UF IPMC manual

A. Greshilov, UF

DRAFT 2021-01-07

# General software structure

## CONFIGURE folder.

This folder contains CONFIG.toml configuration file. There you can specify i2c adapter numbers (/dev/i2c-[**number**]) for IPMB-0 bus (IPMB-A + IPMB-B), i2c adapter numbers (/dev/i2c-[**number**]) for sensor access, bypass to manually manage **Handle Switch state** (2 values: first for enabling/disabling of manual handling -> second for setting of required state), bypass to manually manage **Payload state** (2 values: first for enabling/disabling of manual handling -> second for setting of required state), **power steady state** of the present unit (by default steady state = 190 Watt + 10 Watt).

As an example how to use the bypass of FRU Handle Switch without having real Handle Switch:

***>insert Power Board with ZYNQ into Shelf***

***>vim /root/UF\_IPMC/CONFIG/CONFIG.toml***

***>set man\_handle\_status to 1 (enable manual managing)***

***>set man\_handle\_switch to 0 (in case to stop IPMC ) or to 1 (in case to start IPMC)***

***>:wq + Enter***

For **Payload state** bypass is the same way to manage.

## FRU folder (FRU = Field Replaceable Unit)

This folder contains FRU.toml configuration file with all needed inventory information. E-Keying should be specified there as well if it’s required. For more detailed info see [1].

## SDR folder (Sensor Data Record).

This folder contains all SDR\*.toml configuration files. Existing SDRs (0,1,2,3) are reserved (**Management Controller Device Locator Record** (Type 13h) + **Fru Hot Swap sensor record** (Type 01h) + **IPMB-0 status sensor record** (Type 01h) + **Hot Swap Handle** (Type 01h)).

SDR0 is a general IPMC record which is reserved as **Management Controller Device Locator Record** (Type 13h). There you can specify general information about capabilities of present IPMC. For more details please see **p.545** in [2].

New added SDR’s should have **Full Sensor Record** type (Type 01h, see details on p.521 in [2]).

ATTENTION: Please make sure that all new SDR’s file name numbers and SDR’s table name numbers in folder go in the correct **ascending order** and there are **no other files** other than SDR\*.toml.

## SRC folder (Source files)

This folder contains all source files.

# Adding new sensors

**Step 1:**

Add I2C bus that will be used for this sensor. In **/root/UF\_IPMC/CONFIG/CONFIG.toml** you can find **[I2C\_SENSORS]** table where you can list all needed i2c bus numbers (**/dev/i2c-[number1], /dev/i2c-[number2], …**) in **i2c-sensor** array separated by commas **[number1,number2, …].**

Example:

If your sensor is using I2C bus #1, modify the line as follows:

**i2c\_sensor = [4,1]**

If your new sensor is read out using I2C bus that’s already declared, or without using I2C bus at all, then don’t modify **CONFIG.toml** file.

**Step 2:**

Add new SDR record.

Copy template **SDR\*.toml** file from **/root/UF\_IPMC/TEMPLATES** to **/root/UF\_IPMC/SDR** folder.

Rename this file to SDR**N**.toml, where **N** is the next integer number after the last used in SDR directory. **N** is your new sensor number.

Edit **SDRN.toml** file as follows**:**

* Replace [SDR\*] with [SDR**N**], where **N** is the sensor number. **N** should match the sensor number that you used to rename **SDR\*.toml** file above.
* Replace **record\_id[\*,0]**  with **record\_id[N,0]**, where **N** is your sensor number
* Calculate and rework **record\_len**. Follow formula:   
  **(25 + strlen(id\_string\_bytes))**, where **id\_string\_bytes** is the name of your new sensor as it will be reported in SDR
* **entity\_id** = by default **0x01** (other). Please see p.550 in [2] to determine the value of this parameter for your sensor
* **entity\_instance\_num** = by default **0x60** (could be set **0x60-0x7f**)
* **sensor\_type**. See p.505 in [2].
* **event\_type\_code**.See p.503 in [2].
* **sensor\_units2**.See p.554 in [2].

Study **chapter 36 – “Sensor Types and Data Conversion”**, p.482 in [2] and see “IPMI event thresholds and hysteresis” picture at the end of this document.

