MMC Implementation for the MTCA Devices Used in X-FEL

Jaroslaw Szewinski, NCBJ Swierk, Otwock, Poland
Uros Mavric, Holger Schlarb, DESY, Hamburg, Germany

INTRODUCTION

MTCA becomes more and more popular standard of electronic devices used in the experimental physics. In comparison with older multi-drop parallel buses used in such standards like VME or CompactPCI, it brings new level of performance and reliability by enabling modern fast serial interfaces such as PCI Express or Gigabit Ethernet directly on the backplane.
Except the higher performance, MTCA provides also extensive management based on the IPMI protocol. This covers such aspects like unit identification, hot-swap control, power and interconnect management. Every AMC unit in the MTCA shelf must be equipped in Module Management Controller (MMC) which takes care of above issues. MMC is needed to put board into operational state and provide necessary information to the management unit (MCH - MicroTCA Carrier Hub) to enable power in the particular slot. Implementation of the MMC is not trivial, because it has to follow the IPMI 1.5 specification, and extensions defined by PICMG (consortium which defined the MTCA standard) has to be implemented as well.
MMC implementation seem to be a barrier for new users who would like to switch to MTCA, especially for those who have used standards like VME, where boards can be considered as "plug & play" in comparison with the MTCA. The X-FEL project will use MTCA.4 as a main standard for electronic devices, and for this purpose own MMC implementation has been developed.

MAIN GOALS & FEATURES

- IPMI support
- PICMG extensions support
- Multiple boards support
- Clear separation between platform dependent and independent code (for better portability)
- Unified sensors support
- Command line interface over serial port for debugging MMC itself, and for manual control

MicroTCA Architecture

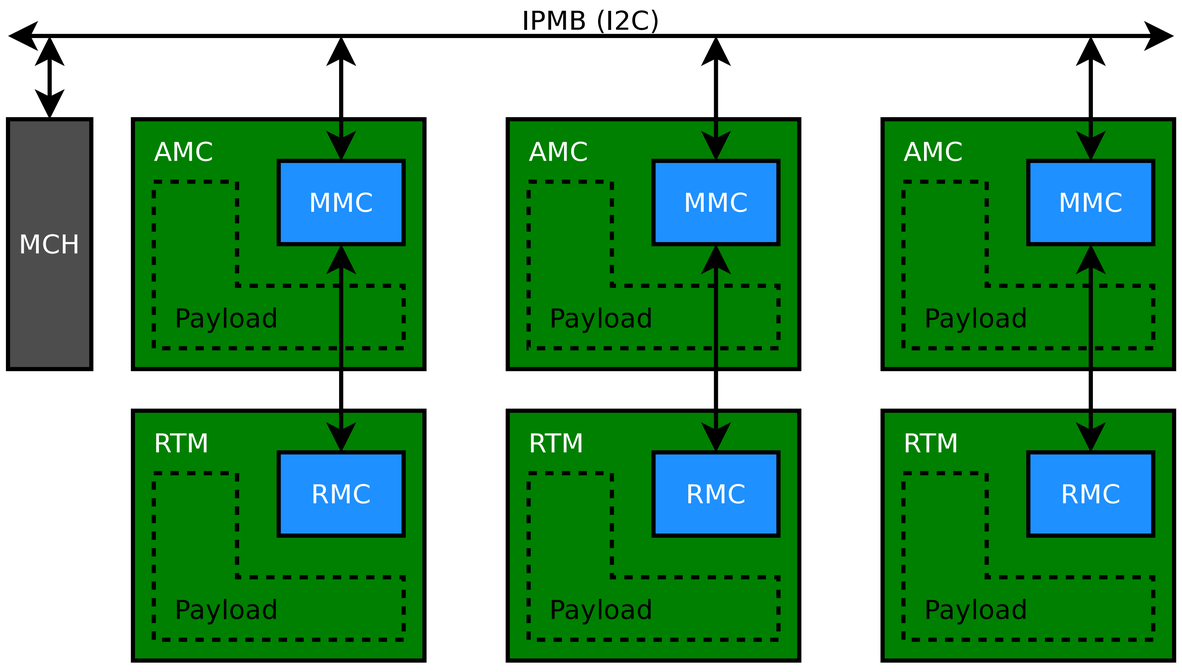


Figure 1: Architecture of MTCA.4 management

PORTABILITY

For portability reasons, code has been divided into platform dependent and platform independent parts. Platform independent should not require any changes during porting to another environment. Platform dependent part should be modified by user when:

- porting to another board with the same microcontroller
- porting to another microcontroller device

Platform independent part shall have no platform specific code (such as dedicated microcontroller function calls or types used). It should be pure ANSI C code, while all platform-specific code should be placed in the board specific part. In this context platform specific code is understood as:

- microcontroller specific code (support for the peripherals in vendor specific way)
- board specific code - regardless of the microcontroller, each AMC board has own specific resources, such as sensors, which has to be controlled by MMC.

