IPM Controller Data Sheet

Revision History:

Rev 1.5: *25.03.2015 Added information on selecting the sensor type for GPIO Sensors*
Rev 1.4: *24.03.2015 Changed Format,Added information for GPIO Sensors*
Rev 1.3*: 15.01.2015 Corrections*
Rev 1.2: *07.11.2013 Modified Additional Auxiliary Circuits chapter*
Rev 1.1: *12.09.2013 Added Chapter E-keying*
Rev 1.0: *09.08.2013 First Draft*

Key features

- ***Compliant to IPMI 1.5***
- ***Compliant to PICMG 3.0***
- ***Compliant to HPM.1 firmware upgrade***
- ***Easy to integrate***
- ***Flexible ordering options:***
  - ***preprogrammed microcontroller***
  - ***source code***
- ***Evaluation Board Available***
- ***Reference Designs Provided***
- ***GUI software for configuration***
- ***Cost Effective:***
  - ***No upfront costs for standard version***
  - ***No royalties***
- ***Analog inputs for voltage, current or temperature measurements***
- ***External I2C for temperature measurements and communication with payload***
- ***Hot swap switch input, LEDs and payload power control outputs***

Table of Contents

Index of Tables

Description

The Intelligent Protocol Management Controller (IPMC) Software allows quick development of boards without prior IPMI knowledge. It provides all the mandatory IPMI functionality required by PICMG 3.0 (AdvancedTCA) specification.

The IPMC Software is a cost effective solution that enables very fast development of boards. There are no royalties and for the standard version there aren't any upfront costs. The standard version of the IPMC Software can be customized to fulfill any particular requirements.

The software is accompanied by reference schematics and a complete set of GUI compilers for the SDR and FRU files.

The IPMC Software is compliant to HPM.1 and thus can be easily upgraded over IPMB-0 using a Shelf Manager with no need for external cables or custom software.

The IPMC Software can implement discrete, temperature, voltage, current, fan or OEM sensors . The SDRs for the standard version of software implement only a basic set of sensors.

Samway's IPMC Software is easy to use and has flexible ordering options. It can be delivered on a preprogrammed micro controller or as source code. The microcontroller used is a Freescale Kinetis K10 hosted in a 64 pin LQFP package that needs 10 mm x 10 mm of PCB space.

FRU Information

The IPMC controller will be used in an IPMI environment. In order to interact to the other FRUs in the system, the board will have to host a FRU information file. This type of file contains important information concerning the board:

- Manufacturers name
- Part Number
- Serial Number
- Revision
- Manufacturing data
- Information related to the communication protocols implemented over the Base, Fabric, Timing and Local Bus interfaces
- other IPMI related information

All the required information can be saved in the FRU file format using the GUI FRU compiler.

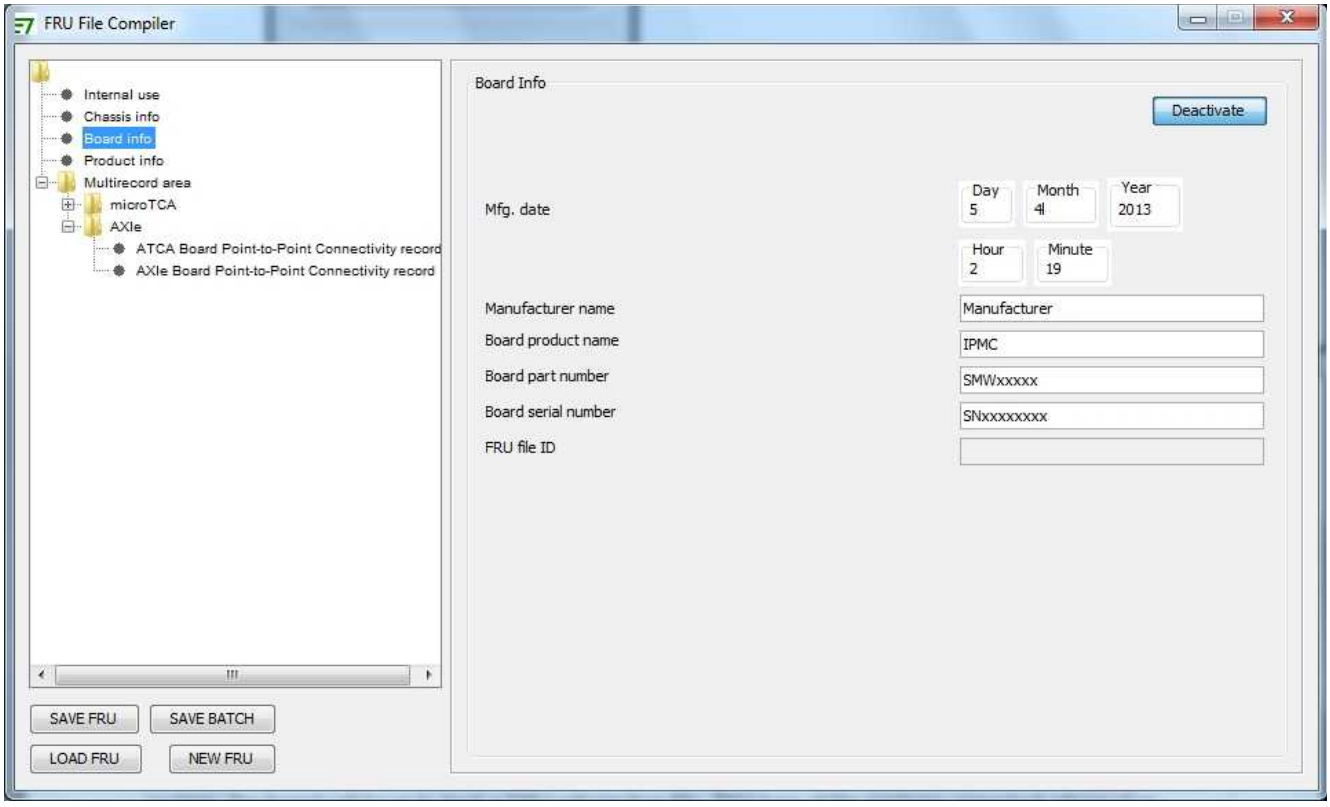

Figure 1: FRU File Compiler

Sensors

The IPMI Controller supports a limited, predefined, set of sensors. Each supported sensor has a predefined sensor number and if it is used,it shall have a Sensor Data Record (SDR) associated to it. Using the set of SDRs the IPMI software knows what sensors are implemented by the card and how to monitor them.

