch nCLOSED</td>
<td>PD2</td>
<td>Handle switch input</td>
</tr>
<tr>
<td>12</td>
<td>GND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Low voltage POK</td>
<td>PA0</td>
<td>Low voltage failure detection</td>
</tr>
<tr>
<td>14</td>
<td>IPMB-L SCL</td>
<td>PD0</td>
<td>IPMB-L bus (SCL)</td>
</tr>
<tr>
<td>15</td>
<td>GA1</td>
<td>PB2</td>
<td>Geographical address bit 1</td>
</tr>
<tr>
<td>16</td>
<td>IPMB-L SDA</td>
<td>PD1</td>
<td>IPMB-L bus (SDA)</td>
</tr>
<tr>
<td>17</td>
<td>GA0</td>
<td>PB1</td>
<td>Geographical address bit 0</td>
</tr>
<tr>
<td>19</td>
<td>nPS1</td>
<td>PE2</td>
<td>Present signal (PS1)</td>
</tr>
<tr>
<td>20</td>
<td>GPIO[5] / DCDC Enable</td>
<td>PC7</td>
<td>User / DCDC enable IO</td>
</tr>
<tr>
<td>21</td>
<td>nPS0</td>
<td>PE3</td>
<td>Present signal (PS0)</td>
</tr>
<tr>
<td>22</td>
<td>GA 2</td>
<td>PB3</td>
<td>Geographical address 2</td>
</tr>
<tr>
<td>23</td>
<td>GPIO[13] / RTM PS</td>
<td>PA2</td>
<td>User (no RTM) / RTM present signal</td>
</tr>
<tr>
<td>24</td>
<td>AMC nENABLE</td>
<td>RESET</td>
<td>AMC enable N signal</td>
</tr>
<tr>
<td>25</td>
<td>GPIO[14] / RTM 12 volts enable</td>
<td>PA3</td>
<td>User (no RTM) / RTM 12V enable signal</td>
</tr>
<tr>
<td>27</td>
<td>GPIO[15] / RTM 3.3 volts enable</td>
<td>PA4</td>
<td>User (no RTM) / RTM 3.3V enable signal</td>
</tr>
<tr>
<td>29</td>
<td>GND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>GPIO[16] / RTM I2C Enable</td>
<td>PA5</td>
<td>User (no RTM) / RTM I2C enable signal</td>
</tr>
<tr>
<td>32</td>
<td>Master TCK</td>
<td>PG0</td>
<td>Master JTAG (TCK)</td>
</tr>
<tr>
<td>33</td>
<td>GPIO[9]</td>
<td>PE6</td>
<td>User / Payload IO</td>
</tr>
<tr>
<td>34</td>
<td>Master TMS</td>
<td>PG1</td>
<td>Master JTAGA (TMS)</td>
</tr>
<tr>
<td>35</td>
<td>GPIO[10]</td>
<td>PE4</td>
<td>User / Payload IO</td>
</tr>
<tr>
<td>36</td>
<td>Master TDO</td>
<td>PG2</td>
<td>Master JTAG (TDO)</td>
</tr>
<tr>
<td>37</td>
<td>GPIO[11]</td>
<td>PE5</td>
<td>User / Payload IO</td>
</tr>
<tr>
<td>38</td>
<td>Local TDI</td>
<td>PG3</td>
<td>Master JTAG (TDI)</td>
</tr>
<tr>
<td>39</td>
<td>3.3 Volts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>GPIO[12]</td>
<td>PE7</td>
<td>User / Payload IO</td>
</tr>
</tbody>
</table>

Table 1: MMC mezzanine pinout
4 HPM.1

HPM.1 (Hardware Platform Management 1) is a PICMG standard specifying remote firmware upgrade features of IPM Controller (e.g.: MMC). HPM.1 support requires a boot loader program to be downloaded into the microcontroller flash memory to re-write the flash application memory. The HPM directory present in the SVN directory contains the boot loader program for the MMC microcontroller. The MMC microcontroller flash architecture is shown below (Figure 12):

![Figure 12: MMC microcontroller flash memory architecture](image)

The Application Flash Section can be erased and written by the boot loader program that contains the HPM project binary. The MMC application is stored in the application flash section. Upon startup, the microcontroller begins at the first boot loader section address with the following algorithm (Figure 13) that decides which of the HPM or MMC application should be executed:

![Figure 13: Start-up algorithm](image)
5 MMC Application software

This section describes the features implemented on the CERN MMC as well as how to customize the source code according to the specific AMC project needs and requirements.

5.1 Commands implemented

The table below (Table 2) lists the IPMI commands supported by the MMC program. These commands are specified by the IPMI 1.5 [1] and PICMG ATCA [4]/AMC.0 [2] standards. In addition, OEM commands (NetFN 2Eh) and/or controller specific commands (NetFN 30h) can optionally be implemented by user.

<table>
<thead>
<tr>
<th>Name</th>
<th>NetFN</th>
<th>Cmd</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get device id</td>
<td>06h</td>
<td>01h</td>
<td>Get device information</td>
</tr>
<tr>
<td>Broadcast get device id</td>
<td>06h</td>
<td>01h</td>
<td>Idem, broadcasted to all devices</td>
</tr>
<tr>
<td>Set event receiver</td>
<td>04h</td>
<td>00h</td>
<td>Set event register IPMB address</td>
</tr>
<tr>
<td>Get event receiver</td>
<td>04h</td>
<td>01h</td>
<td>Get event register IPMB address</td>
</tr>
<tr>
<td>Get device SDR info</td>
<td>04h</td>
<td>20h</td>
<td>Get information about SDR</td>
</tr>
<tr>
<td>Get device SDR</td>
<td>04h</td>
<td>21h</td>
<td>Read SDR register(s)</td>
</tr>
<tr>
<td>Reserve device SDR repo.</td>
<td>04h</td>
<td>22h</td>
<td>Get reservation ID used to read SDR</td>
</tr>
<tr>
<td>Get sensor threshold</td>
<td>04h</td>
<td>27h</td>
<td>Get threshold for specified sensor</td>
</tr>
<tr>
<td>Get sensor reading</td>
<td>04h</td>
<td>2Dh</td>
<td>Get sensor value (raw)</td>
</tr>
<tr>
<td>Get FRU inventory area info</td>
<td>0Ah</td>
<td>10h</td>
<td>Get information about FRU</td>
</tr>
<tr>
<td>Read FRU data</td>
<td>0Ah</td>
<td>11h</td>
<td>Read FRU information byte(s)</td>
</tr>
<tr>
<td>Write FRU data</td>
<td>0Ah</td>
<td>12h</td>
<td>Write FRU information byte(s)</td>
</tr>
<tr>
<td>Get properties</td>
<td>2Ch</td>
<td>00h</td>
<td>Get device properties</td>
</tr>
<tr>
<td>FRU control</td>
<td>2Ch</td>
<td>04h</td>
<td>FRU control (reset, reboot …)</td>
</tr>
<tr>
<td>Get FRU led properties</td>
<td>2Ch</td>
<td>05h</td>
<td>Get properties of specified LED</td>
</tr>
<tr>
<td>Get led colour capabilities</td>
<td>2Ch</td>
<td>06h</td>
<td>Get colour capabilities of specified LED</td>
</tr>
<tr>
<td>Set FRU led state</td>
<td>2Ch</td>
<td>07h</td>
<td>Set LED state (ON/OFF, blink, lamp test)</td>
</tr>
<tr>
<td>Get FRU led state</td>
<td>2Ch</td>
<td>08h</td>
<td>Get state of specified LED</td>
</tr>
<tr>
<td>Get device locator record</td>
<td>2Ch</td>
<td>0Dh</td>
<td>Read device locator record</td>
</tr>
<tr>
<td>Set power level</td>
<td>2Ch</td>
<td>11h</td>
<td>Control power level</td>
</tr>
<tr>
<td>Set AMC port state</td>
<td>2Ch</td>
<td>19h</td>
<td>Enable/Disable AMC port</td>
</tr>
<tr>
<td>Get AMC port state</td>
<td>2Ch</td>
<td>1Ah</td>
<td>Get AMC port state</td>
</tr>
<tr>
<td>FRU control capabilities</td>
<td>2Ch</td>
<td>1Eh</td>
<td>Get control capabilities (reboot, reset …)</td>
</tr>
<tr>
<td>Set clock state</td>
<td>2Ch</td>
<td>2Ch</td>
<td>Foreseen</td>
</tr>
<tr>
<td>Get clock state</td>
<td>2Ch</td>
<td>2Dh</td>
<td>Foreseen</td>
</tr>
</tbody>
</table>