All described above types of platform dependent code is placed together in the platform specific part which can be, and should be modified by user while porting to new environment.
Described approach requires interaction form user in some cases, but on the other hand it gives maximal flexibility. Other attempts of MMC code unification for different boards (even with the same microcontroller) was falling into problems in different cases.

MULTIPLE BOARDS SUPPORT

To provide support for multiple boards in easy way, user have to provide implementation of board specific functions.
To provide consistent interface between platform independent part and board specific part, a header file with function prototypes has been defined. This header file is included in platform independent code, which is calling abstract board specific functions like *turn on LED* or *send IPMI message over IPMB*. The general idea is to provide set of simple functions, where each function is responsible for one single task. This approach makes porting MMC to another platform easy, because user has only to provide own (adjusted to specified platform) implementations of these functions, without need of learning the IPMI basics.

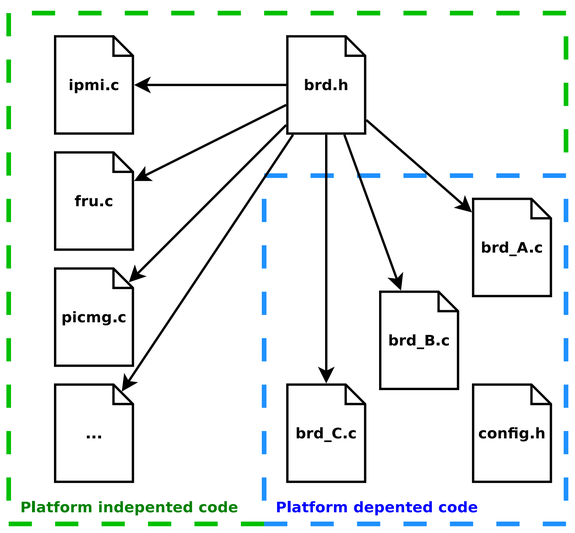


Figure 2: Platform dependent and platform independent code for multiple board support

SENSORS DATA STRUCTURES

To achieve simple and consistent sensor support, an architecture based on data structures has been defined (fig. 3). Presented solution is made of two levels of data structures: general data structures used by the platform independent code, and platform specific data structures related with the particular hardware resources. Interface between these two sections is made by abstract *read-sensor* functions, which has to return proper sensor value (w.r.t. to scaling and offset defined in the SDR record for that sensor).

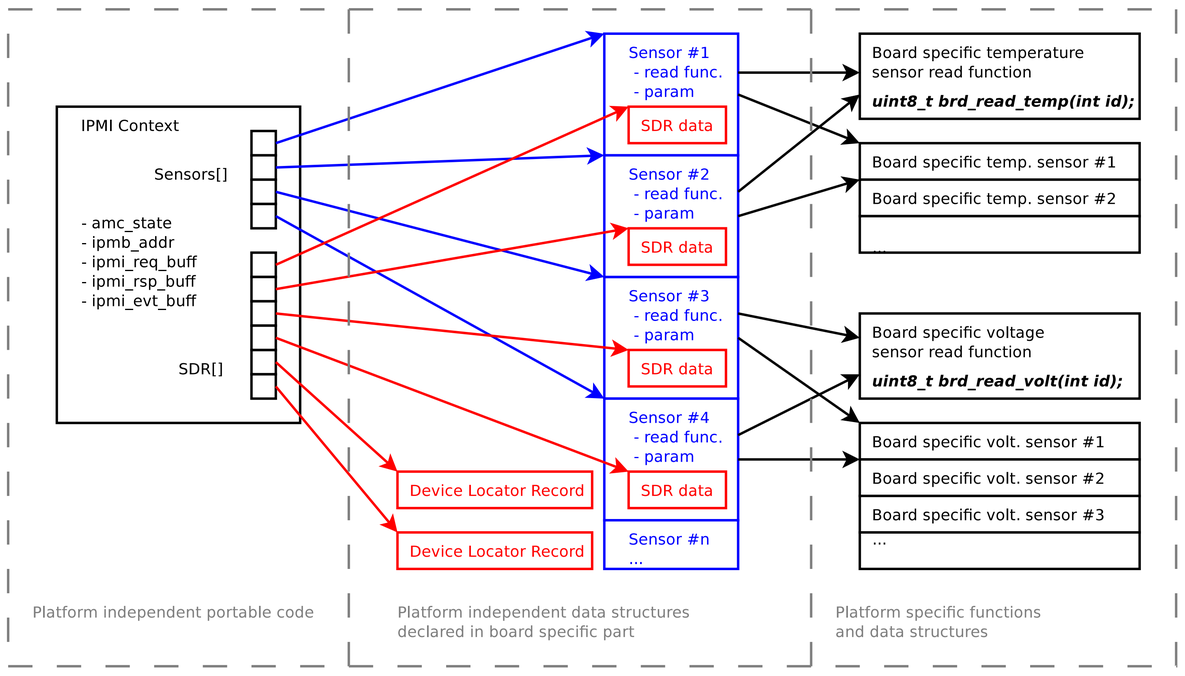


Figure 3: Sensor support architecture

SENSOR SUPPORT - CODE EXAMPLES

Data structures declaration:

```
enum
{
  SENS_HOTSWAP, // Hot-Swap sensor
  SENS_TEMP1,   // Temperature sensor
  SENS_TEMP2,   // Temperature sensor
  SENS_TEMP3,   // Temperature sensor
  SENS_TEMP4,   // Temperature sensor
  SENS_12V_PP,  // Voltage sensor
  SENS_3V3_PP,  // Voltage sensor
  SENS_2V5_PP,  // Voltage sensor
  SENS_MAX      // SENS_MAX must
                // be at the end
};

ipmi_sensor_t sensors[SENS_MAX];
```