Also see p.526 in [2] and below for BYTE 21 parameters, starting with **analog\_data\_format**

Rework the following parameters as needed for your sensor:

* **linearization**
* **M**
* **M\_tolerance**
* **B**
* **B\_accuracy**
* **accuracy**
* **K1**
* **K2**
* **analog\_characteristic\_flags**
* **nominal\_reading**
* **normal\_maximum**
* **normal\_minimum**
* **sensor\_maximum\_reading**
* **sensor\_minimum\_reading**
* **upper\_non\_recoverable\_threshold**
* **upper\_critical\_threshold**
* **upper\_non\_critical\_threshold**
* **lower\_non\_recoverable\_threshold**
* **lower\_critical\_threshold**
* **lower\_non\_critical\_threshold**
* **positive\_going\_threshold\_hysteresis\_value**
* **negative\_going\_threshold\_hysteresis\_value**
* **oem**
* **id\_str\_typ\_len** = **0xC0** + (number of following string bytes).See p.551 in [2].
* **id\_string\_bytes.** Provide sensor name in this string array following the example in **/root/UF\_IPMC/TEMPLATES/SDR\*.toml** .

**Step 3:**

Add the source code for the new sensor:

Implementation of custom sensors should be done in the **user-sensor.c/user-sensor.h** files. In the future, in case of updates to the **UF IPMC** software, so as not to rewrite the custom code for the sensors, it will be very easy to compile the new version by simply **replacing** these files.

Open **user-sensor.c** file. In **user\_module\_sensor\_init()** method you need specify callback scanning function name you want to create (see **/root/UF\_IPMC/TEMPLATES/template.c**).

Declare callback function in **user-sensor.h** header file.

At the end of **user-sensor.c** file you need to implement the callback function as **‘void [*name*](void)’.**

In the callback function body, assign the sensor reading value to **sd[*current\_number\_of\_sensor*].last\_sensor\_reading** variable.

ATTENTION: **current\_number\_of\_sensor = 0,1,2,3** are reserved. User sensors **must** start with **current\_number\_of\_sensor = 4**.

A **Hot Swap Event Messages** can be defined for **Temperature sensor** implementations. See the implementation of the **Dummy Temperature sensor** for an example). This is used to let the Shelf Manager know if the temperature of that sensor is out of normal limits. The Shelf Manager will react appropriately, by increasing or decreasing the fan speed. For more details please see the **Temperature Hot Swap Event message** structure at the end of this document.

Reading sensors using i2c bus can be done through **i2c\_read()** and **i2c\_write()** library methods from **i2c-sensor.h**. The prototypes of these functions are shown below:

**int i2c\_read(int i2c\_fd\_snsr[i2c\_device\_number], u8 chip\_addr, u8 register\_read, u8 \*result)**

**int i2c\_write(int i2c\_fd\_snsr[i2c\_device\_number], u8 chip\_addr, u8 register\_write, u8 data)**

The “**i2c\_device\_number**” is the number of this sensor’s I2C bus as listed in **CONFIG.toml** file. For example if you have this in **CONFIG.toml**:

**i2c\_sensor = [4,1]**

then i2c bus 4 is accessible as **i2c\_fd\_snsr[0]**, and i2c bus 1 is accessible as **i2c\_fd\_snsr[1]**.

To be able to read and write **firmware** registers you need to use defined in **i2c.c** file **reg\_read()** and **reg\_write()** functions:

**void reg\_write(void \*reg\_base, unsigned int offset, unsigned int value);**

**unsigned int reg\_read(void \*reg\_base, unsigned int offset);**

See please as example of using these functions in **module\_payload\_on()** and **module\_payload\_off()** functions in **ipmc.c** file at the end of file.

ATTENTION: You have to use **offsets** of registers with the base **reg = 0x40000000**. If you want to use firmware registers from software that are outside of the range which is defined in the **i2c.c** file, you need to rework it or separately set the required **mmap()** range, but be careful with that.

To manage payload state you have to use **module\_payload\_on()** and **module\_payload\_off()** implemented methods.

To increase the number of sensors redefine **MAX\_SENSOR\_COUNT** and **MAC\_SDR\_COUNT** macros in **ipmc.c, sensor.c,** and **i2c-sensor.c** source files.

