openMMC: An open source modular firmware for board management

H. A. Silva, G. B. M. Bruno, LNLS, Campinas,
henrique.silva@lnls.br

Abstract

openMMC is an open-source firmware designed for board management in MicroTCA systems. It has a modular architecture providing decoupling between application, board and microcontroller-specific routines, making it useful as a base for many different designs, even those using less powerful controllers.

Despite being developed in a MicroTCA context, the firmware can be easily adapted to other hardware platforms and communication protocols.

The firmware is based on the FreeRTOS operating system, over which each monitoring function (sensors, LEDs, Payload management, etc) runs its own independent task. The OS, despite its reduced footprint, also provides numerous tools for reliable communication among the tasks, controlling the board efficiently.

FreeRTOS

Real Time kernel for embedded applications written in C.

- User code is organized as a collection of **parallel independent tasks**
- Implements a **priority-based preemptive scheduler**
- **Round-robin** time slicing algorithm for same priority tasks
- Task Communication tools, such as queues, semaphores, timers, etc.

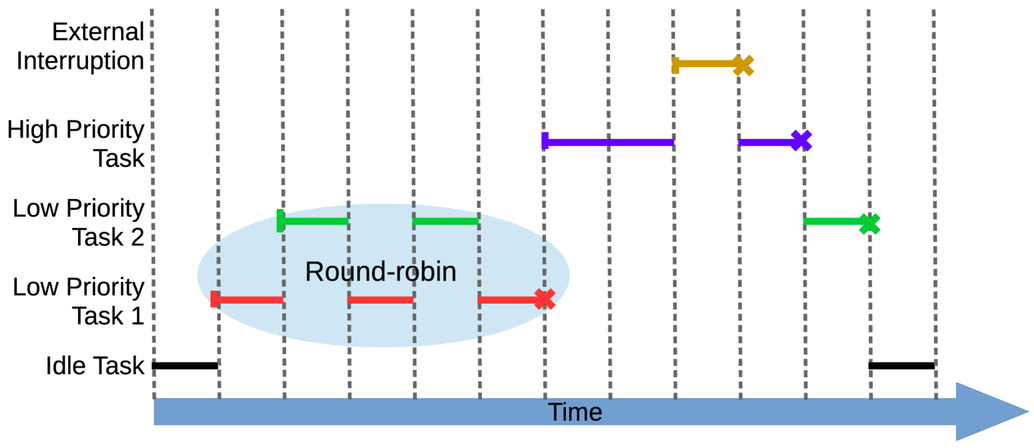

Figure 1: Demonstrates the execution flow of FreeRTOS-based application, in which two low priority and a high priority tasks are created. The scheduler implements the round-robin time slicing from t2 to t3. When the high priority task is enabled in t3, the scheduler switches context to it, interrupting low-priority task 2. An external interruption happens in t4, being serviced instead of the high priority task.

Firmware Structure

OBJECTIVE: Easy to **port** and **upgrade** to different boards and controllers (reusable code).

Four different abstraction layers implemented: **Application**, **Hardware Abstraction**, **Port** and **Driver**, as disposed in Figure 2.

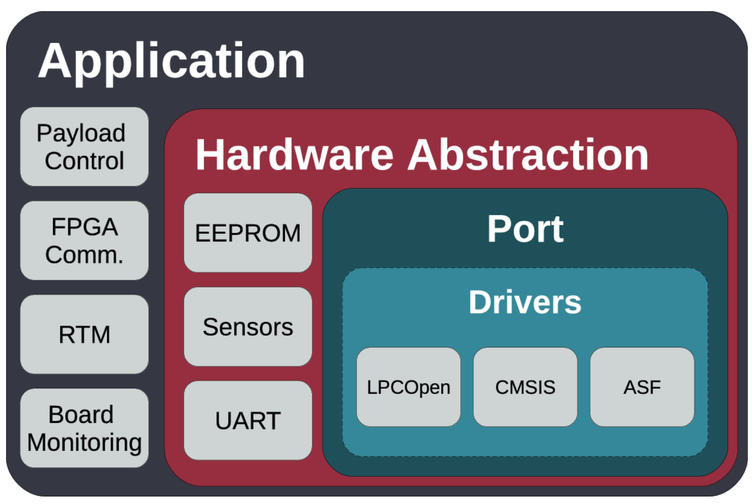

Figure 2: openMMC layers disposal with examples of the type of code present in each one (small gray boxes)

Firmware Structure cont.

Application: High level monitoring tasks (state-machines, sensors management, etc.)

Hardware Abstraction: Interfaces with application's specific peripheral hardware. Examples:

- SCANSTA111 JTAG Switch
- ADN4604 Clock Crossbar Switch
- AD84XX DAC
- AT24MAC EEPROM
- 24xx64 EEPROM
- LM75 Temperature Sensor
- MAX6642 Temperature Sensor
- INA220 Voltage and Current Sensor
- Si57X Oscillator
- PCA9554 I/O Expander

Port: Masks all functions provided by the controller's drivers to the upper layes

Drivers: Microcontroller internal hardware interfaces (e.g. LPCOpen, Atmel ASF, CMSIS)

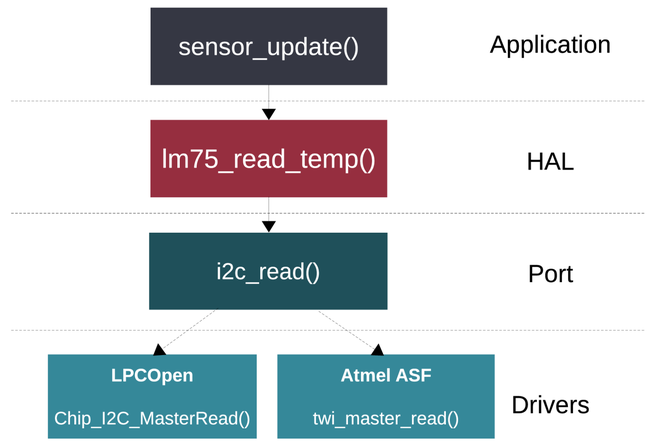

Figure 3: Example of the functions calls through openMMC's layers. The application task calls the generic function sensor_update(), which must call lm75_read_temp() to update the temperature readings. In HAL layer, the LM75 module uses an I2C read to interface with the IC. The I2C driver function is redirected from either LPCOpen or Atmel's ASF drivers.

Board Porting

CMake scripts are used to select each board's modules, according to its hardware. Examples of the CmakeLists.txt file selecting the modules are shown in Figure 3.

```
#AFC-BPM MODULES
set( AFCBPM_MODULES
  "FRU"
  "PAYLOAD"
  "SDR"
  "WATCHDOG"
  "CLOCK_SWITCH"
  "FPGA_SPI"
  "AD84XX_DAC"
  "EEPROM_AT24MAC"
  "HOTSWAP_SENSOR"
  "LM75"
  "MAX6642"
  "INA220"
  "HPM"
)
```

```
#TIMING AMC MODULES
set( TIMINGAMC_MODULES
  "FRU"
  "PAYLOAD"
  "SDR"
  "WATCHDOG"
  "PLL_CTRL"
  "WHITE_RABBIT"
  "HOTSWAP_SENSOR"
  "LM75"
  "RTM"
  "HPM"
)
```

Figure 3: Examples of the code that selects which modules will be compiled for two different board ports.

Acknowledgments

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