The supported sensor list is defined by the following table.

Sensor Number	Default Sensor Name	Description
1	**Hot Swap**	**Discrete sensor**: offers information about the hot swap state of the IPMC 0 = M0 – FRU Absent 1 = M1 - FRU Inactive 2 = M2 – FRU Activation Request 3 = M3 – FRU Activation In Progress 4 = M4 – FRU Active 5 = M5 – FRU Deactivation Request 6 = M6 – FRU Deactivation In Progress 7 = M7 – FRU Communication Lost
2	**Hot Swap Handle**	**Discrete sensor**: offers information about the hot swap handle: 0 = Handle Open 1 = Handle Closed
3	**IPMB0_Status**	**Discrete sensor**: offers information about the state of the IPMB0 bus: IPMB_A ok, error, enabled, disabled IPMB_B ok, error, enabled, disabled
4	**An0**[1]	**Analog Sensor:** can be used to monitor a voltage, current or temperature signal
5	**An1**	**Analog Sensor:** can be used to monitor a voltage, current or temperature signal
6	**An2**	**Analog Sensor:** can be used to monitor a voltage, current or temperature signal
7	**An3**	**Analog Sensor:** can be used to monitor a voltage, current or temperature signal
8	**An4**	**Analog Sensor:** can be used to monitor a voltage, current or temperature signal
9	**An5**	**Analog Sensor:** can be used to monitor a voltage, current or temperature signal
10	**An6**	**Analog Sensor:** can be used to monitor a voltage, current or temperature signal
11	**An7**	**Analog Sensor:** can be used to monitor a voltage, current or temperature signal

1 The analog input pins withstand signals in the range 0..2.5 Volts. If the signals that need to be monitored are outside this range, additional circuitry will be necessary (voltage dividers). For examples of circuits used in monitoring common voltages (3.3V, 12V, -12V, -48V) please refer to the reference schematic

Sensor Number	Default Sensor Name	Description
12	**Temp1**[2]	**Analog Sensor**: monitors a **TMP75** I2C temperature sensor configured for the I2C address **0x90**
13	**Temp2**	**Analog Sensor**: monitors a **TMP75** I2C temperature sensor configured for the I2C address **0x92**
14	**Temp3**	**Analog Sensor**: monitors a **TMP75** I2C temperature sensor configured for the I2C address **0x94**
15	**Temp4**	**Analog Sensor**: monitors a **TMP75** I2C temperature sensor configured for the I2C address **0x98**
16	**Max6699_Temp1**[3]	**Analog Sensor**: monitors a thermal diode connected to input 1 of the **Max6699** circuit
17	**Max6699_Temp2**	**Analog Sensor**: monitors a thermal diode connected to input 2 of the **Max6699** circuit
18	**Max6699_Temp3**	**Analog Sensor**: monitors a thermal diode connected to input 3 of the **Max6699** circuit
19	**Max6699_Temp4**	**Analog Sensor**: monitors a thermal diode connected to input 4 of the **Max6699** circuit
20	**Max6699_Local_Sensor**	**Analog Sensor**: monitors the board temperature sensed by the **Max6699** circuit
21	**FAN1**[4]	**Analog Sensor**: monitors Tachometer input 1
22	**FAN2**	**Analog Sensor**: monitors Tachometer input 2
23	**FAN3**	**Analog Sensor**: monitors Tachometer input 3
24	**FAN4**	**Analog Sensor**: monitors Tachometer input 4
25	**FAN5**	**Analog Sensor**: monitors Tachometer input 5
26	**FAN6**	**Analog Sensor**: monitors Tachometer input 6
27	**FAN7**	**Analog Sensor**: monitors Tachometer input 7
28	**FAN8**	**Analog Sensor**: monitors Tachometer input 8
32	**GPIO1**	**Digital General Pourpose I/O sensor**
33	**GPIO2**	**Digital General Pourpose I/O sensor**
34	**GPIO3**	**Digital General Pourpose I/O sensor**

Table 1: Supported Set of Sensors

2 In order to monitor temperature using sensors Temp1-Temp4 the related TMP75 sensors need to be installed

3 In order to monitor temperature using sensors Max6699_Temp1-Max6699_Local_Sensor the Max6699 circuit needs to be installed

4 In order to monitor fans using sensors FAN1-FAN8 a 8 channel multiplexer(74HC4851) and it's related circuitry have to be present

Configuring Sensors

The IPMC uses standard, IPMI compliant SDR records in order to monitor the board parameters.

The SDR repository of a board will be a software image of the hardware sensors. For each board there may be a different set-up as the requirements are different. So in order to allow a quick and simple set-up, a GUI SDR compiler is provided.