Sensor and SDR data initialization:

```
mmc_sensor_init           (sensors, SENS_12V_PP, "12V PP");
mmc_sdr_nominal_reading   (sensors[SENS_12V_PP].sdr, 12.0);
mmc_sdr_normal_maximum    (sensors[SENS_12V_PP].sdr, 13.0);
mmc_sdr_normal_minimum    (sensors[SENS_12V_PP].sdr, 11.0);
mmc_sdr_sensor_maximum    (sensors[SENS_12V_PP].sdr, 255);
mmc_sdr_sensor_minimum    (sensors[SENS_12V_PP].sdr, 0);

ipmi_sensor_set_read_function(&sensors[SENS_12V_PP], adc_volt_read, 0);
ipmi_sensor_add(ipmi, &sensors[SENS_12V_PP]);
```

Update of all sensors:

```
for(i=0; i<SENS_MAX; i++)
{
    sensors[i].value = sensors[i].read(sensors[i].param);
}
```

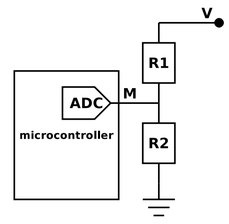
Example of platform specific support for the ADC-based voltage sensors using well known technique of on-board voltages measure using simple resistor voltage divider:

Sensor data structure:

```
typedef struct
{
    char* name;
    int   adc;
    float r1;
    float r2;
}
adc_voltage_sensor_t;
```

Array of all available sensors:

```
adc_voltage_sensor_t adc_voltage_sensors[] =
{
/*   name       adc  R1      R2,  */
    {"12V PP",  0,   470.0,  33.0 },
    {"3.3V PP", 1,   100.0,  33.0 },
    {"2.5V PP", 2,   100.0,  33.0 },
    {"3.3V MP", 3,   33.0,   3.3  }
};
```



Each voltage can be estimated using simple equation: $V = M * \frac{R_1+R_2}{R_2}$, where M is measured value by the ADC, and V is voltage which value has to estimated.

GENERAL OPERATION

General idea of MMC operation was that platform independent code (implementing the general IPMI protocol support, state machine, etc.) was calling simple functions from the board specific part, where each function was responsible for one operation, such as "turn on the LED" or "read part of the FRU record".
There are two exceptions from this general idea:

- Receiving IPMI request on IPMB
- Running periodic actions using timer event

Both mentioned exceptions are platform specific in case of particular microcontroller: I2C and timer support, and These are sources of asynchronous events (w.r.t. running code on the MCU). In in these two cases, the platform specific code has to call (or update data) in the general part of the MMC. All other platform depended interactions are simply synchronous function calls made from the main loop. Both, I2C and timer operation is usually based on the interrupts, and to minimize the time which MCU spends in the ISR (interrupt service routine), the ISR was only setting the flag that particular event (IPMB message or timer event) has occurred and all real work to handle that event was done during next iteration of the MMC main loop.

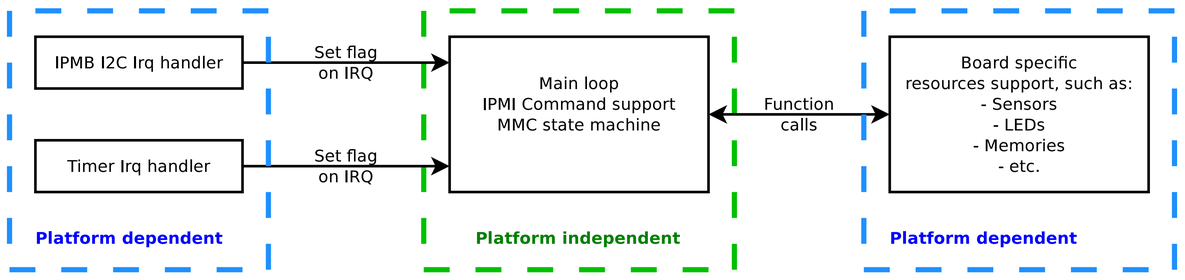


Figure 4: Interaction between platform independent and platform specific code

REFERENCES

1. IPMI - Intelligent Platform Management Interface Specification v1.5, Document Revision 1.1, February 20 2002
2. PICMG, http://www.picmg.org/
3. Original MMC implementation done by DESY MCS4 group: http://tesla.desy.de/~vahan/