To put a new sensor in a **scanning state** implement **state poll function** (just after your callback function) with the following structure (use **/root/UF\_IPMC/TEMPLATES/template.c** as an example):

**void [*name*]\_state\_poll( unsigned char \*arg)**

**{**

**unsigned char [*name*]\_timer\_handle;**

**scanning\_function(); //callback function**

**timer\_add\_callout\_queue( (void \*)&[*name*]\_timer\_handle,**

**[*time*]\*SEC, [*name*]\_state\_poll, 0 ); /\* [*time*] sec timeout \*/**

**}**

Declare ‘**void [*name*]\_poll( unsigned char \*arg)**’ function in the beginning of **user-sensor.c** file (go to “**User Local Function Prototypes**” location).

Call ‘**[*name*]\_state\_poll( 0 )**’ function in **user\_module\_init()** function just after other **\*\_state\_poll( 0 )** functions (go to “**State Poll Functions call**” location).

Note: Temperature sensors already pinged up by the Shelf Manager every **~5 secs** by default.

# Timer’s resolution is up to **100000ns**, which can be used in software for scan functions.

Examples:

**[time]\*SEC** value is **unsigned long**, **SEC = 10000**

1\*SEC for checking sensor each 1s

0.1\*SEC for each 100ms

0.01\*SEC for each 10ms

0.001\*SEC for each 1ms

0.0001\*SEC for each 0.1ms

This can be helpful in usage: **lbolt** – a system counter that increments every **0.1ms.** It is based on the **CLOCK\_MONOTONIC** system clock for **POSIX.1b**.

For logging to file **(/var/log/uf\_ipmc.log**) you should use **void logger(const char\* tag, const char\* fmt, ...)** function.

Examples:

***>logger(“ERROR”, “Fail description”)***

***>logger(“EVENT”, “Parameter = %d”, value)***

# Project compilation

Without any new source files added to **/root/UF\_IPMC/SRC**:

***>cd /root/UF\_IPMC/SRC***

***>make***

With the new source files added to **/root/UF\_IPMC/SRC** ( or first time compilation)**:**

***> cd /root/UF\_IPMC/SRC***

Add info about added files in ***/root/UF\_IPMC/SRC/CMakeLists.txt***

***>cmake .***

***>make***

# Debugging

# UF\_IPMC can be debugged under gdb in case of problems.

# Example (application crashing):

1. *gdb IPMC\_EXEC*
2. *(gdb) handle SIGUSR1 nostop noprint pass*
3. *(gdb) r*
4. *(gdb) bt full*

# Addition

# Needs to be enabled **core dumps** in system for debugging reason in case of unexpected crashes.

# For normal work **IPMC\_EXEC** can be installed as **rpm** package in root file system, then included in **systemd** service to be booted and restarted automatically.