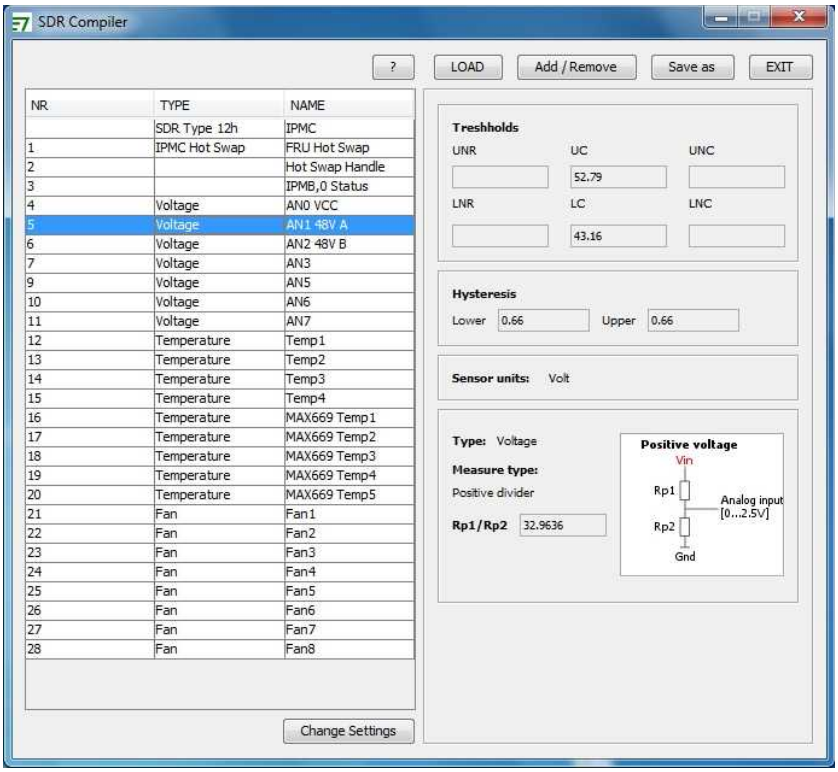

Figure 2: SDR Compiler

Using the GUI compiler a subset of all of the supported sensors can be defined by a simple select operation.

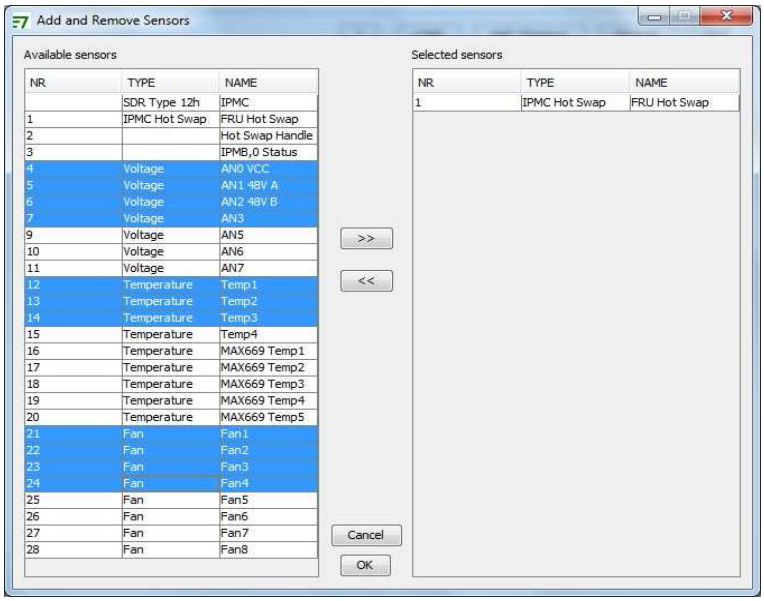

Figure 3: Selecting a subset of sensors

After the SDR set has been defined, all the sensors can be customized:

- threshold and hysteresis values can be changed for analog sensors
- names can be changed for all sensors

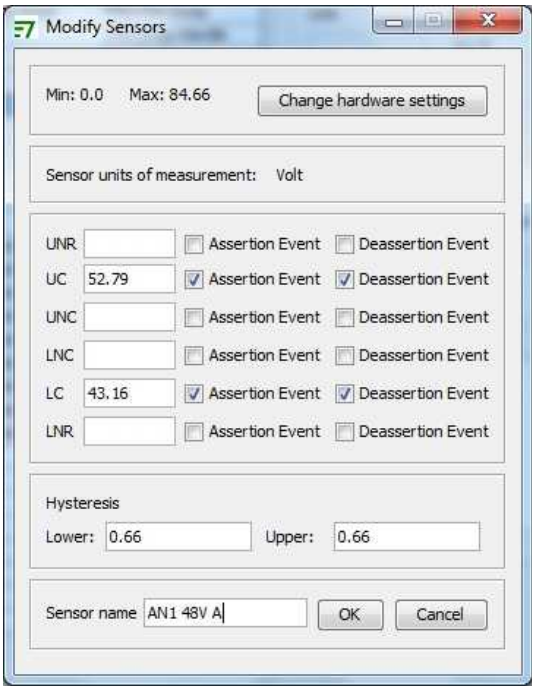

Figure 4: Window for changing parameters for a analog sensor

- for the analog inputs the raw SDR formulas of a few hardware circuits have been implemented. This feature allows easy integration of common analog set-ups: positive voltage divider, negative voltage divider, and gain block. For these common circuits only the divider/gain value has to be inputed, and the raw conversion formula will be automatically computed by the software. For more complex circuits, the raw formula can be inputed manually.