Table 2: IPMI and PICMG commands implemented

The table below (Table 3) lists the HPM commands supported by the CERN MMC.

<table>
<thead>
<tr>
<th>Name</th>
<th>NetFN</th>
<th>Cmd</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get target upgrade capabilities</td>
<td>2Ch</td>
<td>2Eh</td>
<td>Get upgrade features capabilities</td>
</tr>
<tr>
<td>Get component properties</td>
<td>2Ch</td>
<td>2Fh</td>
<td>Get component information (version …)</td>
</tr>
<tr>
<td>Initiate upgrade action</td>
<td>2Ch</td>
<td>30h</td>
<td>Start an upgrade action</td>
</tr>
<tr>
<td>Upload firmware block</td>
<td>2Ch</td>
<td>32h</td>
<td>Upload firmware binary</td>
</tr>
<tr>
<td>Finish firmware upload</td>
<td>2Ch</td>
<td>33h</td>
<td>Check the binary size and finalize the upgrade</td>
</tr>
<tr>
<td>Get upgrade status</td>
<td>2Ch</td>
<td>34h</td>
<td>Get information about the ongoing upgrade</td>
</tr>
</tbody>
</table>

Table 3: HPM commands implemented
5.2 User code architecture

The CERN MMC source code is divided into two parts: core and user. In principle, users only need to modify the files present in the user_code directory:

- **Config file (config.h):**
  - Product information (IANA, Product ID, Firmware version)
  - User IOs configuration
  - Payload sequences (Power ON/OFF, reboot, warm/cold reset)
  - Additional LEDs description
- **Fru info file (fru_info.h):**
  - FRU information (Language, Generator source information, enabled areas)
  - Board information area (Board name, serial number, product number, ...)
  - Product information area (Product name, manufacturer, serial number, ...)
  - Point to point connectivity record (E-Keying)
  - Clock configuration record (Clock E-Keying)
  - Module current record (Current limit)
- **Sensors files (sensors.h and sensors.c):**
  - Header file contains SDR descriptions (**Up to 42 SDRs (40 users + 2 Generic)**)
  - Source code file contains sensor initialisation function
  - Source code file contains sensor monitoring function (polled every 100ms)
- **User code files (user_code.c):**
  - OEM commands
  - Controller specific commands
  - User main function for bench top use
- **E-Keying management file (ekeying.c):**
  - AMC port initialization
  - User command for activation/de-activation of AMC port
  - User command for activation/de-activation of AMC clock

5.3 User GPIO

Some microcontroller pins can optionally be used as user defined GPIO. In addition, a few generic pins can have multiple usages as described in section 3.2. The default configuration of these pins corresponds to their main function (E.g.: default configuration of pin PC2 is output as required by the FPGA nReset signal). In any case, all GPIO compatible pins are freely configurable as shown below:

```c
/** USER GPIO INITIALIZATION */
#endif GPI00_DIR INPUT //Not defined -> FPGA nRESET config.
#endif GPI01_DIR INPUT //Not defined -> FPGA2 Init done config.
#endif GPI02_DIR INPUT //Not defined -> FPGA2 nReload config.
#endif GPI03_DIR INPUT //Not defined -> FPGA1 Init done config.
#endif GPI04_DIR INPUT //Not defined -> Regulator Enable config.
#endif GPI05_DIR INPUT //Not defined -> DCDC Enable config.
#define GPI06_DIR OUTPUT
#define GPI07_DIR OUTPUT
#define GPI08_DIR OUTPUT
#define GPI09_DIR OUTPUT
#define GPI10_DIR OUTPUT
#define GPI11_DIR OUTPUT
#define GPI12_DIR OUTPUT
```
5.4 Analog to Digital Converter (ADC)

Some microcontroller pins can optionally be used as analogue inputs. By default, these pins are configured as GPIO but can be redefined with the ENABLE_ADC<n> macro (n is the identifier of the ADC to be enabled). This configuration overwrites previous GPIO configuration for the dedicated pin. The example below shows how to enable ADC features for pin PF1 and PF2:

```c
/** USER ADC INITIALIZATION */
#define ENABLE_ADC1  //ADC[1] is associated with the PF1 pin.
#define ENABLE_ADC2  //ADC[2] is associated with the PF2 pin.
//#define ENABLE_ADC3 //ADC[3] is associated with the PF3 pin.
//#define ENABLE_ADC4 //ADC[4] is associated with the PF4 pin.
//#define ENABLE_ADC5 //ADC[5] is associated with the PF5 pin.
//#define ENABLE_ADC6 //ADC[6] is associated with the PF6 pin.
//#define ENABLE_ADC7 //ADC[7] is associated with the PF7 pin.
```

**Note:** ADC[0] is used for payload power monitoring

5.5 FRU information

The FRU information is divided in 5 zones: internal use area, chassis info area, board area, product info area and multi-record area. It is important to note that only the last three sections are used by the MMC software. All information required to generate the FRU information binary is defined in the user_code/fru_info.h file. The list below describes the fields of the different sections:

- **Board area info:** Contains the following information about the board
  - BOARD_MANUFACTURER: Manufacturer name (E.g.: “CERN”, max. 63 characters)
  - BOARD_NAME: Board name (E.g.: “AMC-Board”, max. 63 characters)
  - BOARD_SN: Board serial number (E.g.: “AMC-000001”, max. 63 characters)
  - BOARD_PM: Board Product number (E.g.: “00001”, max. 63 characters)

