Advanced Platform Managment Link (APML) Library

Generated by Doxygen 1.8.17

Chapter 1

Advanced Platform Management Link (APML) Library

1.1 (formerly known as EPYC™ System Management Interface (E-SMI) Out-of-band Library)

The Advanced Platform Management Link (APML) Library library, is part of the EPYC™ System Management Out-of-band software stack. It is a C library for Linux that provides a user space interface to monitor and control the CPU's Systems Management features.

1.2 Important note about Versioning and Backward Compatibility

The APML library is currently under development, and therefore subject to change at the API level. The intention is to keep the API as stable as possible while in development, but in some cases we may need to break backwards compatibility in order to achieve future stability and usability. Following Semantic Versioning rules, while the APML library is in a high state of change, the major version will remain 0, and achieving backward compatibility may not be possible.

Once new development has leveled off, the major version will become greater than 0, and backward compatibility will be enforced between major versions.

1.3 Building APML Library

1.3.0.0.1 Additional Required software for building In order to build the APML library, the following components are required. Note that the software versions listed are what is being used in development. Earlier versions are not guaranteed to work:

- CMake (v3.5.0)
- latex (pdfTeX 3.14159265-2.6-1.40.18)
- apml modules (apml_sbrmi and apml_sbtsi)
  - available at `https://github.com/amd/apml_modules/`

1.3.0.0.2 Dowloading the source The source code for APML library is available on `Github`.

1.3.0.0.3 Directory stucture of the source Once the APML library source has been cloned to a local Linux machine, the directory structure of source is as below:

- `$ docs/` Contains Doxygen configuration files and Library descriptions
- `$ tool/` Contains apml_tool based on the APML library
- `$ include/esmi_oob` Contains the header files used by the APML library
- `$ src/esmi_oob` Contains library APML source

1.3.0.0.4 Building the library is achieved by following the typical CMake build sequence for native build, as follows.

```
$ mkdir -p build
```

```
$ mkdir -p install
```

```
$ cd build
```

```
$ cmake -DCMAKE_INSTALL_PREFIX=${PWD}/install <location of root of APML
library CMakeLists.txt>
```

`$ make` The built library will appear in the `build` folder.

1.3.0.0.5 Cross compile the library for Target systems Before installing the cross compiler verfiy the target architecture

`$ uname -m` Eg: To cross compile for ARM32 processor:

`$ sudo apt-get install gcc-arm-linux-gnueabihf` Eg: To cross compile for AARCH64 processor: use

`$ sudo apt-get install gcc-aarch64-linux-gnu` NOTE: For cross compilation, cross-$ARC↩
H.cmake file is provided for below Architectures:

- armhf
- aarch64

Compilation steps

$ mkdir -p build

$ cd build

$ cmake -DCMAKE_TOOLCHAIN_FILE=../cross-[arch..].cmake <location of root of APML library CMakeLists.txt>

$ make The built library will appear in the `build` folder. Copy the required binaries and the dynamic linked library to target board(BMC).

$ scp libapml64.so.0 root@10.x.x.x:/usr/lib

$ scp apml_tool root@10.x.x.x:/usr/bin

1.3.0.0.6 Disclaimer

- Input arguments passed by the user are not validated. It might result in unreliable system behavior

1.3.0.0.7 Building the Documentation The documentation PDF file can be built with the following steps (continued from the steps above):

$ make doc The reference manual (APML_Library_Manual.pdf), release notes (APML_Library_Release_↩ Notes.pdf) upon a successful build.

1.4 Usage Basics

Most of the APIs need socket index as the first argument. Refer tools/apml_tool.c

1.5 Usage

1.5.1 Tool Usage

APML tool is a C program based on the APML Library, the executable "apml_tool" will be generated in the build/ folder. This tool provides options to monitor and control System Management functionality.

In execution platform, user can cross-verfiy "apml_sbrmi" and apml_rmi" modules are loaded. The apml modules are open-sourced at https://github.com/amd/apml_modules.git

For detailed usage information, use -h or –help flag:

```
$ ./apml_tool -h
================================ APML System Management Interface ===================================
Usage: ./apml_tool [soc_num] [Option<s> / [--help] [module-name]
Where:  soc_num : socket number 0 or 1
Description:
```

```
./apml_tool -v                        - Displays tool version
./apml_tool [SOC_NUM] --showdependency  - Displays module dependency
./apml_tool --help <MODULE>           - Displays help on the options for the specified module
./apml_tool <option/s>                - Runs the specified option/s.
Usage: ./apml_tool [soc_num] [Option] params
        MODULES:
        1. mailbox
        2. sbrmi
        3. sbtsi
        4. reg-access
        5. cpuid
        6. recovery
========================================== End of APML SMI ============================================
$ ./apml_tool -v
```

================================ APML System Management Interface ====================================

APML_tool version : X.Y.Z

== End of APML SMI ==

```
Below is a sample usage to get the individual library functionality API's over I2C.
User can pass arguments either any of the ways "./apml_tool [socket_num] -p" or "./apml_tool [socket_num]
      --showpower"


1. $ ./apml_tool 0 -p

    ================================== APML System Management Interface ===================================

    ---------------------------------------------
    | Power (Watts)           | 65.029           |
    | PowerLimit (Watts)      | 210.000          |
    | PowerLimitMax (Watts)   | 400.000          |
    ---------------------------------------------

    ========================================== End of APML SMI ============================================

2. $ ./apml_tool 1 --setpowerlimit 200000

    ================================== APML System Management Interface ===================================


    Set power_limit :          200.000 Watts successfully

    ========================================== End of APML SMI ============================================

3. $ ./apml_tool 0 --showtsiregisters

    ================================== APML System Management Interface ===================================
    ---------------------------------------------------------------------------------


                 *** SB-TSI REGISTER SUMMARY ***
    ----------------------------------------------------------------------------------
     FUNCTION/Reg Name      | Reg offset     | Hexa(0x)       | Value [Units]
    ------------------------------------------------------------------------------------------
    _PROCTEMP               |                |                | 55.125 °C
           PROC_INT         | 0x1            | 0x37           | 55 °C
           PROC_DEC         | 0x10           | 0x1            | 0.125 °C
    _STATUS                 | 0x2            |                |
           PROC Temp Alert  |                |                | PROC No Temp Alert
           Mem Temp Alert   |                |                | HBM High Temp Alert
    _CONFIG                 | 0x3            |                |
           ALERT_L pin      |                |                | Enabled
           Runstop          |                |                | Comparison Enabled
           Atomic Rd order  |                |                | Integer latches Decimal
    _TSI_UPDATERATE         | 0x4            |                | 16.000 Hz
    _HIGH_THRESHOLD_TEMP    |                |                | 70.000 °C
           HIGH_INT         | 0x7            | 0x46           | 70 °C
           HIGH_DEC         | 0x13           | 0x0            | 0.000 °C
    _LOW_THRESHOLD_TEMP     |                |                | 0.000 °C
           LOW_INT          | 0x8            | 0x0            | 0 °C
           LOW_DEC          | 0x14           | 0x0            | 0.000 °C
    _HBM_HIGH_THRESHOLD_TEMP|                |                | 0.000 °C
           HIGH_INT         | 0x40           | 0x0            | 0 °C
```

```
        HIGH_DEC        | 0x44          | 0x0           | 0.000 °C
_HBM_LOW_THRESHOLD_TEMP |               |               | 0.000 °C
        LOW_INT         | 0x48          | 0x0           | 0 °C
        LOW_DEC         | 0x4c          | 0x0           | 0.000 °C
_HBM_MAX_TEMP           |               |               | 49.000 °C
        MAX_INT         | 0x50          | 0x31          | 49 °C
        MAX_DEC         | 0x54          | 0x0           | 0.000 °C
_HBM_TEMP               |               |               | 47.000 °C
        HBM_INT         | 0x5c          | 0x2f          | 47 °C
        HBM_DEC         | 0x60          | 0x0           | 0.000 °C
_TEMP_OFFSET            |               |               | 0.000 °C
        OFF_INT         | 0x11          | 0x0           | 0 °C
        OFF_DEC         | 0x12          | 0x0           | 0.000 °C
_THRESHOLD_SAMPLE       | 0x32          |               |
        PROC Alert TH   |               |               | 1
        HBM Alert TH    |               |               | 1
_TSI_ALERT_CONFIG       | 0xbf          |               |
        PROC Alert CFG  |               |               | Enabled
        HBM Alert CFG   |               |               | Disabled
_TSI_MANUFACTURE_ID     | 0xfe          |               | 0
_TSI_REVISION           | 0xff          |               | 0x4
-----------------------------------------------------------------------------------

========================================== End of APML SMI ============================================
```

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Module Documentation

5.1 Auxiliary functions

Below functions provide interfaces to get the total number of cores, sockets and threads per core in the system.

Functions

- oob_status_t errno_to_oob_status (int err)

  convert linux error to esmi error.
- char ∗ esmi_get_err_msg (oob_status_t oob_err)

  Get the error string message for esmi oob errors.

5.1.1 Detailed Description

Below functions provide interfaces to get the total number of cores, sockets and threads per core in the system.

5.1.2 Function Documentation

5.1.2.1 errno_to_oob_status()

```
oob_status_t errno_to_oob_status (
            int err )
```

convert linux error to esmi error.

Get the appropriate esmi error for linux error.

Parameters

in	*err*	a linux error number

Return values

oob_↩ status_t	is returned upon particular esmi error

5.1.2.2 esmi_get_err_msg()

```
char* esmi_get_err_msg (
            oob_status_t oob_err )
```

Get the error string message for esmi oob errors.

Get the error message for the esmi oob error numbers

Parameters

in	*oob_err*	is a esmi oob error number

Return values

*char**	value returned upon successful call.

5.2 SB-RMI Mailbox Service

write, 'write and read' operations for a given socket.

Modules

- Power Monitor

  Below functions provide interfaces to get the current power usage and Power Limits for a given socket.
- Power Control

  This function provides a way to control Power Limit.
- Performance (Boost limit) Monitor

  This function provides the current boostlimit value for a given core.
- Out-of-band Performance (Boost limit) Control

  Below functions provide ways to control the Out-of-band Boost limit values.
- Current, Min, Max TDP

  Below functions provide interfaces to get the current, Min and Max TDP, Prochot and Prochot Residency for a given socket.
- Prochot

  Below functions provide interfaces to get Prochot and Prochot Residency for a given socket.
- Dram and other features Query

5.2.1 Detailed Description

write, 'write and read' operations for a given socket.