# **uf\_ipmc.log** file should be managed by **logrotate** service for logs managing in automatic mode.

# Useful Shelf Manager commands

***>clia sel 20*** //to see event logs. There you can see all event messages from IPMCs

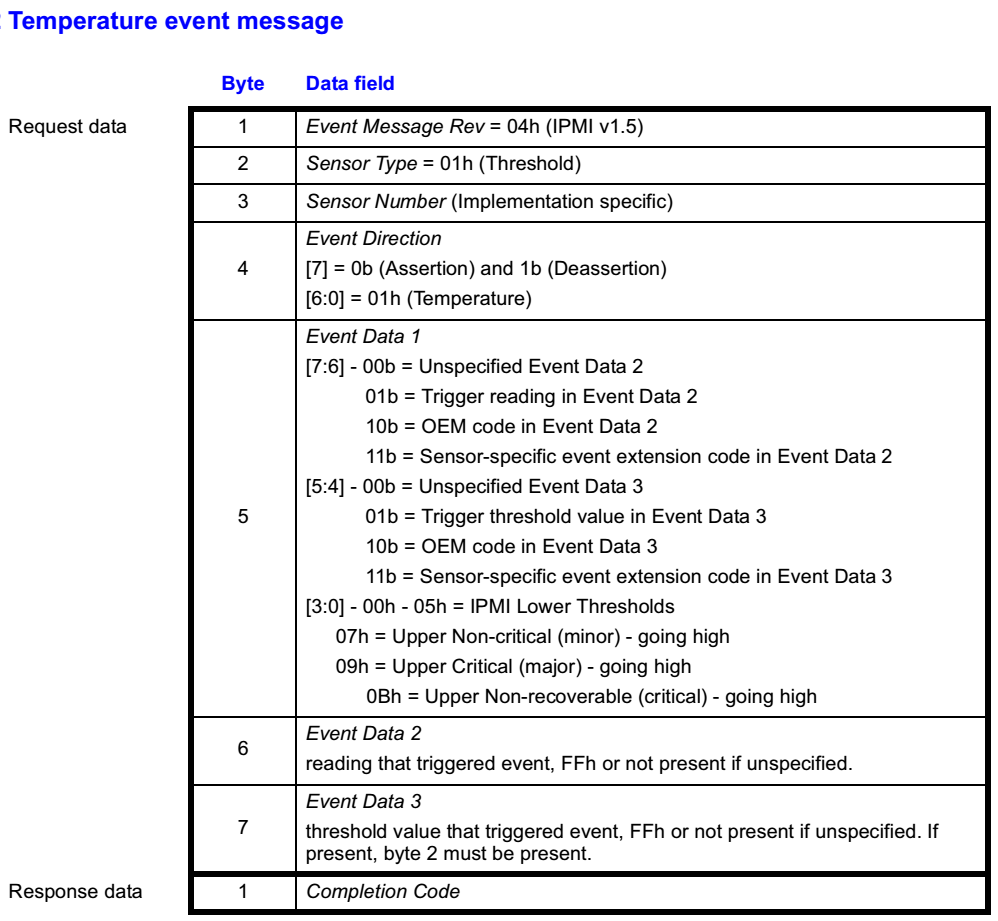
***>clia board -v -x [slot\_number]*** //to see general IPMC info

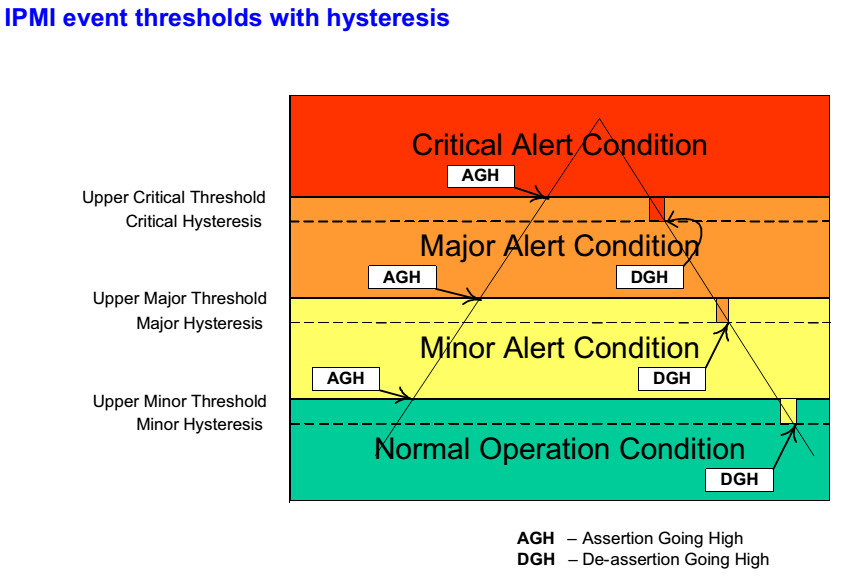
***>clia activate [IPMB\_addr] 0*** //activate IPMC from Shelf Manager

***>clia deactivate*** ***[IPMB\_addr] 0*** //deactivate IPMC from Shelf Manager

***>clia sensorread [IPMB\_addr] [sensor\_number]*** //read sensor info

***>clia sensordata [IPMB\_addr] [sensor\_number]*** //read sensor





# References

1. FRU info storage specification <https://www.intel.com/content/www/us/en/servers/ipmi/ipmi-platform-mgt-fru-infostorage-def-v1-0-rev-1-3-spec-update.html>
2. IPMI specification <https://www.intel.com/content/www/us/en/products/docs/servers/ipmi/ipmi-second-gen-interface-spec-v2-rev1-1.html>