- **Product information area:** Contains the following information about the product
  - PRODUCT_MANUFACTURER: Product manufacturer name (max. 63 characters)
  - PRODUCT_NAME: Product name (E.g.: “MMC-Template”, max. 63 characters)
  - PRODUCT_PN: Product part number (E.g.: “0000”, max. 63 characters)
  - PRODUCT_VERSION: Product version (E.g.: “v1.0”, max. 63 characters)
  - PRODUCT_SN: Product serial number (E.g.: “sn:00000001”, max. 63 characters)
  - PRODUCT_ASSET_TAG: Product asset tag (E.g.: “”, max. 63 characters)

- **MultiRecord area:** Contains the following records
  - AMC point to point connectivity record (Optional)
  - AMC clock configuration record (Optional - foreseen)
  - Module current record (Mandatory)

The AMC point to point connectivity record is defined as shown below:

```c
#define AMC_POINT_TO_POINT_RECORD_LIST   
  <p2p_definition>     
  <p2p_definition>     
  ...
#define AMC_POINT_TO_POINT_RECORD_CNT   
  <number_of_p2p_definition>
```

**Note:** Only one <p2p_definition> must be set by port (only one ID by port).
Where `<p2p_definition>` should be one of the two macros described below:

- `GENERIC_POINT_TO_POINT_RECORD(id, amc_port, protocol, ext, matching)`  
  - **Id**: Should be from 0 to `AMC_POINT_TO_POINT_RECORD_CNT` and unique  
  - **Port**: PORT(n) where n is the AMC port number (from 0 to 11)  
  - **Protocol and ext:**
    - For protocol PCIE, extension could be:  
      - GEN1_NO_SSC  
      - GEN1_SSC  
      - GEN2_NO_SSC  
      - GEN2_SSC  
    - For protocol PCIE_ADV_SWITCHING, there is no extension (NO_EXT)  
    - For protocol ETHERNET, extension could be:  
      - BASE_1G_BX  
      - BASE_10G_BX4  
    - For protocol SERIAL_RAPID_IO, extension could be:  
      - MBAUD_1250 (1.25 GBauds)  
      - MBAUD_2500 (2.5 GBauds)  
      - MBAUD_3125 (3.125 GBauds)  
      - MBAUD_5000 (5.00 GBauds)  
      - MBAUD_6250 (6.25 GBauds)  
    - For protocol STORAGE, there is no extension (NO_EXT)  
  - **Matching:**
    - EXACT_MATCHES  
    - MATCHES_01  
    - MATCHES_10

- `OEM_POINT_TO_POINT_RECORD(id, port, oem_id, matching)`  
  - **Id**: Should be from 0 to `AMC_POINT_TO_POINT_RECORD_CNT` and unique  
  - **Port**: PORT(n) where n is the AMC port number (from 0 to 11)  
  - **Oem_id**: identifies the custom protocol (See OEM_GUID description below)  
  - **Matching:**
    - EXACT_MATCHES  
    - MATCHES_01  
    - MATCHES_10

To identify an OEM connectivity, the carrier manager uses one GUID (Global Unique ID) number based on 16 bytes. The GUID format is defined in the Attachment A of the Wired for Management Baseline, Version 2.0 specification [5]. A GUID generator can be found on the web (E.g.: [http://createguid.com/](http://createguid.com/)). The step given below shows how to register a GUID for an OEM protocol:

```
#define POINT_TO_POINT_OEM_GUID_CNT <number_of_guid>
```
#define POINT_TO_POINT_OEM_GUID_LIST \  <oem_guid (oem_id = 0)> \  <oem_guid (oem_id = 1)> \  <oem_guid (oem_id = ...)> 

Where <oem_guid> shall be:

- **OEM_GUID(guid_0_msb, g1, g2, g3, g4, g5, g6, g7, g8, g9, g10, g11, g12, g13, g14, guid_15_lsb)**
  - guid_15_lsb to guid_0_msb are the 16 GUID bytes.

Below is an example of an AMC point to point connectivity declaration (user_code FRU_INFO.H):

```c
#define POINT_TO_POINT_OEM_GUID_CNT    2
#define POINT_TO_POINT_OEM_GUID_LIST        \  OEM_GUID(0x97,0x47,0x06,0xa2,0x2a,0x98,0x48,0xa9,0xbd,0x96,0x7b,0xf3,0x48,0x91,0x36,0x0f) \  OEM_GUID(0x46,0xbd,0xe8,0x5f,0x44,0xbd,0x44,0x56,0xa8,0x44,0x8,0x78,0x9f,0x4a,0x7b,0x03,0xfa,0x71)
```

```c
#define AMC_POINT_TO_POINT_RECORD_CNT  5
#define AMC_POINT_TO_POINT_RECORD_LIST       \  GENERIC_POINT_TO_POINT_RECORD(0, PORT(0), ETHERNET, BASE_1G_BX, EXACT_MATCHES) \  OEM_POINT_TO_POINT_RECORD(1, PORT(4), 0, EXACT_MATCHES) \  GENERIC_POINT_TO_POINT_RECORD(2, PORT(5), PCIE, GEN1_NO_SSC, MATCHES_10) \  OEM_POINT_TO_POINT_RECORD(1, PORT(6), 1, EXACT_MATCHES) \  GENERIC_POINT_TO_POINT_RECORD(4, PORT(7), PCIE, GEN1_NO_SSC, MATCHES_10)
```

### 5.6 SDR repository

The SDR (Sensor Data Record) information describes the sensors monitored by the MMC. Different record types exist and are described in the IPMI v1.5 standard. This document introduces only the two most frequently used types: Full sensors (E.g.: Temperature, voltage, current …) and Compact sensors (E.g.: flags …).

The sensors descriptions are present in user_code/sensors.h. Each SDR is a byte array described as follows (a commented example for a temperature sensor is available in the header file):

```c
#define AMC<id>_RECORD {                         \  SDR_byte[0], \  SDR_byte[1], \  ... \  SDR_byte[n], \}
```

**Note:** Each line, except the last one, have to finish with a backslash character.

**Note:** Comments must be surrounded with /* and */ only.

The <id> parameter is an identifier used for record generation and must be an integer from 0 to 17. The MMC core is limited to 18 user sensors for this release but it could be extended (contact the MMC technical person at CERN to increase this limit).

**Note:** The device locator, hotswap and 12V generic SDR are implemented in the core. They must be not defined in the user code. The device locator record must be customized via the FRU_NAME macro present in the config.h file.