5.3 Power Monitor

Below functions provide interfaces to get the current power usage and Power Limits for a given socket.

Functions

- oob_status_t read_socket_power (uint8_t soc_num, uint32_t ∗buffer)

  Get the power consumption of the socket.
- oob_status_t read_socket_power_limit (uint8_t soc_num, uint32_t ∗buffer)

  Get the current power cap/limit value for a given socket.
- oob_status_t read_max_socket_power_limit (uint8_t soc_num, uint32_t ∗buffer)

  Get the maximum value that can be assigned as a power cap/limit for a given socket.

5.3.1 Detailed Description

Below functions provide interfaces to get the current power usage and Power Limits for a given socket.

5.3.2 Function Documentation

5.3.2.1 read_socket_power()

```
oob_status_t read_socket_power (
            uint8_t soc_num,
            uint32_t ∗ buffer )
```

Get the power consumption of the socket.

Given socket number and a pointer to a uint32_t `buffer`, this function will get the current power consumption (in watts) to the uint32_t pointed to by `buffer`.

Parameters

`in`	*soc_num*	Socket index.
`in,out`	*buffer*	a pointer to uint32_t value of power consumption

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.3.2.2 read_socket_power_limit()

```
oob_status_t read_socket_power_limit (
            uint8_t soc_num,
            uint32_t * buffer )
```

Get the current power cap/limit value for a given socket.

This function will return the valid power cap `buffer` for a given socket, this value will be used for the system to limit the power.

Parameters

`in`	*soc_num*	Socket index.
`in,out`	*buffer*	a pointer to a uint32_t that indicates the valid possible power cap/limit, in watts

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.3.2.3 read_max_socket_power_limit()

```
oob_status_t read_max_socket_power_limit (
            uint8_t soc_num,
            uint32_t * buffer )
```

Get the maximum value that can be assigned as a power cap/limit for a given socket.

This function will return the maximum possible valid power cap/limit

Parameters

`in`	*soc_num*	Socket index.
`out`	*buffer*	a pointer to a uint32_t that indicates the maximum possible power cap/limit, in watts

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.4 Power Control

This function provides a way to control Power Limit.

Functions

- oob_status_t write_socket_power_limit (uint8_t soc_num, uint32_t limit)

  Set the power cap/limit value for a given socket.

5.4.1 Detailed Description

This function provides a way to control Power Limit.

5.4.2 Function Documentation

5.4.2.1 write_socket_power_limit()

```
oob_status_t write_socket_power_limit (
            uint8_t soc_num,
            uint32_t limit )
```

Set the power cap/limit value for a given socket.

This function will set the power cap/limit

Parameters

in	*soc_num*	Socket index.
in	*limit*	uint32_t that indicates the desired power cap/limit, in milliwatts

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.5 Performance (Boost limit) Monitor

This function provides the current boostlimit value for a given core.

Functions

- oob_status_t read_esb_boost_limit (uint8_t soc_num, uint32_t value, uint32_t ∗buffer)

  Get the Out-of-band boostlimit value for a given core.
- oob_status_t read_bios_boost_fmax (uint8_t soc_num, uint32_t value, uint32_t ∗buffer)

  Get the In-band maximum boostlimit value for a given core.

5.5.1 Detailed Description

This function provides the current boostlimit value for a given core.

5.5.2 Function Documentation

5.5.2.1 read_esb_boost_limit()

```
oob_status_t read_esb_boost_limit (
             uint8_t soc_num,
             uint32_t value,
             uint32_t ∗ buffer )
```

Get the Out-of-band boostlimit value for a given core.

This function will return the core's current Out-of-band boost limit `buffer` for a particular `value`

Parameters

in	*soc_num*	Socket index.
in	*value*	a cpu index
in,out	*buffer*	pointer to a uint32_t that indicates the possible boost limit value

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.5.2.2 read_bios_boost_fmax()

```
oob_status_t read_bios_boost_fmax (
            uint8_t soc_num,
            uint32_t value,
            uint32_t * buffer )
```

Get the In-band maximum boostlimit value for a given core.

This function will return the core's current maximum In-band boost limit `buffer` for a particular `value` is cpu_ind

Parameters

`in`	*soc_num*	Socket index.
`in`	*value*	is a cpu index
`in,out`	*buffer*	a pointer to a uint32_t that indicates the maximum boost limit value set via In-band

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.6 Out-of-band Performance (Boost limit) Control

Below functions provide ways to control the Out-of-band Boost limit values.

Functions

- oob_status_t write_esb_boost_limit (uint8_t soc_num, uint32_t cpu_ind, uint32_t limit)

  Set the Out-of-band boostlimit value for a given core.
- oob_status_t write_esb_boost_limit_allcores (uint8_t soc_num, uint32_t limit)

  Set the boostlimit value for the whole socket (whole system).

5.6.1 Detailed Description

Below functions provide ways to control the Out-of-band Boost limit values.

5.6.2 Function Documentation

5.6.2.1 write_esb_boost_limit()

```
oob_status_t write_esb_boost_limit (
             uint8_t soc_num,
             uint32_t cpu_ind,
             uint32_t limit )
```

Set the Out-of-band boostlimit value for a given core.

This function will set the boostlimit to the provided value `limit` for a given cpu. NOTE: Currently the limit is setting for all the cores instead of a particular cpu. Testing in Progress.

Parameters

in	*soc_num*	Socket index.
in	*cpu_ind*	a cpu index is a given core to set the boostlimit
in	*limit*	a uint32_t that indicates the desired Out-of-band boostlimit value of a given core

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.6.2.2 write_esb_boost_limit_allcores()

```
oob_status_t write_esb_boost_limit_allcores (
             uint8_t soc_num,
             uint32_t limit )
```

Set the boostlimit value for the whole socket (whole system).

This function will set the boostlimit to the provided value `boostlimit` for the socket.

Parameters

in	*soc_num*	Socket index.
in	*limit*	a uint32_t that indicates the desired boostlimit value of the socket

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.7 Current, Min, Max TDP

Below functions provide interfaces to get the current, Min and Max TDP, Prochot and Prochot Residency for a given socket.

Functions

- oob_status_t read_tdp (uint8_t soc_num, uint32_t ∗buffer)

  Get the Thermal Design Power limit TDP of the socket with provided socket index.
- oob_status_t read_max_tdp (uint8_t soc_num, uint32_t ∗buffer)

  Get the Maximum Thermal Design Power limit TDP of the socket with provided socket index.
- oob_status_t read_min_tdp (uint8_t soc_num, uint32_t ∗buffer)

  Get the Minimum Thermal Design Power limit TDP of the socket.

5.7.1 Detailed Description

Below functions provide interfaces to get the current, Min and Max TDP, Prochot and Prochot Residency for a given socket.

5.7.2 Function Documentation

5.7.2.1 read_tdp()

```
oob_status_t read_tdp (
            uint8_t soc_num,
            uint32_t ∗ buffer )
```

Get the Thermal Design Power limit TDP of the socket with provided socket index.

Given a socket and a pointer to a uint32_t `buffer`, this function will get the current TDP (in milliwatts)

Parameters

in	*soc_num*	Socket index.
in,out	*buffer*	a pointer to uint32_t to which the Current TDP value will be copied

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.7.2.2 read_max_tdp()

```
oob_status_t read_max_tdp (
            uint8_t soc_num,
            uint32_t * buffer )
```

Get the Maximum Thermal Design Power limit TDP of the socket with provided socket index.

Given a socket and a pointer, this function will get the Maximum TDP (watts)

Parameters

in	*soc_num*	Socket index.
in,out	*buffer*	a pointer to uint32_t to which the Maximum TDP value will be copied

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.7.2.3 read_min_tdp()

```
oob_status_t read_min_tdp (
            uint8_t soc_num,
            uint32_t * buffer )
```

Get the Minimum Thermal Design Power limit TDP of the socket.

Given a socket and a pointer to a uint32_t, this function will get the Minimum TDP (watts)

Parameters

in	*soc_num*	Socket index.
in,out	*buffer*	a pointer to uint32_t to which the Minimum TDP value will be copied

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.8 Prochot

Below functions provide interfaces to get Prochot and Prochot Residency for a given socket.

Functions

- oob_status_t read_prochot_status (uint8_t soc_num, uint32_t ∗buffer)

  Get the Prochot Status of the socket with provided socket index.
- oob_status_t read_prochot_residency (uint8_t soc_num, float ∗buffer)

  Get the Prochot Residency (since the boot time or last read of Prochot Residency) of the socket.

5.8.1 Detailed Description

Below functions provide interfaces to get Prochot and Prochot Residency for a given socket.

5.8.2 Function Documentation

5.8.2.1 read_prochot_status()

```
oob_status_t read_prochot_status (
            uint8_t soc_num,
            uint32_t ∗ buffer )
```

Get the Prochot Status of the socket with provided socket index.

Given a socket and a pointer to a uint32_t, this function will get the Prochot status as active/1 or inactive/0

Parameters

in	*soc_num*	Socket index.
in,out	*buffer*	a pointer to uint32_t to which the Prochot status will be copied

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.8.2.2 read_prochot_residency()

```
oob_status_t read_prochot_residency (
            uint8_t soc_num,
            float ∗ buffer )
```

Get the Prochot Residency (since the boot time or last read of Prochot Residency) of the socket.

Given a socket and a pointer to a uint32_t, this function will get the Prochot residency as a percentage

Parameters

`in`	*soc_num*	Socket index.
`in,out`	*buffer*	a pointer to float to which the Prochot residency will be copied

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.9 Dram and other features Query

Functions

- oob_status_t read_dram_throttle (uint8_t soc_num, uint32_t ∗buffer)

  Read Dram Throttle will always read the highest percentage value.
- oob_status_t write_dram_throttle (uint8_t soc_num, uint32_t limit)

  Set Dram Throttle value in terms of percentage.
- oob_status_t read_nbio_error_logging_register (uint8_t soc_num, struct nbio_err_log nbio, uint32_t ∗buffer)

  Read NBIO Error Logging Register.
- oob_status_t read_iod_bist (uint8_t soc_num, uint32_t ∗buffer)

  Read IOD Bist status.
- oob_status_t read_ccd_bist_result (uint8_t soc_num, uint32_t input, uint32_t ∗buffer)

  Read CCD Bist status. Results are read for each CCD present in the system.
- oob_status_t read_ccx_bist_result (uint8_t soc_num, uint32_t value, uint32_t ∗ccx_bist)

  Read CPU Core Complex Bist result. results are read for each Logical CCX instance number and returns a value which is the concatenation of L3 pass status and all cores in the complex(n:0).
- oob_status_t read_cclk_freq_limit (uint8_t soc_num, uint32_t ∗cclk_freq)

  Read CCLK frequency limit for the given socket.
- oob_status_t read_socket_c0_residency (uint8_t soc_num, uint32_t ∗c0_res)

  Read socket C0 residency.
- oob_status_t read_ddr_bandwidth (uint8_t soc_num, struct max_ddr_bw ∗max_ddr)

  Get the Theoretical maximum DDR Bandwidth of the system in GB/s, Current utilized DDR Bandwidth (Read + Write) in GB/s and Current utilized DDR Bandwidth as a percentage of theoretical maximum.