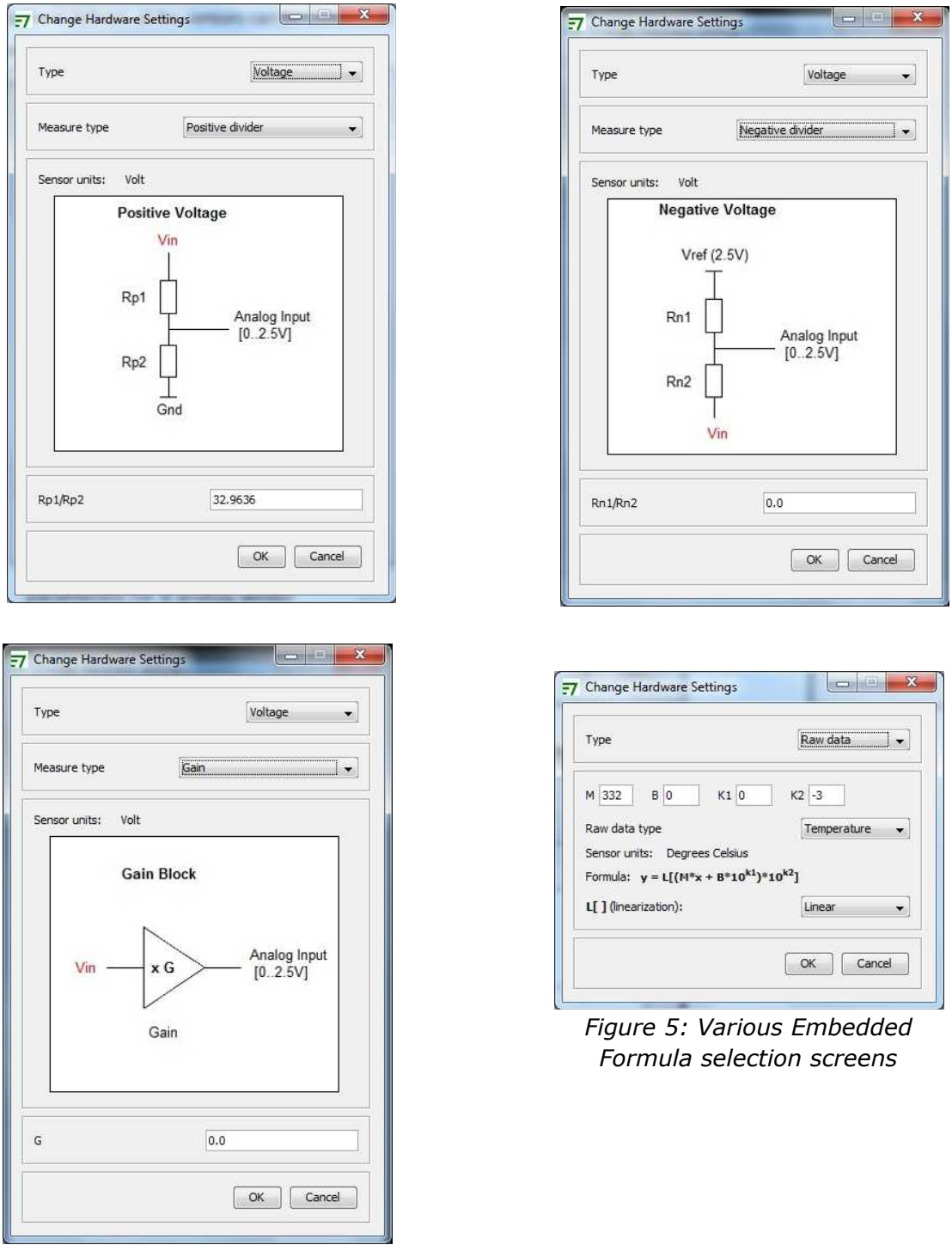

Figure 5: Various Embedded Formula selection screens

GPIO Sensors

The IPMC supports 3 General Purpose Input Output (GPIO) sensors. Each sensor is assigned to a micro-controller pin equipped with an open collector driver.

The GPIO sensors can be configured as:

- Active Low Input
- Active High Input
- Active Low Input – Output

For all types of GPIO sensors the pin state is checked at regular intervals and the sensor state is refreshed acourdingly. The GPIOs have been implemented using a discrete sensor having two states:

- 1 Asserted: physical pin is at the active level
- 0 Deasserted: physical pin is not at the active level

For input-output GPIO sensors,besides checking out the state of the micro-controller pin, the user can also control it using an override state.

The input - output GPIOs can only be used as active low signals, because the GPIO output driver is implemented using an open collector architecture.

Input-Output Overide states:

- 1 Asserted: physical pin is held low (GND) by the micro-controller
- 0 Deasserted: physical pin is a high impedance input, and can be controlled by other drivers.

To figure out who is keeping an input-output signal asserted, you can use the gpio command. For an input-output GPIO asserted by the micro-controller the override state is active and displayed accordingly by the command. Input-output GPIOs that are asserted by another driver do not have the override state active. In the following example GPIO2 is asserted by the micro-controller and GPIO3 is asserted by another driver:

```
%>gpio
----------------------GPIO List-----------------------------------------
GPIO  Sensor       Active
-no---No---Type----Level---Name-------------State-----------Override----
* 1    32    Input    Low      GPIO1             0 De-Asserted
* 2    33    Output   Low      GPIO2             1 Asserted      1 Asserted
* 3    34    Output   Low      GPIO3             1 Asserted
```

Selecting GPIO type

The IPMC supports 3 types of sensor for the GPIOs:

GPIO type	SDR Sensor Type
Active Low Input	0xC0
Active Low Input – Output	0xC1
Active High Input	0xC3

Table 2: GPIO sensor Type

The GPIO type can be configured using the SDR compiler. To do so you have to select the sensor from the list and use the ***Edit Sensor*** button.

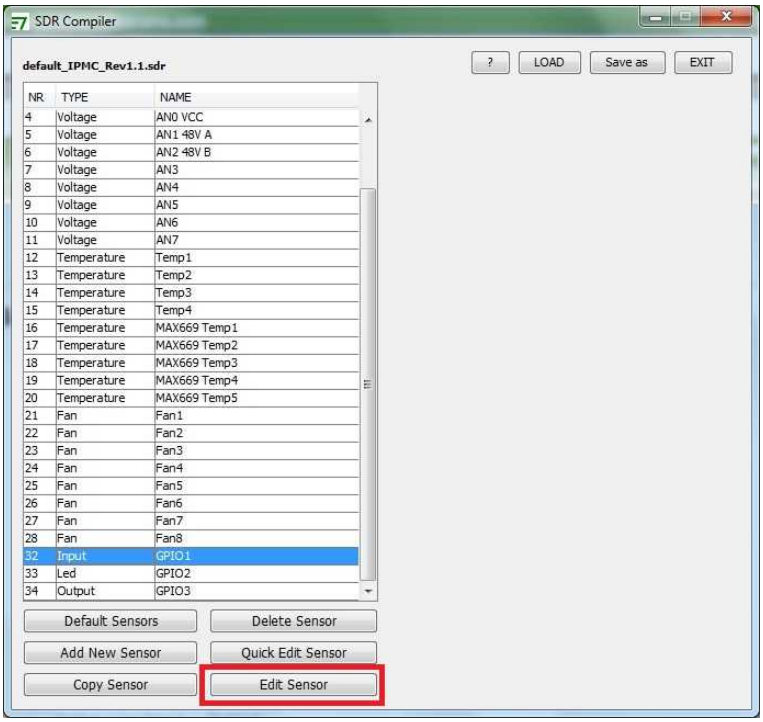

Figure 6: Change the GPIO type step 1

The next step is writing the SDR type necessary for the desired GPIO type, in the dedicated SDR field. The SDR type values are defined in Table 2.