The table below (Table 4) describes Full Sensor records type (01h):
<table>
<thead>
<tr>
<th>Byte</th>
<th>Name</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:2</td>
<td>Record ID</td>
<td>2</td>
<td>Filled by a MMC core function, set to 0000h</td>
</tr>
<tr>
<td>3</td>
<td>SDR Version</td>
<td>1</td>
<td>SDR Version, set to 51h for this version</td>
</tr>
<tr>
<td>4</td>
<td>Record type</td>
<td>1</td>
<td>Full sensor, set to 01h</td>
</tr>
<tr>
<td>5</td>
<td>Record length</td>
<td>1</td>
<td>Filled by a MMC core function, set to 00h</td>
</tr>
<tr>
<td>6</td>
<td>Sensor owner ID</td>
<td>1</td>
<td>Filled by a MMC core function, set to 00h</td>
</tr>
<tr>
<td>7</td>
<td>Sensor owner LUN</td>
<td>1</td>
<td>Filled by a MMC core function, set to 00h</td>
</tr>
<tr>
<td>8</td>
<td>Sensor number</td>
<td>1</td>
<td>Unique sensor number, set by user</td>
</tr>
<tr>
<td>9</td>
<td>Entity ID</td>
<td>1</td>
<td>Entity ID, set to C1h for AMC</td>
</tr>
<tr>
<td>10</td>
<td>Entity Instance</td>
<td>1</td>
<td>Filled by a MMC core function, set to 00h</td>
</tr>
<tr>
<td>11</td>
<td>Sensor initialization</td>
<td>1</td>
<td>Bit 6: Init scanning (1b: Enable / 0b: Disable)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 5: Init Events (1b: Enable / 0b: Disable)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 4: Init Thresholds (1b: Enable / 0b: Disable)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 3: Init hysteresis (1b: Enable / 0b: Disable)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 2: Init sensor type (1b: Enable / 0b: Disable)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 1: Set to 00h</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 0: Set to 00h</td>
</tr>
<tr>
<td>12</td>
<td>Sensor capabilities</td>
<td>1</td>
<td>Set to F6h to be compliant with MMC core</td>
</tr>
<tr>
<td>13</td>
<td>Sensor type</td>
<td>1</td>
<td>Sensor types are listed in table 36-3, Sensor Type Code of the IPMI v1.5 standard.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>E.g.: Temp. (01h), Voltage (02h), Current (03h) ....</td>
</tr>
<tr>
<td>14</td>
<td>Event type code</td>
<td>1</td>
<td>Event type code are listed in table 36-1, Event/Reading Type Code Range of the IPMI v1.5 standard.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>E.g.: Threshold events (01h)</td>
</tr>
<tr>
<td>15:16</td>
<td>Assertion Event Mask</td>
<td>2</td>
<td>Described in Table 37-1, Full Sensor Record of the IPMI v1.5 standard.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>E.g.: 7Fh to enable all assertion events</td>
</tr>
<tr>
<td>17:18</td>
<td>Deassertion Event Mask</td>
<td>2</td>
<td>Described in Table 37-1, Full Sensor Record of the IPMI v1.5 standard.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>E.g.: 7Fh to enable all deassertion events</td>
</tr>
<tr>
<td>19:20</td>
<td>Discrete reading Mask</td>
<td>2</td>
<td>00FFh to be compliant with the MMC</td>
</tr>
<tr>
<td>21</td>
<td>Sensor units 1</td>
<td>1</td>
<td>Bits [7:6]: Analog (numeric) data format</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00b: unsigned</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01b: 1’s complement (signed)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10b: 2’s complement (signed)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11b: Does not return analog reading</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bits [5:3]: Rate unit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>000b: none</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>001b: per μs</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>010b: per ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>011b: per s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>100b: per minute</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>101b: per hour</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>110b: per day</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>111b: reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bits [2:1]: Modifier unit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00b: none</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01b: Basic unit / Modifier Unit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10b: Basic unit * modifier unit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11b: reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 0: Percentage</td>
</tr>
<tr>
<td>No.</td>
<td>Field</td>
<td>Width</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>------------------------------</td>
<td>-------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>22</td>
<td>Base unit</td>
<td>1</td>
<td>Described in Table 37-14, Sensor Unit Type Codes of the IPMI v1.5 standard. E.g.: Degrees C (01h), Amps (05h), Volt (05h) ...</td>
</tr>
<tr>
<td>23</td>
<td>Modifier unit</td>
<td>1</td>
<td>Idem, 00h if not specified</td>
</tr>
<tr>
<td>24</td>
<td>Linearization</td>
<td>1</td>
<td>enum (linear, ln, log10, log2, e, exp10, exp2, 1/x, sqrt(x), cube(x), sqrt(x), cube-1 (x) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00h: linear</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>70h: non-linear. 71h-7Fh: non-linear, OEM defined</td>
</tr>
<tr>
<td>25</td>
<td>M</td>
<td>1</td>
<td>M, LS 8 bits (Sens_val = (M x Raw + B x 10^{Bexp}) + 10^{Rexp})</td>
</tr>
<tr>
<td>26</td>
<td>M, Tolerance</td>
<td>1</td>
<td>Bits[7:6]: M, MS 2 bits Bits[5:0]: Tolerance, unsigned in +/- ½ raw</td>
</tr>
<tr>
<td>27</td>
<td>B</td>
<td>1</td>
<td>B, LS 8 bit (Sens_val = (M x Raw + B x 10^{Bexp}) + 10^{Rexp})</td>
</tr>
<tr>
<td>28</td>
<td>B, Accuracy</td>
<td>1</td>
<td>Bits[7:6]: B, MS 2 bits Bits[5:0]: Accuracy, LS 6 bits in raw</td>
</tr>
<tr>
<td>29</td>
<td>Accuracy, Accuracy exp</td>
<td>1</td>
<td>Bits[7:4]: Accuracy, MS 4 bits in raw Bits[3:2]: Accuracy exp, 2 bits unsigned Bits[1:0]: Reserved, set to 00b</td>
</tr>
<tr>
<td>30</td>
<td>R exp, B exp</td>
<td>1</td>
<td>Bits[7:4]: R exponent, 4 bits 2’s complement signed Bits[3:0]: B exponent, 4 bits 2’s complement signed</td>
</tr>
<tr>
<td>31</td>
<td>Analog characteristic flags</td>
<td>1</td>
<td>Set to 07h to be compliant with the MMC core</td>
</tr>
<tr>
<td>32</td>
<td>Nominal reading</td>
<td>1</td>
<td>Nominal sensor value in raw</td>
</tr>
<tr>
<td>33</td>
<td>Normal maximum</td>
<td>1</td>
<td>Normal maximum value in raw</td>
</tr>
<tr>
<td>34</td>
<td>Normal minimum</td>
<td>1</td>
<td>Normal minimum value in raw</td>
</tr>
<tr>
<td>35</td>
<td>Sensor maximum reading</td>
<td>1</td>
<td>Max reading val (E.g.: FFh for unsigned, 7Fh for signed)</td>
</tr>
<tr>
<td>36</td>
<td>Sensor minimum reading</td>
<td>1</td>
<td>Min reading val (E.g.: 00h for unsigned, 80h for signed)</td>
</tr>
<tr>
<td>37</td>
<td>Upper Non-rec. threshold</td>
<td>1</td>
<td>Upper non-recoverable threshold value in raw</td>
</tr>
<tr>
<td>38</td>
<td>Upper critical threshold</td>
<td>1</td>
<td>Upper critical threshold value in raw</td>
</tr>
<tr>
<td>39</td>
<td>Upper non-crit. threshold</td>
<td>1</td>
<td>Upper non-critical threshold value in raw</td>
</tr>
<tr>
<td>40</td>
<td>Lower non-rec. threshold</td>
<td>1</td>
<td>Lower non-recoverable threshold value in raw</td>
</tr>
<tr>
<td>41</td>
<td>Lower critical threshold</td>
<td>1</td>
<td>Lower critical threshold value in raw</td>
</tr>
<tr>
<td>42</td>
<td>Lower non-crit. threshold</td>
<td>1</td>
<td>Lower non critical threshold value in raw</td>
</tr>
<tr>
<td>43</td>
<td>Positive going hysteresis</td>
<td>1</td>
<td>Positive going threshold hysteresis value in raw (00h: no hysteresis)</td>
</tr>
<tr>
<td>44</td>
<td>Negative going-hysteresis</td>
<td>1</td>
<td>Negative going threshold hysteresis value in raw (00h: no hysteresis)</td>
</tr>
<tr>
<td>45</td>
<td>Reserved</td>
<td>1</td>
<td>Set to 00h</td>
</tr>
<tr>
<td>46</td>
<td>Reserved</td>
<td>1</td>
<td>Set to 00h</td>
</tr>
<tr>
<td>47</td>
<td>OEM</td>
<td>1</td>
<td>Set to 00h</td>
</tr>
<tr>
<td>48</td>
<td>ID String type/length</td>
<td>1</td>
<td>Bits[7:6]: Type (E.g.: 11b for ASCII) Bits[5:0]: ID string length in bytes</td>
</tr>
<tr>
<td></td>
<td>ID String bytes</td>
<td>N</td>
<td>Sensor ID string bytes</td>
</tr>
</tbody>
</table>