5.9.1 Detailed Description

5.9.2 Function Documentation

5.9.2.1 read_dram_throttle()

```
oob_status_t read_dram_throttle (
             uint8_t soc_num,
             uint32_t * buffer )
```

Read Dram Throttle will always read the highest percentage value.

This function will always read the highest percentage value as represented by PROCHOT throttle or write dram throttle.

Parameters

in	*soc_num*	Socket index.
out	*buffer*	is to read the dram throttle in % (0 - 100).

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.9.2.2 write_dram_throttle()

```
oob_status_t write_dram_throttle (
              uint8_t soc_num,
              uint32_t limit )
```

Set Dram Throttle value in terms of percentage.

This function will set the dram throttle of the provided value limit for the given socket.

Parameters

in	*soc_num*	Socket index.
in	*limit*	that indicates the desired limit as per SSP PPR write can be between 0 to 80% to for a given socket

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.9.2.3 read_nbio_error_logging_register()

```
oob_status_t read_nbio_error_logging_register (
              uint8_t soc_num,
              struct nbio_err_log nbio,
              uint32_t * buffer )
```

Read NBIO Error Logging Register.

Given a socket, quadrant and register offset as `input`, this function will read NBIOErrorLoggingRegister.

Parameters

in	*soc_num*	Socket index.
in	*nbio*	nbio_err_log Struct containing nbio quadrant and offset.
out	*buffer*	is to read NBIOErrorLoggingRegiter(register value).

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.9.2.4 read_iod_bist()

```
oob_status_t read_iod_bist (
             uint8_t soc_num,
             uint32_t * buffer )
```

Read IOD Bist status.

This function will read IOD Bist result for the given socket.

Parameters

in	*soc_num*	Socket index.
out	*buffer*	is to read IODBistResult 0 = Bist pass, 1 = Bist fail

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.9.2.5 read_ccd_bist_result()

```
oob_status_t read_ccd_bist_result (
             uint8_t soc_num,
             uint32_t input,
             uint32_t * buffer )
```

Read CCD Bist status. Results are read for each CCD present in the system.

Given a socket bus number and address, Logical CCD instance number as `input`, this function will read CCD↩ BistResult.

Parameters

in	*soc_num*	Socket index.
in	*input*	is a Logical CCD instance number.
out	*buffer*	is to read CCDBistResult 0 = Bist pass, 1 = Bist fail

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.9.2.6 read_ccx_bist_result()

```
oob_status_t read_ccx_bist_result (
            uint8_t soc_num,
            uint32_t value,
            uint32_t * ccx_bist )
```

Read CPU Core Complex Bist result. results are read for each Logical CCX instance number and returns a value which is the concatenation of L3 pass status and all cores in the complex(n:0).

Given a socket bus number, address, Logical CCX instance number as `input`, this function will read CCXBist←Result.

Parameters

in	*soc_num*	Socket index.
in	*value*	is a Logical CCX instance number. Valid values [0, (k -1)] where k is the number of logical CCX instances.
out	*ccx_bist*	result is concatenation of bist results for all cores[31:16] in the complex(n:0) L3 bist[15:0], where n num of cores in CCX.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.9.2.7 read_cclk_freq_limit()

```
oob_status_t read_cclk_freq_limit (
            uint8_t soc_num,
            uint32_t * cclk_freq )
```

Read CCLK frequency limit for the given socket.

This function will read CPU core clock frequency limit for the given socket.

Parameters

in	*soc_num*	Socket index.
out	*cclk_freq*	CPU core clock frequency limit [MHz]

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.9.2.8 read_socket_c0_residency()

```
oob_status_t read_socket_c0_residency (
             uint8_t soc_num,
             uint32_t * c0_res )
```

Read socket C0 residency.

This function will provides average C0 residency across all cores in the socket. 100% specifies that all enabled cores in the socket are runningin C0.

Parameters

in	*soc_num*	Socket index.
out	*c0_res*	is to read Socket C0 residency[%].

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.9.2.9 read_ddr_bandwidth()

```
oob_status_t read_ddr_bandwidth (
             uint8_t soc_num,
             struct max_ddr_bw * max_ddr )
```

Get the Theoretical maximum DDR Bandwidth of the system in GB/s, Current utilized DDR Bandwidth (Read + Write) in GB/s and Current utilized DDR Bandwidth as a percentage of theoretical maximum.

Parameters

in	*soc_num*	Socket index.
out	*max_ddr*	max_ddr_bw struct containing max bandwidth, utilized bandwidth and utilized bandwidth percentage.

Return values

OOB_SUCCESS	is returned upon successful call.

Return values

Non-zero	is returned upon failure.

5.10 using CPUID Register Access

Below function provide interface to read the processor info using CPUID register. output from commmand will be written into the buffer.

Functions

- oob_status_t esmi_get_vendor_id (uint8_t soc_num, char ∗vendor_id)

    Get the number of logical cores per socket.
- oob_status_t esmi_get_processor_info (uint8_t soc_num, struct processor_info ∗proc_info)

    Get the number of logical cores per socket.
- oob_status_t esmi_get_logical_cores_per_socket (uint8_t soc_num, uint32_t ∗logical_cores_per_socket)

    Get the number of logical cores per socket.
- oob_status_t esmi_get_threads_per_socket (uint8_t soc_num, uint32_t ∗threads_per_socket)

    Get the number of threads per socket.
- oob_status_t esmi_get_threads_per_core (uint8_t soc_num, uint32_t ∗threads_per_core)

    Get number of threads per core.

5.10.1 Detailed Description

Below function provide interface to read the processor info using CPUID register. output from commmand will be written into the buffer.

5.10.2 Function Documentation

5.10.2.1 esmi_get_vendor_id()

```
oob_status_t esmi_get_vendor_id (
             uint8_t soc_num,
             char ∗ vendor_id )
```

Get the number of logical cores per socket.

Get the processor vendor

Parameters

in	*soc_num*	Socket index.
out	*vendor↔_id*	to get the processor vendor, 12 byte RO value

Return values

uint32↔_t	is returned upon successful call.

5.10.2.2 esmi_get_processor_info()

```
oob_status_t esmi_get_processor_info (
            uint8_t soc_num,
            struct processor_info * proc_info )
```

Get the number of logical cores per socket.

Get the effective family, model and step_id of the processor.

Parameters

in	*soc_num*	Socket index.
out	*proc_info*	to get family, model & stepping identifier

Return values

uint32←↩ _t	is returned upon successful call.

5.10.2.3 esmi_get_logical_cores_per_socket()

```
oob_status_t esmi_get_logical_cores_per_socket (
            uint8_t soc_num,
            uint32_t * logical_cores_per_socket )
```

Get the number of logical cores per socket.

Get the total number of logical cores in a socket.

Parameters

in	*soc_num*	Socket index.
in,out	*logical_cores_per_socket*	is returned

Return values

logical_cores_per_socket	is returned upon successful call.

5.10.2.4 esmi_get_threads_per_socket()

```
oob_status_t esmi_get_threads_per_socket (
              uint8_t soc_num,
              uint32_t * threads_per_socket )
```

Get the number of threads per socket.

Get the total number of threads in a socket.

Parameters

in	*soc_num*	Socket index.
in,out	*threads_per_socket*	is returned

Return values

threads_per_socket	is returned upon successful call.

5.10.2.5 esmi_get_threads_per_core()

```
oob_status_t esmi_get_threads_per_core (
              uint8_t soc_num,
              uint32_t * threads_per_core )
```

Get number of threads per core.

Get the number of threads per core.

Parameters

in	*soc_num*	Socket index.
in,out	*threads_per_core*	is returned

Return values

threads_per_core	is returned upon successful call.

5.11 SB_RMI Read Processor Register Access

Below function provide interface to read the SB-RMI MCA MSR register. output from MCA MSR commmand will be written into the buffer.

Functions

- oob_status_t esmi_oob_read_msr (uint8_t soc_num, uint32_t thread, uint32_t msraddr, uint64_t ∗buffer)

  Read the MCA MSR register for a given thread.

5.11.1 Detailed Description

Below function provide interface to read the SB-RMI MCA MSR register. output from MCA MSR commmand will be written into the buffer.

5.11.2 Function Documentation

5.11.2.1 esmi_oob_read_msr()

```
oob_status_t esmi_oob_read_msr (
             uint8_t soc_num,
             uint32_t thread,
             uint32_t msraddr,
             uint64_t * buffer )
```

Read the MCA MSR register for a given thread.

Given a `thread` and SB-RMI register command, this function reads msr value.

Parameters

in	*soc_num*	Socket index.
in	*thread*	is a particular thread in the system.
in	*msraddr*	MCA MSR register to read
out	*buffer*	is to hold the return output of msr value.

Return values

OOB_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure.

5.12 SB-RMI CPUID Register Access

Below function provide interface to get the CPUID access via the SBRMI.

Functions

- oob_status_t esmi_oob_cpuid (uint8_t soc_num, uint32_t thread, uint32_t ∗eax, uint32_t ∗ebx, uint32_t ∗ecx, uint32_t ∗edx)

  Read CPUID functionality for a particular thread in a system.
- oob_status_t esmi_oob_cpuid_eax (uint8_t soc_num, uint32_t thread, uint32_t fn_eax, uint32_t fn_ecx, uint32_t ∗eax)

  Read eax register on CPUID functionality.
- oob_status_t esmi_oob_cpuid_ebx (uint8_t soc_num, uint32_t thread, uint32_t fn_eax, uint32_t fn_ecx, uint32_t ∗ebx)

  Read ebx register on CPUID functionality.
- oob_status_t esmi_oob_cpuid_ecx (uint8_t soc_num, uint32_t thread, uint32_t fn_eax, uint32_t fn_ecx, uint32_t ∗ecx)

  Read ecx register on CPUID functionality.
- oob_status_t esmi_oob_cpuid_edx (uint8_t soc_num, uint32_t thread, uint32_t fn_eax, uint32_t fn_ecx, uint32_t ∗edx)

  Read edx register on CPUID functionality.
- oob_status_t read_max_threads_per_l3 (uint8_t soc_num, uint32_t ∗threads_l3)

  Read max threads per L3 cache.