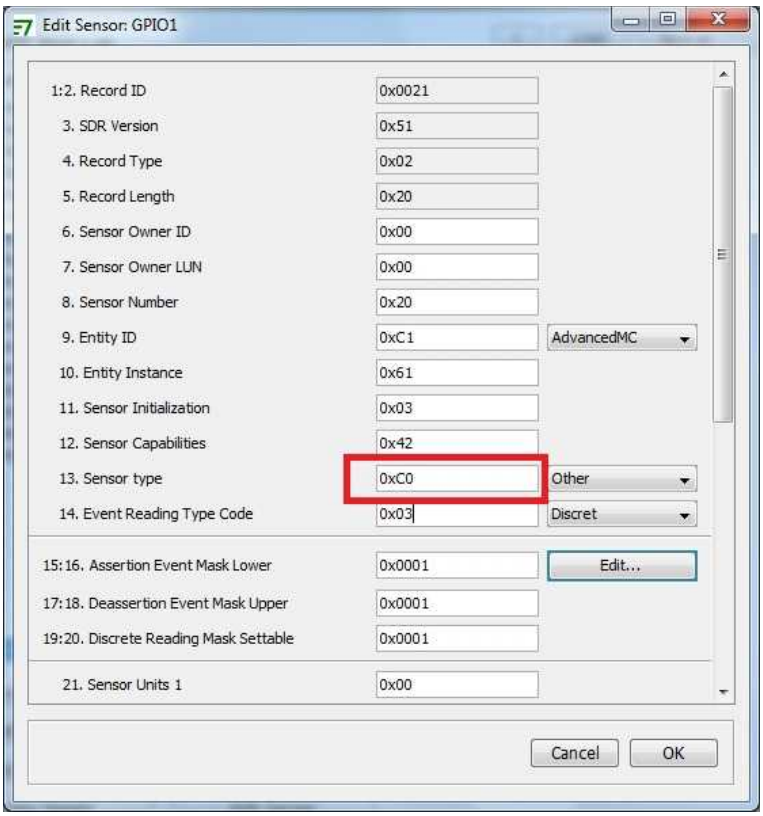

Figure 7: Changing GPIO type step 2

Controlling GPIOs using the CLI

The Command Line interface (CLI) is available over the RS232 interface. For controlling GPIOs a dedicated command has been implemented:

gpio [no as | de]

The parameters entered are GPIO number: 1..3, and desired action as – assert, de – deassert.

Example 1:

```
%>gpio 2 as
 Done! GPIO2 Asserted!
```

The command can also be used to check out the state of the GPIOs. For reading the state the command is entered without any parameters.

Example 2:

```
%>gpio
------GPIO List-------------------------------------------------
    Sensor
-no---No---Type----Name-------------State-----------Override----
* 1    32    Input    GPIO1              0 De-Asserted
* 2    33    Output   GPIO2              0 De-Asserted
* 3    34    Output   GPIO3              1 Asserted       1 Asserted
```

In the example above:

- GPIO1 is configured as input only, while GPIOs 2,and 3 are configured as input-output.

- GPIO3 is asserted locally (the local override is also asserted)

The state of the GPIOs can also be checked out using the sensor command:

```
%>sensor
---------------------Sensor List-----------------------------
--no--Name--------------Value---Unit---State-----------------

*    1  Hot Swap Handle  Closed
*    2  AN0 VCC          3.27     V      Ok
*    3  AN1 +12V         12.31    V      Ok
*    4  AN2              0.00     V      Ok
*    5  AN3              0.00     V      Ok
*    6  AN4              0.00     V      Ok
*    7  AN5              0.01     V      Ok
*    8  AN6              0.00     V      Ok
*    9  AN7              0.00     V      Ok
*   10  Temp1            26.00    deg C  Ok
*   11  Temp2            26.00    deg C  Ok
*   32  GPIO1            0 De-Asserted
*   33  GPIO2            0 De-Asserted
*   34  GPIO3            0 De-Asserted
```

The sensor command identifies the GPIOs using the sensor number and not the GPIO number. The correspondence between the two set of numbers is displayed by the gpio command.

Controlling GPIOs using IPMI commands

The GPIOs that have been configured as input-output pins can also be controlled over IPMB using the IPMI 2.0 ***Set Sensor Reading and Event Status*** command. In order for the IPMC to acknowledge the command and control the GPIOs the following values have to be used for the parameters:

Field		Description	Value
Responder's Slave Addr.(**RsSA**)		IPMB address of the IPMC	It depends on the physical slot in which the board is inserted: 0x82 – slot 1,0x84 – slot 2 ...
NetFn/RsLUN		Sensor/Event request + LUN = 0	0x10
Command		***Set Sensor Reading and Event Status***	0x30
Request Data	Byte 1	Sensor Number	GPIO sensor number displayed by the ***sensor*** CLI command
	Byte 2	Operation	0x10
	Byte 3	Sensor reading- not used	0x00
	Byte 4	GPIO state	0x00 - deassert 0x01 – assert

Table 3: Parameters for Set Sensor Reading and Event Status command for controlling GPIOs

Example 1: turn GPIO1 (sensor no. 32 = 0x20) on

```
Request from ShMC 0x88 0x10 0x68 0x20 0xB8 0x30 0x20 0x10 0x00 0x01 0xC7
Response from AMC 0x20 0x14 0xCC 0x88 0xB8 0x30 0x00 0xA2
```

E-keying

For E-Keing the IPMC software implements 2 mechanisms:

- Forwarding the data received from the Shelf Manager via the “Set Port state” command(The command data is forwarded to a user defined I2C address)
- 2 user configurable 16 bit IO-Expanders. The IO-Expanders allow up to 31 user configurable pins that can act as active high outputs and can be either asserted or deasserted for either enabling or disabling the link.