Table 4: Full sensor SDR data

The table below (Table 5) describes Compact Sensor records type (02h):
<table>
<thead>
<tr>
<th>Byte</th>
<th>Name</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:2</td>
<td>Record ID</td>
<td>2</td>
<td>Filled by a MMC core function, set to 0000h</td>
</tr>
<tr>
<td>3</td>
<td>SDR Version</td>
<td>1</td>
<td>SDR Version, set to 51h for this version</td>
</tr>
<tr>
<td>4</td>
<td>Record type</td>
<td>1</td>
<td>Full sensor, set to 01h</td>
</tr>
<tr>
<td>5</td>
<td>Record length</td>
<td>1</td>
<td>Filled by a MMC core function, set to 00h</td>
</tr>
<tr>
<td>6</td>
<td>Sensor owner ID</td>
<td>1</td>
<td>Filled by a MMC core function, set to 00h</td>
</tr>
<tr>
<td>7</td>
<td>Sensor owner LUN</td>
<td>1</td>
<td>Filled by a MMC core function, set to 00h</td>
</tr>
<tr>
<td>8</td>
<td>Sensor number</td>
<td>1</td>
<td>Unique sensor number, set by user</td>
</tr>
<tr>
<td>9</td>
<td>Entity ID</td>
<td>1</td>
<td>Entity ID, set to C1h for AMC</td>
</tr>
<tr>
<td>10</td>
<td>Entity Instance</td>
<td>1</td>
<td>Filled by a MMC core function, set to 00h</td>
</tr>
<tr>
<td>11</td>
<td>Sensor initialization</td>
<td>1</td>
<td>Bit 6: Init scanning (1b: Enable / 0b: Disable)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 5: Init Events (1b: Enable / 0b: Disable)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 4: Reserved, set to 00h</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 3: Init hysteresis (1b: Enable / 0b: Disable)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 2: Init sensor type (1b: Enable / 0b: Disable)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 1: Set to 00h</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 0: Set to 00h</td>
</tr>
<tr>
<td>12</td>
<td>Sensor capabilities</td>
<td>1</td>
<td>Set to F6h to be compliant with MMC core</td>
</tr>
<tr>
<td>13</td>
<td>Sensor type</td>
<td>1</td>
<td>Sensor types are listed in table 36-3, Sensor Type Code of the IPMI v1.5 standard. E.g.: Temp. (01h), Voltage (02h), Current (03h) ...</td>
</tr>
<tr>
<td>14</td>
<td>Event type code</td>
<td>1</td>
<td>Event type code are listed in table 36-1, Event/Reading Type Code Range of the IPMI v1.5 standard. E.g.: Threshold events (01h)</td>
</tr>
<tr>
<td>15:16</td>
<td>Assertion Event Mask</td>
<td>2</td>
<td>Described in Table 37-1, Full Sensor Record of the IPMI v1.5 standard. E.g.: 7Fh to enable all assertion events</td>
</tr>
<tr>
<td>17:18</td>
<td>Deassertion Event Mask</td>
<td>2</td>
<td>Described in Table 37-1, Full Sensor Record of the IPMI v1.5 standard. E.g.: 7Fh to enable all deassertion events</td>
</tr>
<tr>
<td>19:20</td>
<td>Discrete reading Mask</td>
<td>2</td>
<td>00FFh to be compliant with the MMC</td>
</tr>
<tr>
<td>21</td>
<td>Sensor units 1</td>
<td>1</td>
<td>Bits [7:6]: Analog (numeric) data format</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00b: unsigned</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01b: 1’s complement (signed)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10b: 2’s complement (signed)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11b: Does not return analog reading</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bits [5:3]: Rate unit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00b: none</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>001b: per μS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>010b: per ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>011b: per s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>100b: per minute</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>101b: per hour</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>110b: per day</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>111b: reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bits [2:1]: Modifier unit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00b: none</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01b: Basic unit / Modifier Unit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10b: Basic unit * modifier unit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11b: reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 0: Percentage</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Base unit</td>
<td>1</td>
<td>Described in Table 37-14, Sensor Unit Type Codes of the IPMI v1.5 standard. E.g.: Degrees C (01h), Amps (05h), Volt (05h)</td>
</tr>
<tr>
<td>23</td>
<td>Modifier unit</td>
<td>1</td>
<td>Idem, 00h if not specified</td>
</tr>
<tr>
<td>24:25</td>
<td>Sensor Record Sharing</td>
<td>2</td>
<td>Bits[15:14]: Reserved, set to 00h Bits[13:12]: ID String instance modifier type 00b: numeric 01b: alpha Bits[11:8]: Share count (number of sensor sharing this record). E.g.: if the starting sensor number was 10, and the share count was 3, then sensors 10, 11 and 12 would share this record. Bit[7]: Entity instance sharing. Set to 1b Bits[6:0]: ID String Instance modifier offset Multiple Discrete sensors can share the same sensor data record. The ID string Instance Modifier and Modifier Offset are used to modify the Sensor ID string as follows: Suppose sensor ID is “Temp” for ‘Temperature Sensor’, share count = 3, ID string instance modifier = numeric, instance modifier offset = 5 - then the sensors could be identified as: Temp 5, Temp 6, Temp 7. If the modifier = alpha, offset=0 corresponds to ‘A’, offset=25 corresponds to ‘Z’, and offset = 26 corresponds to ‘AA’, thus, for offset=26 the sensors could be identified as: Temp AA, Temp AB, Temp AC (alpha characters are considered to be base 26 for ASCII)</td>
</tr>
<tr>
<td>26</td>
<td>Positive going hysteresis</td>
<td>1</td>
<td>Positive going threshold hysteresis value in raw (00h: no hysteresis). Note: Cannot use shared record if sensors require individual hysteresis settings.</td>
</tr>
<tr>
<td>27</td>
<td>Negative going-hysteresis</td>
<td>1</td>
<td>Negative going threshold hysteresis value in raw (00h: no hysteresis). Note: Cannot use shared record if sensors require individual hysteresis settings.</td>
</tr>
<tr>
<td>45</td>
<td>Reserved</td>
<td>1</td>
<td>Set to 00h</td>
</tr>
<tr>
<td>46</td>
<td>Reserved</td>
<td>1</td>
<td>Set to 00h</td>
</tr>
<tr>
<td>47</td>
<td>OEM</td>
<td>1</td>
<td>Set to 00h</td>
</tr>
<tr>
<td>48</td>
<td>ID String type/length</td>
<td>1</td>
<td>Bits[7:6]: Type (E.g.: 11b for ASCII) Bits[5:0]: ID string length in bytes</td>
</tr>
<tr>
<td>49:+N</td>
<td>ID String bytes</td>
<td>N</td>
<td>Sensor ID string bytes</td>
</tr>
</tbody>
</table>