5.12.1 Detailed Description

Below function provide interface to get the CPUID access via the SBRMI.

Output from CPUID commmand will be written into registers eax, ebx, ecx and edx.

5.12.2 Function Documentation

5.12.2.1 esmi_oob_cpuid()

```
oob_status_t esmi_oob_cpuid (
            uint8_t soc_num,
            uint32_t thread,
            uint32_t * eax,
            uint32_t * ebx,
            uint32_t * ecx,
            uint32_t * edx )
```

Read CPUID functionality for a particular thread in a system.

Given a `thread` , `eax` as function input and `ecx` as extended function input. this function will get the cpuid details for a particular thread in a pointer to `eax`, `ebx`, `ecx`, `edx`

Parameters

in	*soc_num*	Socket index.
in	*thread*	is a particular thread in the system.
in,out	*eax*	a pointer uint32_t to get eax value
out	*ebx*	a pointer uint32_t to get ebx value
in,out	*ecx*	a pointer uint32_t to get ecx value
out	*edx*	a pointer uint32_t to get edx value

Return values

OOB_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure.

5.12.2.2 esmi_oob_cpuid_eax()

```
oob_status_t esmi_oob_cpuid_eax (
            uint8_t soc_num,
            uint32_t thread,
            uint32_t fn_eax,
            uint32_t fn_ecx,
            uint32_t * eax )
```

Read eax register on CPUID functionality.

Given a `thread`, `fn_eax` as function and `fn_ecx` as extended function input, this function will get the cpuid details for a particular thread at `eax`.

Parameters

in	*soc_num*	Socket index.
in	*thread*	is a particular thread in the system.
in	*fn_eax*	cpuid function
in	*fn_ecx*	cpuid extended function
out	*eax*	is to read eax from cpuid functionality.

Return values

OOB_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure.

5.12.2.3 esmi_oob_cpuid_ebx()

```
oob_status_t esmi_oob_cpuid_ebx (
            uint8_t soc_num,
            uint32_t thread,
            uint32_t fn_eax,
            uint32_t fn_ecx,
            uint32_t * ebx )
```

Read ebx register on CPUID functionality.

Given a `thread`, `fn_eax` as function and `fn_ecx` as extended function input, this function will get the cpuid details for a particular thread at `ebx`.

Parameters

in	*soc_num*	Socket index.
in	*thread*	is a particular thread in the system.
in	*fn_eax*	cpuid function
in	*fn_ecx*	cpuid extended function
out	*ebx*	is to read ebx from cpuid functionality.

Return values

OOB_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure.

5.12.2.4 esmi_oob_cpuid_ecx()

```
oob_status_t esmi_oob_cpuid_ecx (
            uint8_t soc_num,
            uint32_t thread,
            uint32_t fn_eax,
            uint32_t fn_ecx,
            uint32_t * ecx )
```

Read ecx register on CPUID functionality.

Given a `thread`, `fn_eax` as function and `fn_ecx` as extended function input, this function will get the cpuid details for a particular thread at `ecx`.

Parameters

in	*soc_num*	Socket index.
in	*thread*	is a particular thread in the system.
in	*fn_eax*	cpuid function
in	*fn_ecx*	cpuid extended function
out	*ecx*	is to read ecx from cpuid functionality.

Return values

OOB_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure.

5.12.2.5 esmi_oob_cpuid_edx()

```
oob_status_t esmi_oob_cpuid_edx (
              uint8_t soc_num,
              uint32_t thread,
              uint32_t fn_eax,
              uint32_t fn_ecx,
              uint32_t * edx )
```

Read edx register on CPUID functionality.

Given a `thread`, `fn_eax` as function and `fn_ecx` as extended function input, this function will get the cpuid details for a particular thread at `edx`.

Parameters

in	*soc_num*	Socket index.
in	*thread*	is a particular thread in the system.
in	*fn_eax*	cpuid function
in	*fn_ecx*	cpuid extended function
out	*edx*	is to read edx from cpuid functionality.

Return values

OOB_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure.

5.12.2.6 read_max_threads_per_l3()

```
oob_status_t read_max_threads_per_l3 (
              uint8_t soc_num,
              uint32_t * threads_l3 )
```

Read max threads per L3 cache.

Reads max threads per L3 cache.

Parameters

in	*soc_num*	Socket index.
out	*threads↩_l3*	threads per L3 cache.

Return values

OOB_SUCCESS	is returned upon successful call.

Return values

None-zero	is returned upon failure.

5.13 SB-RMI Register Read Byte Protocol

The SB-RMI registers can be read or written from the SMBus interface using the SMBus defined PEC-optional Read Byte and Write Byte protocols with the SB-RMI register number in the command byte.

Functions

- oob_status_t read_sbrmi_revision (uint8_t soc_num, uint8_t ∗buffer)

  Read one byte from a given SB_RMI register number provided socket index and buffer to get the read data for a particular SB-RMI command register.
- oob_status_t read_sbrmi_control (uint8_t soc_num, uint8_t ∗buffer)

  Read Control byte from SB_RMI register command.
- oob_status_t read_sbrmi_status (uint8_t soc_num, uint8_t ∗buffer)

  Read one byte of Status value from SB_RMI register command.
- oob_status_t read_sbrmi_readsize (uint8_t soc_num, uint8_t ∗buffer)

  This register specifies the number of bytes to return when using the block read protocol to read SBRMI_x[4F:10].
- oob_status_t read_sbrmi_threadenablestatus (uint8_t soc_num, uint8_t ∗buffer)

  Read one byte of Thread Status from SB_RMI register command.
- oob_status_t read_sbrmi_multithreadenablestatus (uint8_t soc_num, uint8_t ∗buffer)

  Read one byte of Thread Status from SB_RMI register command.
- oob_status_t read_sbrmi_swinterrupt (uint8_t soc_num, uint8_t ∗buffer)

  This register is used by the SMBus master to generate an interrupt to the processor to indicate that a message is available..
- oob_status_t read_sbrmi_threadnumber (uint8_t soc_num, uint8_t ∗buffer)

  This register indicates the maximum number of threads present.
- oob_status_t read_sbrmi_mp0_msg (uint8_t soc_num, uint8_t ∗buffer)

  This register will read the message running on the MP0.
- oob_status_t read_sbrmi_alert_status (uint8_t soc_num, uint8_t num_of_alert_status_reg, uint8_t ∗∗buffer)

  This function will read bit vector for all the threads. Value of 1 indicates MCE occured for the thread and is set by hardware.
- oob_status_t read_sbrmi_alert_mask (uint8_t soc_num, uint8_t num_of_alert_mask_reg, uint8_t ∗∗buffer)

  This function will read bit vector for all the threads. Value of 1 indicates alert signaling disabled for corresponding SBRMI::AlertStatus[MceStat] for the thread.
- oob_status_t read_sbrmi_inbound_msg (uint8_t soc_num, uint8_t ∗buffer)

  This register will read the inbound message.
- oob_status_t read_sbrmi_outbound_msg (uint8_t soc_num, uint8_t ∗buffer)

  This register will read the outbound message.
- oob_status_t read_sbrmi_threadnumberlow (uint8_t soc_num, uint8_t ∗buffer)

  This register indicates the low part of maximum number of threads.
- oob_status_t read_sbrmi_threadnumberhi (uint8_t soc_num, uint8_t ∗buffer)

  This register indicates the upper part of maximum number of threads.
- oob_status_t read_sbrmi_thread_cs (uint8_t soc_num, uint8_t ∗buffer)

  This register is used to read the thread cs.
- oob_status_t read_sbrmi_ras_status (uint8_t soc_num, uint8_t ∗buffer)

  This register will read the ras status.
- oob_status_t clear_sbrmi_ras_status (uint8_t soc_num, uint8_t buffer)

  This API will clear ras status register.

5.13.1 Detailed Description

The SB-RMI registers can be read or written from the SMBus interface using the SMBus defined PEC-optional Read Byte and Write Byte protocols with the SB-RMI register number in the command byte.

5.13.2 Function Documentation

5.13.2.1 read_sbrmi_revision()

```
oob_status_t read_sbrmi_revision (
            uint8_t soc_num,
            uint8_t * buffer )
```

Read one byte from a given SB_RMI register number provided socket index and buffer to get the read data for a particular SB-RMI command register.

Given a socket index `socket_ind` and a pointer to hold the output at uint8_t `buffer`, this function will get the value from a particular command of SB_RMI register.

Parameters

in	*soc_num*	Socket uindex.
in,out	*buffer*	a pointer to a uint8_t that indicates value to hold

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

This value specifies the APML specification revision that the product is compliant to. 0x10 = 1.0x Revision.

5.13.2.2 read_sbrmi_alert_status()

```
oob_status_t read_sbrmi_alert_status (
            uint8_t soc_num,
            uint8_t num_of_alert_status_reg,
            uint8_t ** buffer )
```

This function will read bit vector for all the threads. Value of 1 indicates MCE occured for the thread and is set by hardware.

Parameters

in	*soc_num*	Socket index.
in	*num_of_alert_status_reg*	number of alert status registers.
in,out	*buffer*	a pointer to read all "num_of_alert_status_reg" of alert status registers. Buffer length should be equal to "num_of_alert_status_reg" value.

Return values

OOB_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure.

5.13.2.3 read_sbrmi_alert_mask()

```
oob_status_t read_sbrmi_alert_mask (
             uint8_t soc_num,
             uint8_t num_of_alert_mask_reg,
             uint8_t ** buffer )
```

This function will read bit vector for all the threads. Value of 1 indicates alert signaling disabled for corresponding SBRMI::AlertStatus[MceStat] for the thread.

Parameters

in	*soc_num*	Socket index.
in	*num_of_alert_mask_reg*	number of alert mask registers.
in,out	*buffer*	a pointer to read all "num_of_alert_mask_reg" of alert mask registers. Buffer length should be equal to "num_of_alert_mask_reg" value.

Return values

OOB_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure.

5.13.2.4 clear_sbrmi_ras_status()

```
oob_status_t clear_sbrmi_ras_status (
             uint8_t soc_num,
             uint8_t buffer )
```

This API will clear ras status register.