Either one, both or none of the above options can be used for implementing E-keying.
In order to allow a simple setup the GUI FRU Compiler implements Samway's OEM Ekey Add-On record.

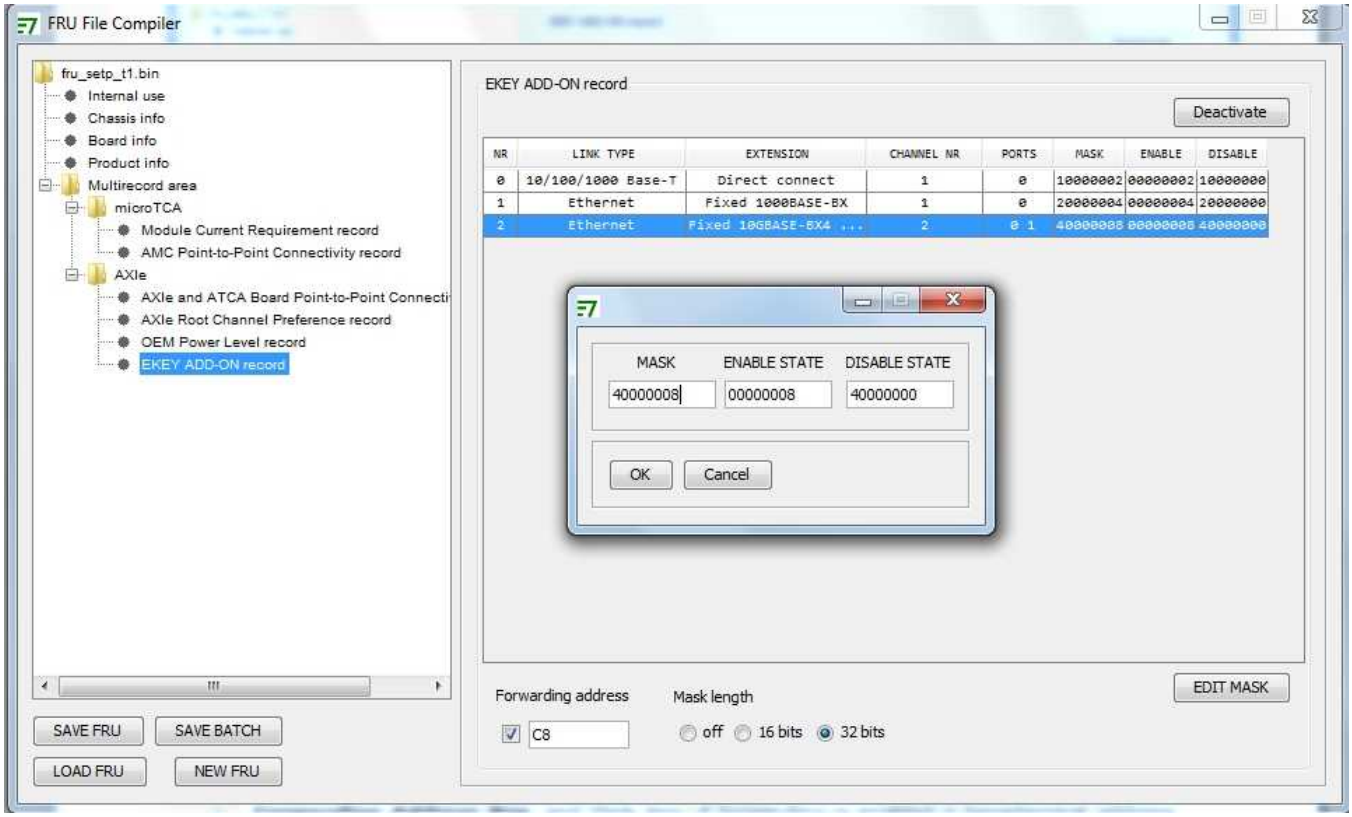

Figure 8:Samway OEM Ekey Add-On Record

For every link the 32 IO pins behavior is defined by a set of tree hexadecimal values:

- **Used Pins Mask**: the mask defines the pins that will be affected by a state change for the link. If a bit's value is 1, the corresponding pin is used by the current link.

- **Enable Link State:** the mask defines the values for the used bits when the link is enabled. If a bit's value is 1 the pin will be asserted, otherwise it will be deasserted.
- **Disable Link State:** the mask defines the values for the used bits when the link is disabled. If a bit's value is 1 the pin will be asserted, otherwise it will be deasserted.

The tree masks are hexadecimal values that represent the 32 IO Expander pins: Ekey0 ..Ekey31. Ekey0 is reserved for the MASK_VALID signal and is not available to the user. All the other pins can be used freely.

For the values in Figure 6:

- **Used Pins Mask**: for the current link, pins Ekey3 and Ekey30 are used
- **Enable Link State**: when the link is enabled Ekey3 is asserted and Ekey30 is deasserted
- **Disable Link State**: when the link is disabled Ekey3 is deasserted and Ekey30 is asserted

IPMC E-keying procedure:

1. The IPMC receives the "Set Port State" command from the shelf manager
2. If forwarding is enabled and a non-zero I2C address has been defined, the command data is forwarded on the external I2C bus using the following format:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4..7	Byte 8
I2C Address	0x01 **(FWD command)**	0x00	**Link Type**	**Link Descriptor**	**State**

Link Type : 0x02 ATCA link
all other values are reserved

Link Descriptor: 4 byte value per ATCA received from the Shelf Manager

State : 0x00 Disable
0x01 Enable
all other values are reserved

3. If the IO Expanders are used and the tree masks have been defined for the current link, the MASK_VALID signal is deasserted.
4. The values for the pins defined by the **Used Pins Mask** are changed. If the link is enabled the pin values are defined by the **Enable Link State** mask, and if the link is disabled the values are changed according to the **Disable Link State** mask.
5. After the pin values have been changed, the MASK_VALID signal is asserted.
6. The IPMC sends the completion code to the shelf manager for the E-keying command.