Table 5: Compact sensor SDR data
5.7 Power management

The power management (DC/DC control, Regulators enabling, etc) of an AMC must be taken care of by the MMC. The standard defines a few sequences to be executed: power ON sequence (Mandatory), power OFF sequence (Mandatory), reboot (Optional), warm reset (Optional) or cold reset (Optional). These sequences are user defined (depend on the AMC requirements).

The sequences configuration is described below (user_code/config.h):

```c
#define POWER_ON_SEQ
   <action>
   ...

#define POWER_OFF_SEQ
   <action>
   ...

#define REBOOT_SEQ
   <action>
   ...

#define WARM_RESET_SEQ
   <action>
   ...

#define COLD_RESET_SET
   <action>
   ...
```

Note: To disable optional sequences, the related define must be removed or commented out.

Note: Each line, except the last one, have to finish with a backslash character.

Note: Comments must be surrounded with /* and */ only.

The <action> parameter must be one of the macros described below:

- SET_PAYLOAD_SIGNAL(<signal>)
  - <signal>: name of the pin’s signal (Appendix A)
- CLEAR_PAYLOAD_SIGNAL(<signal>)
  - <signal>: name of the pin’s signal (Appendix A)
- WAIT_FOR_HIGH(<signal>,<timeout >,<error>)
  - <signal>: name of the pin’s signal (Appendix A)
  - <timeout>: timeout in ms
  - <error>: could be C code (executed in if statement) or one of the following macros
    - PON_ERROR: Send a power ON failure event to the MCH and then, execute POWER_OFF_SEQ.
    - PON_CONTINUE: Continue the in progress sequence
    - POFF_ERROR: Stop the sequence
• WAIT_FOR_LOW(<signal>,<timeout >,<error>)
  o  <signal>: name of the pin’s signal (Appendix A)
  o  <timeout>: timeout in ms
  o  <error>: could be C code (executed in if statement) or one of the following macros
    ▪  PON_ERROR: Send a power ON failure event to the MCH and then, execute
      POWER_OFF_SEQ.
    ▪  PON_CONTINUE: Continue the in progress sequence
    ▪  POFF_ERROR: Stop the sequence
• SET_REGISTER(<reg_name>,<value>)
  o  <reg_name>: register name (e.g.: DDRx, PORTx …)
  ▪  <value>: value to set in the register
• DELAY(<delay >)
  o  <delay>: delay in ms
• CUSTOM_C(<c_code>)
  o  <c_code>: user’s C code to perform custom action (configure i2C component …)

Below is an example of the power ON sequence:

```c
#define POWER_ON_SEQ          
SET_PAYLOAD_SIGNAL(LOCAL_DCDC_ENABLE)   /* Set LOCAL_DCDC_ENABLE signal to high */   
SET_PAYLOAD_SIGNAL(LOCAL_REG_ENABLE)  /* Set LOCAL_REG_ENABLE signal to high */ 
WAIT_FOR_LOW(LOCAL_FPGA1_INIT_DONE, 500, PON_ERROR)    /* Wait for LOCAL_FPGA1_INIT_DONE */ 
SET_REGISTER(DDRA, (DDRA & 0x0F) | 0x04)       /* Set DDRA register */  
CUSTOM_C(           /* Execute custom c code */  
/* Put your C code here (below is an example) */      
int myvar = 0;          
for(myvar =0; myvar < 5; myvar ++)       
  PORTA = get_ipmi_address()+myvar;       
)             
DELAY(500)    /* Delay of 500 ms */     
CLEAR_PAYLOAD_SIGNAL(GPIO_1)  /* Set GPIO_1 signal to low */
```

5.8  E-keying

The e-keying specification, included in the AMC.0 standard [2], describes an automatic feature to
check port compatibility between modules installed on a common backplane. The AMC ports must be
defined in the FRU information (section 5.5). This information, complemented with the backplane
information, is used to check the compatibility during the AMC initialization phase. Upon AMC board
insertion and when the payload power is enabled, the carrier manager sends the “set AMC Port State
(Enable)” command to enable all compatible AMC ports. Similarly, during the extraction procedure,
the AMC ports are disabled with the “set AMC port State (Disable)” command that is sent before the
payload power is disabled. The three user functions defined below (user_code/ekeying.c) are used to
perform initialization, enabling and disabling actions:

• void ekeying_init()
  o  By default, all AMC ports are defined as disabled. Users can modify it by calling the
    following function: set_channel_init(<desc_id>, <state>).
    ▪  <desc_id>: defines the port specified. The port is selected by the associated
      ID defined in the point to point record of the FRU information (section 5.5).
    ▪  <state>: Define the initialization state and can be:
      •  PORT_ACTIVE
      •  PORT_INACTIVE
- **u08 port_ekeying_enable(u08 id)**
  - This function is called when the “set AMC port state (Enable)” command is received from the carrier manager. In this function, the user can configure devices (E.g.: enable interface, configure switch ...) with C functions. The macros defined in section 5.10 can also be used.
  - Id: defines the specified port. The port is selected by the associated ID defined in the point to point record of the FRU information (section 5.5).
  - Returned value:
    - SUCCES
    - FAILED
    - NI (Not Implemented, inform that this port cannot be controlled)

- **u08 port_ekeying_enable(u08 id)**
  - This function is called when the “set AMC port state (disabled)” command is received from the carrier manager. In this function, the user can configure devices (E.g.: disable interface, remove switch configuration ...) with C functions. The macros defined in section 5.10 can also be used.
  - Id: defines the specified port. The port is selected by the associated ID defined in the point to point record of the FRU information (section 5.5).
  - Returned value:
    - SUCCES
    - FAILED
    - NI (Not Implemented, inform that this port cannot be controlled)

### 5.9 Sensors

Two functions are dedicated to the user’s sensors. One for initialization and one for monitoring. These functions are implemented in the user_code/sensors.c file with the following prototypes: `void sensor_init_user()` and `void sensor_monitoring_user()`.