Parameters

in	*soc_num*	Socket index.
in	*buffer*	bit mask to clear ras status bits

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14 SBTSI Register Read Byte Protocol

Below functions provide interface to read one byte from the SB-TSI register and output is from a given SB_TSI register command.

Functions

- oob_status_t read_sbtsi_cpuinttemp (uint8_t soc_num, uint8_t ∗buffer)

  integer CPU temperature value The CPU temperature is calculated by adding the CPU temperature offset(SBT↩ SI::CpuTempOffInt, SBTSI::CpuTempOffDec) to the processor control temperature (Tctl). SBTSI::CpuTempInt and SBTSI::CpuTempDec combine to return the CPU temperature.
- oob_status_t read_sbtsi_status (uint8_t soc_num, uint8_t ∗buffer)

  Status register is Read-only, volatile field If SBTSI::AlertConfig[AlertCompEn] == 0 , the temperature alert is latched high until the alert is read. If SBTSI::AlertConfig[AlertCompEn] == 1, the alert is cleared when the temperature does not meet the threshold conditions for temperature and number of samples.
- oob_status_t read_sbtsi_config (uint8_t soc_num, uint8_t ∗buffer)

  The bits in this register are Read-only and can be written by Writing to the corresponding bits in SBTSI::ConfigWr.
- oob_status_t read_sbtsi_updaterate (uint8_t soc_num, float ∗buffer)

  This register value specifies the rate at which CPU temperature is compared against the temperature thresholds to determine if an alert event has occurred.
- oob_status_t write_sbtsi_updaterate (uint8_t soc_num, float uprate)

  This register value specifies the rate at which CPU temperature is compared against the temperature thresholds to determine if an alert event has occurred.
- oob_status_t read_sbtsi_hitempint (uint8_t soc_num, uint8_t ∗buffer)

  This value specifies the integer portion of the high temperature threshold. The high temperature threshold specifies the CPU temperature that causes ALERT_L to assert if the CPU temperature is greater than or equal to the threshold.
- oob_status_t read_sbtsi_lotempint (uint8_t soc_num, uint8_t ∗buffer)

  This value specifies the integer portion of the low temperature threshold. The low temperature threshold specifies the CPU temperature that causes ALERT_L to assert if the CPU temperature is less than or equal to the threshold.
- oob_status_t read_sbtsi_configwrite (uint8_t soc_num, uint8_t ∗buffer)

  This register provides write access to SBTSI::Config.
- oob_status_t read_sbtsi_cputempdecimal (uint8_t soc_num, float ∗buffer)

  The value returns the decimal portion of the CPU temperature.
- oob_status_t read_sbtsi_cputempoffint (uint8_t soc_num, uint8_t ∗temp_int)

  SBTSI::CpuTempOffInt and SBTSI::CpuTempOffDec combine to specify the CPU temperature offset.
- oob_status_t read_sbtsi_cputempoffdec (uint8_t soc_num, float ∗temp_dec)

  This value specifies the decimal/fractional portion of the CPU temperature offset added to Tctl to calculate the CPU temperature.
- oob_status_t read_sbtsi_hitempdecimal (uint8_t soc_num, float ∗temp_dec)

  This value specifies the decimal portion of the high temperature threshold.
- oob_status_t read_sbtsi_lotempdecimal (uint8_t soc_num, float ∗temp_dec)

  value specifies the decimal portion of the low temperature threshold.
- oob_status_t read_sbtsi_timeoutconfig (uint8_t soc_num, uint8_t ∗timeout)

  value specifies 0=SMBus defined timeout support disabled. 1=SMBus defined timeout support enabled. SMBus timeout enable. If SB-RMI is in use, SMBus timeouts should be enabled or disabled in a consistent manner on both interfaces. SMBus defined timeouts are not disabled for SB-RMI when this bit is set to 0.
- oob_status_t read_sbtsi_alertthreshold (uint8_t soc_num, uint8_t ∗samples)

  Specifies the number of consecutive CPU temperature samples for which a temperature alert condition needs to remain valid before the corresponding alert bit is set.
- oob_status_t read_sbtsi_alertconfig (uint8_t soc_num, uint8_t ∗mode)

  Status register is Read-only, volatile field If SBTSI::AlertConfig[AlertCompEn] == 0 , the temperature alert is latched high until the alert is read. If SBTSI::AlertConfig[AlertCompEn] == 1, the alert is cleared when the temperature does not meet the threshold conditions for temperature and number of samples.

- oob_status_t read_sbtsi_manufid (uint8_t soc_num, uint8_t ∗man_id)

  Returns the AMD manufacture ID.
- oob_status_t read_sbtsi_revision (uint8_t soc_num, uint8_t ∗rivision)

  Specifies the SBI temperature sensor interface revision.
- oob_status_t sbtsi_get_cputemp (uint8_t soc_num, float ∗cpu_temp)

  CPU temperature value The CPU temperature is calculated by adding SBTSI::CpuTempInt and SBTSI::CpuTempDec combine to return the CPU temperature.
- oob_status_t sbtsi_get_temp_status (uint8_t soc_num, uint8_t ∗loalert, uint8_t ∗hialert)

  Status register is Read-only, volatile field If SBTSI::AlertConfig[AlertCompEn] == 0 , the temperature alert is latched high until the alert is read. If SBTSI::AlertConfig[AlertCompEn] == 1, the alert is cleared when the temperature does not meet the threshold conditions for temperature and number of samples.
- oob_status_t sbtsi_get_config (uint8_t soc_num, uint8_t ∗al_mask, uint8_t ∗run_stop, uint8_t ∗read_ord, uint8_t ∗ara)

  The bits in this register are Read-only and can be written by Writing to the corresponding bits in SBTSI::ConfigWr.
- oob_status_t sbtsi_set_configwr (uint8_t soc_num, uint8_t mode, uint8_t config_mask)

  The bits in this register are defined sbtsi_config_write and can be written by writing to the corresponding bits in SBTSI::ConfigWr.
- oob_status_t sbtsi_get_timeout (uint8_t soc_num, uint8_t ∗timeout_en)

  To verify if timeout support enabled or disabled.
- oob_status_t sbtsi_set_timeout_config (uint8_t soc_num, uint8_t mode)

  To enable/disable timeout support.
- oob_status_t sbtsi_set_hitemp_threshold (uint8_t soc_num, float hitemp_thr)

  This value set the high temperature threshold. The high temperature threshold specifies the CPU temperature that causes ALERT_L to assert if the CPU temperature is greater than or equal to the threshold.
- oob_status_t sbtsi_set_lotemp_threshold (uint8_t soc_num, float lotemp_thr)

  This value set the low temperature threshold. The low temperature threshold specifies the CPU temperature that causes ALERT_L to assert if the CPU temperature is less than or equal to the threshold.
- oob_status_t sbtsi_get_hitemp_threshold (uint8_t soc_num, float ∗hitemp_thr)

  This value specifies the high temperature threshold. The high temperature threshold specifies the CPU temperature that causes ALERT_L to assert if the CPU temperature is greater than or equal to the threshold.
- oob_status_t sbtsi_get_lotemp_threshold (uint8_t soc_num, float ∗lotemp_thr)

  This value specifies the low temperature threshold. The low temperature threshold specifies the CPU temperature that causes ALERT_L to assert if the CPU temperature is less than or equal to the threshold.
- oob_status_t read_sbtsi_cputempoffset (uint8_t soc_num, float ∗temp_offset)

  SBTSI::CpuTempOffInt and SBTSI::CpuTempOffDec combine to specify the CPU temperature offset.
- oob_status_t write_sbtsi_cputempoffset (uint8_t soc_num, float temp_offset)

  SBTSI::CpuTempOffInt and SBTSI::CpuTempOffDec combine to set the CPU temperature offset.
- oob_status_t sbtsi_set_alert_threshold (uint8_t soc_num, uint8_t samples)

  Specifies the number of consecutive CPU temperature samples for which a temperature alert condition needs to remain valid before the corresponding alert bit is set.
- oob_status_t sbtsi_set_alert_config (uint8_t soc_num, uint8_t mode)

  Alert comparator mode enable.

5.14.1 Detailed Description

Below functions provide interface to read one byte from the SB-TSI register and output is from a given SB_TSI register command.

5.14.2 Function Documentation

5.14.2.1 read_sbtsi_cpuinttemp()

```
oob_status_t read_sbtsi_cpuinttemp (
             uint8_t soc_num,
             uint8_t * buffer )
```

integer CPU temperature value The CPU temperature is calculated by adding the CPU temperature offset(SBT↩
SI::CpuTempOffInt, SBTSI::CpuTempOffDec) to the processor control temperature (Tctl). SBTSI::CpuTempInt and SBTSI::CpuTempDec combine to return the CPU temperature.

This field returns the integer portion of the CPU temperature

Parameters

in	*soc_num*	Socket index.
in,out	*buffer*	a pointer to hold the cpu temperature

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.2 read_sbtsi_status()

```
oob_status_t read_sbtsi_status (
             uint8_t soc_num,
             uint8_t * buffer )
```

Status register is Read-only, volatile field If SBTSI::AlertConfig[AlertCompEn] == 0 , the temperature alert is latched high until the alert is read. If SBTSI::AlertConfig[AlertCompEn] == 1, the alert is cleared when the temperature does not meet the threshold conditions for temperature and number of samples.

Parameters

in	*soc_num*	Socket index.
in,out	*buffer*	a pointer to hold the cpu temperature

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.3 read_sbtsi_config()

```
oob_status_t read_sbtsi_config (
              uint8_t soc_num,
              uint8_t * buffer )
```

The bits in this register are Read-only and can be written by Writing to the corresponding bits in SBTSI::ConfigWr.

Parameters

in	*soc_num*	Socket index.
in,out	*buffer*	a pointer to hold the cpu temperature

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.4 read_sbtsi_updaterate()

```
oob_status_t read_sbtsi_updaterate (
              uint8_t soc_num,
              float * buffer )
```

This register value specifies the rate at which CPU temperature is compared against the temperature thresholds to determine if an alert event has occurred.

Parameters

in	*soc_num*	Socket index.
in,out	*buffer*	a pointer to hold the cpu temperature

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.5 write_sbtsi_updaterate()

```
oob_status_t write_sbtsi_updaterate (
              uint8_t soc_num,
              float uprate )
```

This register value specifies the rate at which CPU temperature is compared against the temperature thresholds to determine if an alert event has occurred.