Payload available signals

Signal Name	Type	Active level	Description
PP_RST	Output	High	Payload reset signal
SHDN_RDY#	Input	Low	When the signal is active the payload is ready to be shutdown
Payload_SCL	In-Out		I2C serial clock
Payload_SDA	In-Out		I2C serial data
PCIe_Ready#	Input	Low	When the signal is active the payload has finished the PCIe initialization
SLOT_ID[4:1][5]	IO Expander Output		4 signals that represent the slot number for the current board position in the system
Temp_Fail#	IO Expander Output	Low	If any of the temperature sensor violates any of their thresholds, this signal is asserted
SHDN_REQ#	IO Expander Output	Low	When active, signals the payload that a shutdown has been requested
PP_EN#	IO Expander Output	Low	Enable for the Payload Power switch
LED2	In-Out		Green Led that can be controlled both by the payload and the IPMC
LED3	In-out		Amber Led that can be controlled both by the payload and the IPMC

Table 4: Payload available Signals

Leds:

Name	Color	Control	Description
Blue Led	Blue	IPMC	Hot swap led
Led 1	Green,Red, Amber	IPMC	Status Led: Green – ok Red - error
Led 2	Green	IPMC, Payload	User defined
Led 3	Amber	IPMC, Payload	User defined

Table 5: Available Leds

5 For IO Expander signals a PCA9532 chip configured for address 0x96 has to be assembled

Command Line Interface (CLI)

The IPMC provides a RS232 serial interface through which the commands of the Command Line Interface (CLI) can be sent.

On Windows systems, we recommend the use of “Tera Term” or “Hyperterminal” as the terminal programs.

Terminal settings:
- 115200 bits per second
- data bits: 8
- parity: none
- stop bit: 1

For file transfer the CLI implements the xmodem protocol.

List of CLI commands

gpio command

Syntax: gpio [no as | de]
Function: Displays or changes the state of the GPIOs.
No = 1..1

Example 1:

```
%>gpio
----------------------GPIO List-----------------------------------------
GPIO  Sensor       Active
-no---No---Type----Level---Name------------State-----------Override----

* 1    32    Input    Low      GPIO1             0 De-Asserted
* 2    33    Input    High     GPIO2             1 Asserted
* 3    34    Output   Low      GPIO3             1 Asserted      1 Asserted
```

Example 2:

```
%>gpio 3 de
 Done! GPIO3 De-Asserted!
```

help command

Syntax: help
Function: Displays a list of the available commands.

payload_reset

Syntax: payload_reset [Active_time_10_ms]
Function: Asserts the payload reset signal , keeps it active for the time value entered as a parameter and then de-assert it.

payload_signals

Syntax: payload_signals
Function: Displays the status for the payload signals.
Example:

```
%>payload_signals
 Payload Signals status:

 TEMP_FAIL#: De-Asserted
 SHDN_REQ#: De-Asserted
 SHDN_RDY#: De-Asserted
 PCIE_READY#: Asserted
 PP_EN#: Asserted
 PP_RST#: De-Asserted
 PCIE_ENUM_EN#: De-Asserted
```

reboot command

Syntax: reboot
Function: Restarts the IPMC
Example 1:

```
%>reboot
 System will restart! Please wait...
```

sensor command

Syntax: sensor
Function: Displays information for the installed set of sensors.
Example 1:

```
%>sensor
---------------------Sensor List-----------------------------
--no--Name--------------Value--Unit---State-----------------
*   1  FRU Hot Swap     M0: FRU Not Installed
*   2  Hot Swap Handle
*   3  IPMB-0 Status    IPMB A: ok   , Enabled
                        IPMB B: ok   , Enabled
*   4  AN0 VCC          3.31   V      Ok
*   5  AN1 48V A        47.14  V      Ok
*   6  AN2 48V B        47.48  V      Ok
*   7  AN3              0.01   V      Ok
*   8  AN4              0.01   V      Ok
*   9  AN5              0.00   V      Ok
*  10  AN6              0.01   V      Ok
*  11  AN7              0.01   V      Ok
*  12  Temp1            28.00  deg C  Ok
*  13  Temp2            28.00  deg C  Ok
*  14  Temp3            27.00  deg C  Ok
*  15  Temp4            28.00  deg C  Ok
*  16  MAX669 Temp1     28.00  deg C  Ok
*  17  MAX669 Temp2     28.00  deg C  Ok
*  18  MAX669 Temp3     27.00  deg C  Ok
*  19  MAX669 Temp4     28.00  deg C  Ok
```

```
*  20  MAX669 Temp5     28.00  deg C  Ok
*  21  Fan1             0      RPM    LC
```

uptime command

Syntax: uptime
Function: Displays the amount of time which has past since the IPMC became operational.
Example 1:

```
%>uptime
 Uptime=0 days 03:05:12
```

version command

Syntax: version
Function: Displays various information about the IPMC: firmware version, Hardware Id, Slot ID.
Example1:

```
%>version
 IPMC FW 1.1
 Hardware Id : 7
 Slot : 3
```

xmodem command

Syntax: xmodem fru | sdr
Function: Upload the FRU or SDR file to the IPMC using the xmodem protocol
Example 1:

```
%>xmodem fru
 Please upload the file...
%>...Done!
```

Updating the Firmware

The Firmware of the IPMC can be updated using the on-board bootloader.
For uploading a file the following steps are required :

- Connect to the CLI interface(**Terminal settings:** 115200 bits per second, data bits: 8, parity: none, stop bit: 1)
- Stop the bootloader by pressing 'x'
- Issue the **xmodem** firmware command

  %> xmodem firmware
- Upload the **.firm* file using the terminal program
- After the file transfer is completed the firmware will be updated.