How to send/get information to/from an I2C based sensor:

- **GET_I2C_VAL(<address>,<sub_addr>,<sub_addr_type>,<len>,<buf>)**
  - <address>: I2C address of the device
  - <sub_addr>: Sub-address / register address
  - <sub_addr_type>: length of the sub-address
    - ZERO_LEN: no sub-address
    - BYTE_LEN: 1 sub-address byte
    - SHORT_LEN: 2 sub-address byte
  - <len>: number of byte to be read
  - <buf>: pointer to unsigned char (u08) array where the read byte should be stocked
- **SEND_I2C_VAL(address, sub_addr, sub_addr_type, len, buf)**
  - `<address>`: I2C address of the device
  - `<sub_addr>`: Sub-address / register address
  - `<sub_addr_type>`: length of the sub-address
    - ZERO_LEN: no sub-address
    - BYTE_LEN: 1 sub-address byte
    - SHORT_LEN: 2 sub-address byte
  - `<len>`: number of byte to be sent
  - `<buf>`: pointer to unsigned char (u08) array containing bytes to be sent

How to read an analogue value from a microcontroller ADC pin:

- **GET_ADC(chID)**
  - `<chID>`: ADC channel ID to be read
  - Returned value: unsigned char (u08) – value read by the ADC

How to set/clear a MMC microcontroller GPIO pin:

- **SET_SIGNAL(signal)**
  - `<signal>`: name of the pin’s signal (Appendix B)
- **CLEAR_SIGNAL(signal)**
  - `<signal>`: name of the pin’s signal (Appendix B)

How to register the sensor value:

- **set_sensor_value(sensor_number, value)**
  - `<sensor_number>`: sensor number specified in the SDR
  - `<value>`: unsigned char (u08) value

### 5.10 Custom LEDs

The PICMG standard defines functions to control and monitor LEDs. Three LEDs are described in the specification: the blue led (mandatory), green led (optional – implemented) and red led (optional – implemented). Moreover, custom LEDs can be defined and controlled with the same functions or by the MMC application itself. The LEDs are implemented as described below:

```c
#define AMC_USER_LED_LIST { 
  <led array (ledID = 3)>, 
  <led array (ledID = 4)>, 
  ... 
  <led array (ledID = n)> 
}
```

**Note:** the ledID is automatically incremented and starts from 3.

**Note:** Each line, except the last one, have to finish with a backslash character and comments must be surrounded with /* and */ only.
Where `<led array>` can be one of the two arrays described below (depending on the way the LED is controlled):

- **LED controlled through IO Extender (PCF8574 family):**

  ```
  {       
    io_type: IO_EXTENDER_PCF8574AT,   
    addr: `<i2c_address>`   
    pin: `<pin number>`    
    colour: `<colour>`    
    init: `<init_state>`    
    active: `<pin_state>`    
    inactive: `<pin_state>`   
  },
  ``

  - `<i2c_address>`: I²C address of the IO extender
  - `<pin number>`: Pin of the IO extender (from 1 to 8)
  - `<colour>`: LED colour, can be one of the following value
    - BLUE
    - RED
    - GREEN
    - AMBER
    - ORANGE
    - WHITE
  - `<init_state>`: Initialization state of the LED, can be one of the following value
    - ACTIVE (led ON)
    - INACTIVE (led OFF)
  - `<pin_state>`: pin value for active/inactive state, can be one of the following value
    - LOW
    - HIGH

- **LED controlled by a MMC microcontroller pin:**

  ```
  {       
    io_type: MMC_PORT,    
    port: `<port>`     
    pin: `<pin>`     
    colour: `<colour>`     
    init: `<init_state>`     
    active: `<pin_state>`     
    inactive: `<pin_state>`    
  },
  ``

  - `<port>`: pin’s port must be set as follow: PORT(<signal>)
    - `<signal>`: name of the pin’s signal (Appendix B)
  - `<pin>`: MMC pin, set as follow: PIN(<signal>)
    - `<signal>`: name of the pin’s signal (Appendix B)
5.11 OEM commands

The OEM commands defined by the IPMI v1.5 standard [1] are user defined IPMI commands. These can be executed with the netFN 2Eh. To enable this feature, the ENABLE_OEM macro must be set in the user_code/config.h file and the ipmi_oem_user function must be implemented in the user_code/user_code.c file.

Enable oem command (user_code/config.h):

```c
#define ENABLE_OEM
```

ipmi_oem_user function:

```c
u08 ipmi_oem_user(u08 cmd , u08 *iana, u08 * data, u08 data_len, u08 *buf, u08 *error)
```

- `<cmd>`: command received
- `<iana>`: 3 bytes array containing the IANA identifier (must be checked)
- `<data>`: data received
- `<data_len>`: number of received data bytes
- `<buf>`: bytes array pointer to set data to be sent (3 first bytes must be the IANA)
- `<error>`: IPMI completion code value (see Appendix C)
- Returned value: Number of bytes to be sent (response)

Below is an example of the implementation of the following OEM command:

- Command: 01h
- IANA: 000060h
- Cmd data length: 1 byte
- Cmd data: data[0] = <eg0>
- Returned data: data[0] = <eg0>

```c
u08 ipmi_oem_user(u08 cmd, u08 *iana, u08 * data, u08 data_len, u08 *buf, u08 *err){
    //Note: You must not return more than "MAX_BYTES_READ" bytes
    u08 res_data_len = 0;
    *err = IPMI_CC_OK;
    if (iana[0] != 0x00 || iana[1] != 0x00 || iana[2] != 0x60){
        *error = IPMI_CC_PARAM_OUT_OF_RANGE;
        return 0;             //Wrong IANA code
    }
    buf[0] = buf[1] = 0x00;   //IANA ID must be returned as the first 3 bytes
    buf[2] = 0x60;
    switch(command){
        case 0x01: if(data_size == 1){
            buf[3] = user_data[0];
            res_data_len = 4;
        }else{
            *err = IPMI_CC_PARAM_OUT_OF_RANGE;
            return 0;
        }
        break;
    }
}```
Controller specific commands

The controller specific commands are defined in the IPMI v1.5 standard [1]. These can be executed with the netFN 30h. To enable this feature, the ENABLE_CONTROLLER_SPECIFIC macro must be set in the user_code/config.h file and the ipmi_controller_spec function must be implemented in the user_code/user_code.c file.

Enable controller specific command (user_code/config.h):

```c
#define ENABLE_CONTROLLER_SPECIFIC
```

ipmi_oem_spec function:

```c
u08 ipmi_controller_spec(u08 cmd, u08 *data, u08 data_len, u08 *buf, u08 *err)
```
- <cmd>: command received
- <data>: data received
- <data_len>: number of received data bytes
- <buf>: bytes array pointer to set data to be sent (3 first bytes must be the IANA)
- <err>: IPMI completion code value (see Appendix C)
- Returned value: Number of bytes to be sent (response)

Below is an example of the implementation of the following controller specific command:

- Command: 01h
- Cmd data length: 1 byte
- Cmd data: data[0] = <eg0>
- Returned data: data[0] = <eg0>