Parameters

in	*soc_num*	Socket index.
in	*uprate*	value to write in raw format

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.6 read_sbtsi_hitempint()

```
oob_status_t read_sbtsi_hitempint (
             uint8_t soc_num,
             uint8_t * buffer )
```

This value specifies the integer portion of the high temperature threshold. The high temperature threshold specifies the CPU temperature that causes ALERT_L to assert if the CPU temperature is greater than or equal to the threshold.

Parameters

in	*soc_num*	Socket index.
in,out	*buffer*	a pointer to hold the integer part of high cpu temp

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.7 read_sbtsi_lotempint()

```
oob_status_t read_sbtsi_lotempint (
             uint8_t soc_num,
             uint8_t * buffer )
```

This value specifies the integer portion of the low temperature threshold. The low temperature threshold specifies the CPU temperature that causes ALERT_L to assert if the CPU temperature is less than or equal to the threshold.

Parameters

in	*soc_num*	Socket index.
in,out	*buffer*	a pointer to hold the integer part of low cpu temp

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.8 read_sbtsi_configwrite()

```
oob_status_t read_sbtsi_configwrite (
             uint8_t soc_num,
             uint8_t * buffer )
```

This register provides write access to SBTSI::Config.

Parameters

in	*soc_num*	Socket index.
in,out	*buffer*	a pointer to hold the configuraion

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.9 read_sbtsi_cputempdecimal()

```
oob_status_t read_sbtsi_cputempdecimal (
             uint8_t soc_num,
             float * buffer )
```

The value returns the decimal portion of the CPU temperature.

Parameters

in	*soc_num*	Socket index.
in,out	*buffer*	a pointer to hold the cpu temperature decimal

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.10 read_sbtsi_cputempoffint()

```
oob_status_t read_sbtsi_cputempoffint (
             uint8_t soc_num,
             uint8_t * temp_int )
```

SBTSI::CpuTempOffInt and SBTSI::CpuTempOffDec combine to specify the CPU temperature offset.

Parameters

in	*soc_num*	Socket index.
in,out	*temp_int*	a pointer to hold the cpu offset interger

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.11 read_sbtsi_cputempoffdec()

```
oob_status_t read_sbtsi_cputempoffdec (
             uint8_t soc_num,
             float * temp_dec )
```

This value specifies the decimal/fractional portion of the CPU temperature offset added to Tctl to calculate the CPU temperature.

Parameters

in	*soc_num*	Socket index.
in,out	*temp_dec*	a pointer to hold the cpu offset decimal

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.12 read_sbtsi_hitempdecimal()

```
oob_status_t read_sbtsi_hitempdecimal (
             uint8_t soc_num,
             float * temp_dec )
```

This value specifies the decimal portion of the high temperature threshold.

Parameters

`in`	*soc_num*	Socket index.
`in,out`	*temp_dec*	a pointer to hold the decimal part of cpu high temp

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.13 read_sbtsi_lotempdecimal()

```
oob_status_t read_sbtsi_lotempdecimal (
            uint8_t soc_num,
            float * temp_dec )
```

value specifies the decimal portion of the low temperature threshold.

Parameters

`in`	*soc_num*	Socket index.
`in,out`	*temp_dec*	a pointer to hold the decimal part of cpu low temperature

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.14 read_sbtsi_timeoutconfig()

```
oob_status_t read_sbtsi_timeoutconfig (
            uint8_t soc_num,
            uint8_t * timeout )
```

value specifies 0=SMBus defined timeout support disabled. 1=SMBus defined timeout support enabled. SMBus timeout enable. If SB-RMI is in use, SMBus timeouts should be enabled or disabled in a consistent manner on both interfaces. SMBus defined timeouts are not disabled for SB-RMI when this bit is set to 0.

Parameters

`in`	*soc_num*	Socket index.
`in,out`	*timeout*	a pointer to hold the cpu timeout configuration

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.15 read_sbtsi_alertthreshold()

```
oob_status_t read_sbtsi_alertthreshold (
            uint8_t soc_num,
            uint8_t * samples )
```

Specifies the number of consecutive CPU temperature samples for which a temperature alert condition needs to remain valid before the corresponding alert bit is set.

Parameters

`in`	*soc_num*	Socket index.
`in,out`	*samples*	a pointer to hold the cpu temperature alert threshold

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.16 read_sbtsi_alertconfig()

```
oob_status_t read_sbtsi_alertconfig (
            uint8_t soc_num,
            uint8_t * mode )
```

Status register is Read-only, volatile field If SBTSI::AlertConfig[AlertCompEn] == 0 , the temperature alert is latched high until the alert is read. If SBTSI::AlertConfig[AlertCompEn] == 1, the alert is cleared when the temperature does not meet the threshold conditions for temperature and number of samples.

Parameters

`in`	*soc_num*	Socket index.
`in,out`	*mode*	a pointer to hold the cpu temperature alert configuration

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.17 read_sbtsi_manufid()

```
oob_status_t read_sbtsi_manufid (
             uint8_t soc_num,
             uint8_t * man_id )
```

Returns the AMD manufacture ID.

Parameters

in	*soc_num*	Socket index.
in,out	*man_id*	a pointer to hold the manufacture id

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.18 read_sbtsi_revision()

```
oob_status_t read_sbtsi_revision (
             uint8_t soc_num,
             uint8_t * rivision )
```

Specifies the SBI temperature sensor interface revision.

Parameters

in	*soc_num*	Socket index.
in,out	*rivision*	a pointer to hold the cpu temperature revision

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.19 sbtsi_get_cputemp()

```
oob_status_t sbtsi_get_cputemp (
             uint8_t soc_num,
             float * cpu_temp )
```

CPU temperature value The CPU temperature is calculated by adding SBTSI::CpuTempInt and SBTSI::CpuTemp↩ Dec combine to return the CPU temperature.

Parameters

in	*soc_num*	Socket index.
in,out	*cpu_temp*	a pointer to get temperature of the CPU

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.20 sbtsi_get_temp_status()

```
oob_status_t sbtsi_get_temp_status (
             uint8_t soc_num,
             uint8_t * loalert,
             uint8_t * hialert )
```

Status register is Read-only, volatile field If SBTSI::AlertConfig[AlertCompEn] == 0 , the temperature alert is latched high until the alert is read. If SBTSI::AlertConfig[AlertCompEn] == 1, the alert is cleared when the temperature does not meet the threshold conditions for temperature and number of samples.

Parameters

in	*soc_num*	Socket index.
in,out	*loalert*	1=> CPU temp is less than or equal to low temperature threshold for consecutive samples
in,out	*hialert*	1=> CPU temp is greater than or equal to high temperature threshold for consecutive samples

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.21 sbtsi_get_config()

```
oob_status_t sbtsi_get_config (
             uint8_t soc_num,
             uint8_t * al_mask,
             uint8_t * run_stop,
             uint8_t * read_ord,
             uint8_t * ara )
```

The bits in this register are Read-only and can be written by Writing to the corresponding bits in SBTSI::ConfigWr.

Parameters

in	*soc_num*	Socket index.
in,out	*al_mask*	0=> ALERT_L pin enabled. 1=> ALERT_L pin disabled and does not assert.
in,out	*run_stop*	0=> Updates to CpuTempInt and CpuTempDec and alert comparisons are enabled. 1=> Updates are disabled and alert comparisons are disabled.
in,out	*read_ord*	0=> Reading CpuTempInt causes the satate of CpuTempDec to be latched. 1=> Reading CpuTempInt causes the satate of CpuTempDec to be latched.
in,out	*ara*	1=> ARA response disabled.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.22 sbtsi_set_configwr()

```
oob_status_t sbtsi_set_configwr (
              uint8_t soc_num,
              uint8_t mode,
              uint8_t config_mask )
```

The bits in this register are defined sbtsi_config_write and can be written by writing to the corresponding bits in SBTSI::ConfigWr.

NOTE: Currently testing is not done for this API.

Parameters

in	*soc_num*	Socket index.
in	*mode*	value to update 0 or 1
in	*config_mask*	which bit need to update

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.23 sbtsi_get_timeout()

```
oob_status_t sbtsi_get_timeout (
              uint8_t soc_num,
              uint8_t * timeout_en )
```

To verify if timeout support enabled or disabled.

Parameters

in	*soc_num*	Socket index.
in,out	*timeout_en*	0=>SMBus defined timeout support disabled.

1=SMBus defined timeout support enabled. SMBus timeout enable. If SB-RMI is in use, SMBus timeouts should be enabled or disabled in a consistent manner on both interfaces. SMBus defined timeouts are not disabled for SB-RMI when this bit is set to 0.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.24 sbtsi_set_timeout_config()

```
oob_status_t sbtsi_set_timeout_config (
             uint8_t soc_num,
             uint8_t mode )
```

To enable/disable timeout support.

Parameters

in	*soc_num*	Socket index.
in	*mode*	0=>SMBus defined timeout support disabled.

1=>SMBus defined timeout support enabled. SMBus timeout enable. If SB-RMI is in use, SMBus timeouts should be enabled or disabled in a consistent manner on both interfaces. SMBus defined timeouts are not disabled for SB-RMI when this bit is set to 0.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.25 sbtsi_set_hitemp_threshold()

```
oob_status_t sbtsi_set_hitemp_threshold (
             uint8_t soc_num,
             float hitemp_thr )
```

This value set the high temperature threshold. The high temperature threshold specifies the CPU temperature that causes ALERT_L to assert if the CPU temperature is greater than or equal to the threshold.

Parameters

in	*soc_num*	Socket index.
in	*hitemp_thr*	Specifies the high temperature threshold

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.26 sbtsi_set_lotemp_threshold()

```
oob_status_t sbtsi_set_lotemp_threshold (
             uint8_t soc_num,
             float lotemp_thr )
```

This value set the low temperature threshold. The low temperature threshold specifies the CPU temperature that causes ALERT_L to assert if the CPU temperature is less than or equal to the threshold.

Parameters

in	*soc_num*	Socket index.
in	*lotemp_thr*	Specifies the low temperature threshold

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.27 sbtsi_get_hitemp_threshold()

```
oob_status_t sbtsi_get_hitemp_threshold (
             uint8_t soc_num,
             float * hitemp_thr )
```

This value specifies the high temperature threshold. The high temperature threshold specifies the CPU temperature that causes ALERT_L to assert if the CPU temperature is greater than or equal to the threshold.