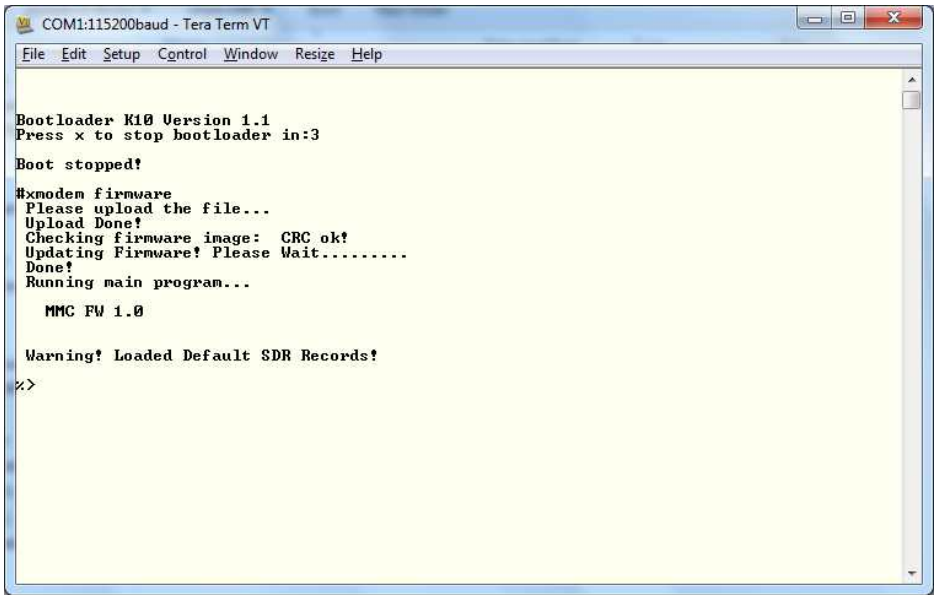

Figure 9: Firmware update example

Updating the FRU and SDR files

In order to configure the IPMC two files are required: the FRU and SDR file. Both can be easily created using the GUI software suites that accompany the IPMC: FRU File compiler and SDR File compiler.

Creating new files or modifying old ones is really straight forward due to the graphical interface. For more details on all the available options please refer to the respective software user manuals.

After the files are created they have to be uploaded using the CLI.

For uploading a file the following steps are required :

1. Connect to the CLI interface
2. Issue the **xmodem** command, using the correct parameter:

%> xmodem fru | sdr

3. Upload the file using the terminal program
4. After the file transfer is completed a confirmation message will be displayed. At this point the file has been saved and a reboot is required in order to activate the changes.

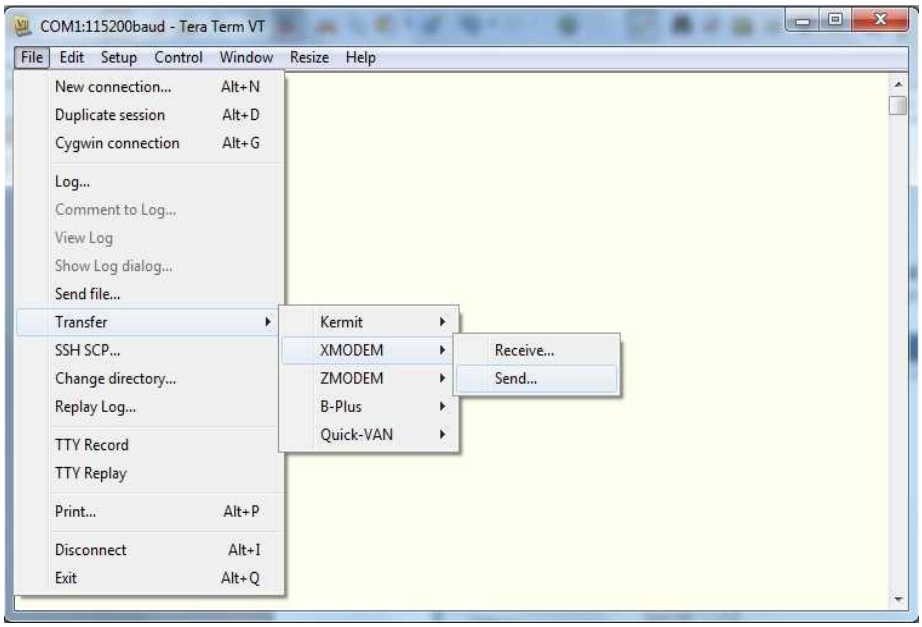

Figure 10: Tera Term Screen shot for sending a file using xmodem

Additional auxiliary circuits:

External Watchdog

An external watchdog could be used as a security measure, to prevent software freezing. The watchdog is strobed by the IPMC software. If the software fails to strobe the watchdog before it's timeout expires, the IPMC will be reset. The reset will be transparent to the payload and will only affect the IPMC.

IPMB0 Buffers

For connecting to the IPMB A and IPMB B buses two I2C buffers are required.

Voltage Reference

If analog sensors are implemented a 2.5 Volts voltage reference needs to be provided. The monitored signals should be above 0V and bellow 2.5V.

Temperature sensors

In order to monitor the temperature of the board, external temperature sensors will have to be assembled. The IPMC supports up to four TMP75 I2C temperature sensors and a MAX6699 device that can be used to monitor thermal diode sensors embedded in FPGAs or other complex ICs.

Tachometer sensors

The IPMC can monitor up to 8 tachometer signal for measuring fans speeds

IO Expanders

Some of the signals used for interfacing the payload are not hosted by the microcontroller: SLOT_ID[4:1],Temp_Fail#, SHDN_REQ#,PP_EN#. If any of these signals are required a PCA9532 IO expander configured for I2C address 0xC6 has to be assembled and connected to the local bus.

The IPMC is compliant to PICMG 3.0 and will respond the the Set Port State commands received during the Ekey-ing process. If the payload requires ekey-ing the commands can be reduced to 32 distinct digital enable signals. In order to provide the 32 enable signals two more additional PCA9532 IO expanders are required. The expanders will be configured for addresses 0xC0 and 0xC2 and will be connected to the local bus.

EEPROM (24LC512 Type)

A serial I2C EEPROM can be connected to the local I2C bus of the micro-controller. The EEPROM is not mandatory, but if space is available, it's footprint can be placed in the layout in order to leave room for future development.

For more details on the auxiliary components and the way they interact to the IPMC and the payload please refer to the reference schematic.

Order codes

P07037-A – Preprogrammed microcontroller with IPMC software
P07033-A – IPMC software evaluation board