```c
u08 ipmi_controller_specific(u08 cmd, u08 *data, u08 data_len, u08 *buf, u08 *err){
  u08 rsp_length = 0;
  *error = IPMI_CC_OK;

  switch(command){
    case 0x01:
      buf[0] = data[0];
      rsp_length = 1;
      break;
    default:
      *error = IPMI_CC_INV_CMD;
  }
  return rsp_length;
}
```
5.13 Bench top

A custom geographical addresses can be defined to power the AMC card in bench top use (outside of an xTCA shelf). These addresses must be outside of the PICMG specification range and can be defined in the user_code/config.h file as follow:

```c
#define CUSTOM_ADDR_LIST
  ADDR(<addr>,<gaddr_0>,<gaddr_1>,<gaddr_2>)
...```

Where:

- `<addr>`: custom address, should be out of PICMG specification (E.g.: FFh)
- `<gaddr_0>`: value for GA0 pin associated to this address
  - POWERED: Bit is set
  - GROUNDED: bit is cleared
  - UNCONNECTED: bit is unconnected
- `<gaddr_1>`: value for GA1 pin associated to this address (same as above)
- `<gaddr_2>`: value for GA2 pin associated to this address (same as above)

If custom addresses are used, the user_main function should be implemented in the user_code/user_code.c file as follow:

```c
u08 user_main(u08 addr){ /* user code */
  while(1)
    manage_payload();
    return 1;
}
```

Moreover, sensors can be monitored with the function sensor_monitoring_user() defined in user_code/sensors.c. The sensor values can be read with:

```c
get_sensor_value(u08 sensID)
  sensID: sensor number defined in SDR
  Returned value: raw sensor value
```
Appendix A: Cygwin installation on Windows

Cygwin is a Unix-like command line interface for Windows. It can be downloaded from the project website (https://cygwin.com/install.html). During the installation process, the following packet must be selected to use the MMC related tools:

- Make
- GCC
- LibSSL

Make:
Gcc:

LibSSL:
## Appendix B: MMC microcontroller signals

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA0</td>
<td>LOCAL_LOW_VOLTAGE_POK</td>
</tr>
<tr>
<td>PA2</td>
<td>RTM_PS or GPIO_13</td>
</tr>
<tr>
<td>PA3</td>
<td>RTM_12V_ENABLE or GPIO_14</td>
</tr>
<tr>
<td>PA4</td>
<td>RTM_3V3_ENABLE or GPIO_15</td>
</tr>
<tr>
<td>PA5</td>
<td>RTM_I2C_ENABLE or GPIO_16</td>
</tr>
<tr>
<td>PB0</td>
<td>GA_PULLUP</td>
</tr>
<tr>
<td>PB1</td>
<td>GA2</td>
</tr>
<tr>
<td>PB2</td>
<td>GA1</td>
</tr>
<tr>
<td>PB3</td>
<td>GA0</td>
</tr>
<tr>
<td>PB5</td>
<td>LOCAL_RED_LED</td>
</tr>
<tr>
<td>PB6</td>
<td>LOCAL_GREEN_LED</td>
</tr>
<tr>
<td>PB7</td>
<td>LOCAL_BLUE_LED</td>
</tr>
<tr>
<td>PC2</td>
<td>LOCAL_RESET_FPGA or GPIO_0</td>
</tr>
<tr>
<td>PC3</td>
<td>LOCAL_RELOAD_FPGA or GPIO_2</td>
</tr>
<tr>
<td>PC4</td>
<td>LOCAL_FPGA2_INIT_DONE or GPIO_1</td>
</tr>
<tr>
<td>PC5</td>
<td>LOCAL_FPGA1_INIT_DONE or GPIO_3</td>
</tr>
<tr>
<td>PC6</td>
<td>LOCAL_REG_ENABLE or GPIO_4</td>
</tr>
<tr>
<td>PC7</td>
<td>LOCAL_DCDC_ENABLE or GPIO_5</td>
</tr>
<tr>
<td>PD0</td>
<td>IPMB_SCL</td>
</tr>
<tr>
<td>PD1</td>
<td>IPMB_SDA</td>
</tr>
<tr>
<td>PD2</td>
<td>LOCAL_HANDLE_SWITCH</td>
</tr>
<tr>
<td>PD4</td>
<td>LOCAL_I2C_SCL</td>
</tr>
<tr>
<td>PD5</td>
<td>LOCAL_I2C_SDA</td>
</tr>
<tr>
<td>PE2</td>
<td>PS1</td>
</tr>
<tr>
<td>PE3</td>
<td>PS0</td>
</tr>
<tr>
<td>PE4</td>
<td>GPIO_10</td>
</tr>
<tr>
<td>PE5</td>
<td>GPIO_11</td>
</tr>
<tr>
<td>PE6</td>
<td>GPIO_9</td>
</tr>
<tr>
<td>PE7</td>
<td>GPIO_12</td>
</tr>
<tr>
<td>PF0</td>
<td>PRESENCE_12V</td>
</tr>
<tr>
<td>PF1</td>
<td>GPIO_6</td>
</tr>
<tr>
<td>PF2</td>
<td>GPIO_7</td>
</tr>
<tr>
<td>PF3</td>
<td>GPIO_8</td>
</tr>
<tr>
<td>PG0</td>
<td>MASTER_TCK</td>
</tr>
<tr>
<td>PG1</td>
<td>MASTER_TMS</td>
</tr>
<tr>
<td>PG2</td>
<td>MASTER_TDO</td>
</tr>
<tr>
<td>PG3</td>
<td>MASTER_TDI</td>
</tr>
</tbody>
</table>
8 References


https://www.picmg.org/openstandards/advanced-mezzanine-card/

https://www.picmg.org/openstandards/hardware-platform-management/

https://www.picmg.org/openstandards/advancedtca/


https://www.picmg.org/openstandards/microtca/
9 Table of figures

Figure 1: ATCA Hardware Platform Management ................................................................. 3
Figure 2: MicroTCA Hardware Platform Management .......................................................... 4
Figure 3: Activation / De-activation state machine .............................................................. 5
Figure 4: SVN repository architecture .................................................................................. 6
Figure 5: USB driver installation ......................................................................................... 7
Figure 6: Atmel Studio installation ...................................................................................... 7
Figure 7: Atmel Studio environment .................................................................................... 8
Figure 8: MMC microcontroller fuse configuration .............................................................. 9
Figure 9: MMC bootloader programming using JTAG ......................................................... 9
Figure 10: MMC application programming via JTAG ......................................................... 10
Figure 11: MMC programming using HPM.1 .................................................................... 11
Figure 12: MMC microcontroller flash memory architecture ............................................ 13
Figure 13: Start-up algorithm ............................................................................................. 13

10 Table of tables

Table 1: MMC mezzanine pinout .......................................................................................... 12
Table 2: IPMI and PICMG commands implemented ........................................................ 14
Table 3: HPM commands implemented ............................................................................. 14
Table 4: Full sensor SDR data ........................................................................................... 20
Table 5: Compact sensor SDR data .................................................................................... 22