Parameters

in	*soc_num*	Socket index.
in	*hitemp_thr*	Specifies the high temperature threshold

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.28 sbtsi_get_lotemp_threshold()

```
oob_status_t sbtsi_get_lotemp_threshold (
             uint8_t soc_num,
             float * lotemp_thr )
```

This value specifies the low temperature threshold. The low temperature threshold specifies the CPU temperature that causes ALERT_L to assert if the CPU temperature is less than or equal to the threshold.

Parameters

in	*soc_num*	Socket index.
in,out	*lotemp_thr*	Get the low temperature threshold

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.29 read_sbtsi_cputempoffset()

```
oob_status_t read_sbtsi_cputempoffset (
             uint8_t soc_num,
             float * temp_offset )
```

SBTSI::CpuTempOffInt and SBTSI::CpuTempOffDec combine to specify the CPU temperature offset.

Parameters

in	*soc_num*	Socket index.
in,out	*temp_offset*	to get the offset value for temperature

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.30 write_sbtsi_cputempoffset()

```
oob_status_t write_sbtsi_cputempoffset (
            uint8_t soc_num,
            float temp_offset )
```

SBTSI::CpuTempOffInt and SBTSI::CpuTempOffDec combine to set the CPU temperature offset.

Parameters

in	*soc_num*	Socket index.
in	*temp_offset*	to set the offset value for temperature

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.31 sbtsi_set_alert_threshold()

```
oob_status_t sbtsi_set_alert_threshold (
            uint8_t soc_num,
            uint8_t samples )
```

Specifies the number of consecutive CPU temperature samples for which a temperature alert condition needs to remain valid before the corresponding alert bit is set.

Parameters

in	*soc_num*	Socket index.
in	*samples*	Number of samples 0h: 1 sample 6h-1h: (value + 1) sample 7h: 8 sample

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

5.14.2.32 sbtsi_set_alert_config()

```
oob_status_t sbtsi_set_alert_config (
            uint8_t soc_num,
            uint8_t mode )
```

Alert comparator mode enable.

Parameters

in	*soc_num*	Socket index.
in	*mode*	0=> SBTSI::Status[TempHighAlert] & SBTSI::Status[TempLowAlert] are read-clear. 1=> SBTSI::Status[TempHighAlert] & SBTSI::Status[TempLowAlert] are read-only. ARA response disabled.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

Chapter 6

Data Structure Documentation

6.1 apml_encodings Struct Reference

APML link ID encodings for legacy platforms and MI300A. Structure contains the following members. Link ID encoding value and its name. MI300A APML Link ID Encoding: 00000011b: P2 00000100b: P3 00001000b: G0 00001001b: G1 00001010b: G2 00001011b: G3 00001100b: G4 00001101b: G5 00001110b: G6 00001111b: G7 For other platforms the APML Link ID Encoding: 00000001b: P0 00000010b: P1 00000100b: P2 00001000b: P3 00010000b: G0 00100000b: G1 01000000b: G2 10000000b: G3.

```
#include <apml_common.h>
```

Data Fields

- uint8_t val

    Link ID encoding value.
- char name [3]

    Link ID encoding name.

6.1.1 Detailed Description

APML link ID encodings for legacy platforms and MI300A. Structure contains the following members. Link ID encoding value and its name. MI300A APML Link ID Encoding: 00000011b: P2 00000100b: P3 00001000b: G0 00001001b: G1 00001010b: G2 00001011b: G3 00001100b: G4 00001101b: G5 00001110b: G6 00001111b: G7 For other platforms the APML Link ID Encoding: 00000001b: P0 00000010b: P1 00000100b: P2 00001000b: P3 00010000b: G0 00100000b: G1 01000000b: G2 10000000b: G3.

The documentation for this struct was generated from the following file:

- apml_common.h

6.2 dimm_power Struct Reference

DIMM power(mW), update rate(ms) and dimm address.

```
#include <esmi_mailbox.h>
```

Data Fields

- uint16_t power: 15

  Dimm power consumption.
- uint16_t update_rate: 9

  update rate in ms
- uint8_t dimm_addr

  Dimm address.

6.2.1 Detailed Description

DIMM power(mW), update rate(ms) and dimm address.

The documentation for this struct was generated from the following file:

- esmi_mailbox.h

6.3 dimm_thermal Struct Reference

DIMM thermal sensor (degree C), update rate and dimm address.

```
#include <esmi_mailbox.h>
```

Data Fields

- uint16_t sensor: 11

  Dimm thermal sensor.
- uint16_t update_rate: 9

  update rate in ms
- uint8_t dimm_addr

  Dimm address.

6.3.1 Detailed Description

DIMM thermal sensor (degree C), update rate and dimm address.

The documentation for this struct was generated from the following file:

- esmi_mailbox.h

6.4 dpm_level Struct Reference

Max and min LCK DPM level on a given NBIO ID. Valid Max and min DPM level values are 0 - 1.

```
#include <esmi_mailbox.h>
```

Data Fields

- uint8_t max_dpm_level

  Max LCLK DPM level [0 - 1].
- uint8_t min_dpm_level

  Min LCLK DPM level [0 - 1].

6.4.1 Detailed Description

Max and min LCK DPM level on a given NBIO ID. Valid Max and min DPM level values are 0 - 1.

The documentation for this struct was generated from the following file:

- esmi_mailbox.h

6.5 freq_limits Struct Reference

struct containing max frequency and min frequencey limit

```
#include <rmi_mailbox_mi300.h>
```

Data Fields

- uint16_t max

  Max clock frequency.
- uint16_t min

  Min clock frequency.

6.5.1 Detailed Description

struct containing max frequency and min frequencey limit

The documentation for this struct was generated from the following file:

- rmi_mailbox_mi300.h

6.6 host_status Struct Reference

struct containing power management controlling status, driving running status.

```
#include <rmi_mailbox_mi300.h>
```

Data Fields

- uint8_t controller_status: 1

  power mangagement controller status
- uint8_t driver_status: 1

  driver running status

6.6.1 Detailed Description

struct containing power management controlling status, driving running status.

The documentation for this struct was generated from the following file:

- rmi_mailbox_mi300.h

6.7 lclk_dpm_level_range Struct Reference

Max and Min Link frequency clock (LCLK) DPM level on a socket. 8 bit NBIO ID, dpm_level struct containing 8 bit max DPM level, 8 bit min DPM level.

```
#include <esmi_mailbox.h>
```

Data Fields

- uint8_t nbio_id

  NBIOD id (8 bit data [0 - 3])
- struct dpm_level dpm

  struct with max dpm, min dpm levels

6.7.1 Detailed Description

Max and Min Link frequency clock (LCLK) DPM level on a socket. 8 bit NBIO ID, dpm_level struct containing 8 bit max DPM level, 8 bit min DPM level.

The documentation for this struct was generated from the following file:

- esmi_mailbox.h

6.8 link_id_bw_type Struct Reference

APML LINK ID and Bandwidth type Information.It contains APML LINK ID Encoding. Non-MI300 Platforms Valid Link ID encodings are 1(P0), 2(P1), 4(P2), 8(P3), 16(G0), 32(G1), 64(G2), 128(G3). Valid APML MI300 APML LINK ID Encoding. Valid Link ID encodings are 3(P2), 4(P3), 8(G0), 9(G1), 10(G2), 11(G3), 12(G4), 13(G5), 14(G6), 15(G7). IO Bandwidth types 1(Aggregate_BW), 2 (Read BW), 4 (Write BW).

```
#include <esmi_mailbox.h>
```

Data Fields

- apml_io_bw_encoding bw_type

  Bandwidth Type Information [1, 2, 4].
- uint8_t link_id

  Link ID [3,4,8,9,10,11,12,13,14,15] for MI300 platforms.

6.8.1 Detailed Description

APML LINK ID and Bandwidth type Information.It contains APML LINK ID Encoding. Non-MI300 Platforms Valid Link ID encodings are 1(P0), 2(P1), 4(P2), 8(P3), 16(G0), 32(G1), 64(G2), 128(G3). Valid APML MI300 APML LINK ID Encoding. Valid Link ID encodings are 3(P2), 4(P3), 8(G0), 9(G1), 10(G2), 11(G3), 12(G4), 13(G5), 14(G6), 15(G7). IO Bandwidth types 1(Aggregate_BW), 2 (Read BW), 4 (Write BW).

6.8.2 Field Documentation

6.8.2.1 link_id

```
uint8_t link_id_bw_type::link_id
```

Link ID [3,4,8,9,10,11,12,13,14,15] for MI300 platforms.

Link ID [1,2,4,8,16,32,64,128] for Non-MI300 platforms

The documentation for this struct was generated from the following file:

- esmi_mailbox.h

6.9 max_ddr_bw Struct Reference

Structure for Max DDR bandwidth and utilization. It contains max bandwidth(12 bit data) in GBps, current utilization bandwidth(12 bit data) in GBps, current utilized bandwidth(8 bit data) in percentage.

```
#include <esmi_mailbox.h>
```

Data Fields

- uint16_t max_bw: 12

  Max Bandwidth (12 bit data)
- uint16_t utilized_bw: 12

  Utilized Bandwidth (12 bit data)
- uint8_t utilized_pct

  Utliized Bandwidth percentage.

6.9.1 Detailed Description

Structure for Max DDR bandwidth and utilization. It contains max bandwidth(12 bit data) in GBps, current utilization bandwidth(12 bit data) in GBps, current utilized bandwidth(8 bit data) in percentage.

The documentation for this struct was generated from the following file:

- esmi_mailbox.h

6.10 max_mem_bw Struct Reference

Structure for Max DDR/HBM bandwidth and utilization. It contains max bandwidth(16 bit data) in GBps, current utilization bandwidth(Read+Write)(16 bit data) in GBps.

```
#include <rmi_mailbox_mi300.h>
```

Data Fields

- uint16_t max_bw

  Max Bandwidth (16 bit data)
- uint16_t utilized_bw

  Utilized Bandwidth (16 bit data)

6.10.1 Detailed Description

Structure for Max DDR/HBM bandwidth and utilization. It contains max bandwidth(16 bit data) in GBps, current utilization bandwidth(Read+Write)(16 bit data) in GBps.

The documentation for this struct was generated from the following file:

- rmi_mailbox_mi300.h

6.11 mca_bank Struct Reference

MCA bank information.It contains 16 bit Index for MCA Bank and 16 bit offset.

```
#include <esmi_mailbox.h>
```

Data Fields

- uint16_t offset

  Offset with in MCA Bank.
- uint16_t index

  Index of MCA Bank.

6.11.1 Detailed Description

MCA bank information.It contains 16 bit Index for MCA Bank and 16 bit offset.

The documentation for this struct was generated from the following file:

- esmi_mailbox.h

6.12 mclk_fclk_pstates Struct Reference

struct containing memory clock and fabric clock pstate mappings.

```
#include <rmi_mailbox_mi300.h>
```

Data Fields

- uint16_t mem_clk

  memory clock frequency in MHz
- uint16_t f_clk

  fabric clock frequnecy in MHz

6.12.1 Detailed Description

struct containing memory clock and fabric clock pstate mappings.

The documentation for this struct was generated from the following file:

- rmi_mailbox_mi300.h

6.13 nbio_err_log Struct Reference

NBIO quadrant(8 bit data) and NBIO register offset(24 bit) data.

```
#include <esmi_mailbox.h>
```

Data Fields

- uint8_t quadrant

  < NBIO quadrant data
- uint32_t offset: 24

  < NBIO register offset (24 bit data)

6.13.1 Detailed Description

NBIO quadrant(8 bit data) and NBIO register offset(24 bit) data.

The documentation for this struct was generated from the following file:

- esmi_mailbox.h

6.14 pci_address Struct Reference

PCI address information .PCI address includes 4 bit segment, 12 bit aligned offset, 8 bit bus, 5 bit device info and 3 bit function.

```
#include <esmi_mailbox.h>
```

Data Fields

- uint8_t func: 3

  function (3 bit data)
- uint8_t device: 5

  device info (5 bit data)
- uint8_t bus

  bus (8 bit data)
- uint16_t offset: 12

  offset address (12 bit data)
- uint8_t segment: 4

  segment (4 bit data)

6.14.1 Detailed Description

PCI address information .PCI address includes 4 bit segment, 12 bit aligned offset, 8 bit bus, 5 bit device info and 3 bit function.

The documentation for this struct was generated from the following file:

- esmi_mailbox.h

6.15 processor_info Struct Reference

Read Proccessor Info.

```
#include <esmi_cpuid_msr.h>
```

Data Fields

- uint32_t family

  Processor Family in hexa.
- uint32_t model

  Processor Model in hexa.
- uint32_t step_id

  Stepping Identifier in hexa.

6.15.1 Detailed Description

Read Proccessor Info.

The documentation for this struct was generated from the following file:

- esmi_cpuid_msr.h

6.16 pstate_freq Struct Reference

DF P-state frequency.It includes mem clock(16 bit data) frequency (DRAM memory clock), data fabric clock (12 bit data), UMC clock divider (UMC) (1 bit data).

```
#include <esmi_mailbox.h>
```

Data Fields

- uint16_t mem_clk

  DRAM Memory clock Frequency (MHz)(12 bit)
- uint16_t fclk: 12

  Data fabric clock (MHz)(12 bit data)
- uint8_t uclk: 1

  UMC clock divider (1 bit data)

6.16.1 Detailed Description

DF P-state frequency.It includes mem clock(16 bit data) frequency (DRAM memory clock), data fabric clock (12 bit data), UMC clock divider (UMC) (1 bit data).

The documentation for this struct was generated from the following file:

- esmi_mailbox.h

6.17 ras_df_err_chk Struct Reference

RAS df err validity check output status. Structure contains the following members. df_block_instances number of block instance with error log to report (0 - 256) err_log_len length of error log in bytes per instance (0 - 256).

```
#include <esmi_mailbox.h>
```

Data Fields

- uint16_t df_block_instances: 9

  Number of DF block instances.
- uint16_t err_log_len: 9

  len of er log in bytes per inst.

6.17.1 Detailed Description

RAS df err validity check output status. Structure contains the following members. df_block_instances number of block instance with error log to report (0 - 256) err_log_len length of error log in bytes per instance (0 - 256).

The documentation for this struct was generated from the following file:

- esmi_mailbox.h

6.18 ras_df_err_dump Union Reference

RAS df error dump input. Union contains the following members. input[0] 4 byte alligned offset in error log (0 - 255) input[1] DF block ID (0 - 255) input[2] Zero based index of DF block instance (0 - 255) input[3] Reserved data_in 32-bit data input.

```
#include <esmi_mailbox.h>
```

Data Fields

- uint8_t input [4]

  [2] block ID inst, [3] RESERVED
- uint32_t data_in

  32 bit data in for the DF err dump

6.18.1 Detailed Description

RAS df error dump input. Union contains the following members. input[0] 4 byte alligned offset in error log (0 - 255) input[1] DF block ID (0 - 255) input[2] Zero based index of DF block instance (0 - 255) input[3] Reserved data_in 32-bit data input.

6.18.2 Field Documentation

6.18.2.1 input

```
uint8_t ras_df_err_dump::input[4]
```

[2] block ID inst, [3] RESERVED

[0] offset, [1] DF block ID

The documentation for this union was generated from the following file:

- esmi_mailbox.h

6.19 ras_override_delay Struct Reference

BMC RAS override delay reset CPU on sync flood. The structure contains delay value override in mins [5 -120 mins], disable delay counter and stop delay counter. If disable delay counter is set ResetCpuOnSyncFlood response will NOT be delayed in the next SyncFlood regardless of the value specified in delay_val_override. If StopDelayCounter is set it stops the active delay countdown which extends the DelayResetCpuOnSyncFlood indefinitely and system will not reset.

```
#include <esmi_mailbox.h>
```

Data Fields

- uint8_t delay_val_override

    Delay value override [5 -120 mins].
- uint8_t disable_delay_counter: 1

    Disable delay counter.
- uint8_t stop_delay_counter: 1

    stop delay counter

6.19.1 Detailed Description

BMC RAS override delay reset CPU on sync flood. The structure contains delay value override in mins [5 -120 mins], disable delay counter and stop delay counter. If disable delay counter is set ResetCpuOnSyncFlood response will NOT be delayed in the next SyncFlood regardless of the value specified in delay_val_override. If StopDelayCounter is set it stops the active delay countdown which extends the DelayResetCpuOnSyncFlood indefinitely and system will not reset.

The documentation for this struct was generated from the following file:

- esmi_mailbox.h

6.20 statistics Struct Reference

struct containing statistics parameter of interest and output control pstate mappings.

```
#include <rmi_mailbox_mi300.h>
```

Data Fields

- uint16_t stat_param

  statistics parameter of interest
- uint16_t output_control

  Output control.

6.20.1 Detailed Description

struct containing statistics parameter of interest and output control pstate mappings.

The documentation for this struct was generated from the following file:

- rmi_mailbox_mi300.h

6.21 svi_port_domain Struct Reference

struct containing port and slave address.

```
#include <rmi_mailbox_mi300.h>
```

Data Fields

- uint8_t port: 2

  SVI port.
- uint8_t slave_addr: 3

  slave address

6.21.1 Detailed Description

struct containing port and slave address.

The documentation for this struct was generated from the following file:

- rmi_mailbox_mi300.h

6.22 temp_refresh_rate Struct Reference

DIMM temperature range and refresh rate, temperature update flag.

```
#include <esmi_mailbox.h>
```

Data Fields

- uint8_t range: 3

  temp refresh rate (3 bit data)
- uint8_t ref_rate: 1

  temp update flag (1 bit data)

6.22.1 Detailed Description

DIMM temperature range and refresh rate, temperature update flag.

The documentation for this struct was generated from the following file:

- esmi_mailbox.h

6.23 xgmi_speed_rate_n_width Struct Reference

struct containing xgmi speed rate in MHZ and link width in units of Gpbs. If link_width[0] = 1 then XGMI link X2 is supported. If link_width[1] = 1 then XGMI link X4 is supported. If Link_width[2] = 1 then XGMI link X4 is supported and similarly if link_width[3] = 1 then XGMI link X8 is supported.

```
#include <rmi_mailbox_mi300.h>
```

Data Fields

- uint16_t speed_rate

  Speed rate.
- uint8_t link_width: 4

  XGMI link width.

6.23.1 Detailed Description

struct containing xgmi speed rate in MHZ and link width in units of Gpbs. If link_width[0] = 1 then XGMI link X2 is supported. If link_width[1] = 1 then XGMI link X4 is supported. If Link_width[2] = 1 then XGMI link X4 is supported and similarly if link_width[3] = 1 then XGMI link X8 is supported.

The documentation for this struct was generated from the following file:

- rmi_mailbox_mi300.h

Chapter 7

File Documentation

7.1 apml.h File Reference

```
#include <stdbool.h>
#include <linux/amd-apml.h>
#include "apml_err.h"
```

Macros

- #define SBRMI "sbrmi"

  SBRMI module //.
- #define SBTSI "sbtsi"

  SBTSI module //.
- #define MAX_DEV_COUNT 8

  APML ADDRESSES count.

Enumerations

- enum sbrmi_outbnd_msg {
  SBRMI_OUTBNDMSG0 = 0x30, **SBRMI_OUTBNDMSG1**, **SBRMI_OUTBNDMSG2**, **SBRMI_OUTBNDM**←
  SG3,
  SBRMI_OUTBNDMSG4, **SBRMI_OUTBNDMSG5**, **SBRMI_OUTBNDMSG6**, **SBRMI_OUTBNDMSG7** }

  SBRMI outbound messages defined in the APML library.
- enum sbrmi_inbnd_msg {
  SBRMI_INBNDMSG0 = 0x38, **SBRMI_INBNDMSG1**, **SBRMI_INBNDMSG2**, **SBRMI_INBNDMSG3**,
  SBRMI_INBNDMSG4, **SBRMI_INBNDMSG5**, **SBRMI_INBNDMSG6**, **SBRMI_INBNDMSG7** }

  SBRMI inbound messages defined in the APML library.

Functions

- oob_status_t esmi_oob_read_byte (uint8_t soc_num, uint8_t reg_offset, char ∗file_name, uint8_t ∗buffer)

  Reads data for the given register.
- oob_status_t esmi_oob_write_byte (uint8_t soc_num, uint8_t reg_offset, char ∗file_name, uint8_t value)

  Writes data to the specified register.
- oob_status_t esmi_oob_read_mailbox (uint8_t soc_num, uint32_t cmd, uint32_t input, uint32_t ∗buffer)

  Reads mailbox command data.
- oob_status_t esmi_oob_write_mailbox (uint8_t soc_num, uint32_t cmd, uint32_t data)

  Writes data to the given mailbox command.
- oob_status_t sbrmi_xfer_msg (uint8_t soc_num, char ∗file_name, struct apml_message ∗msg)

  Writes data to device file.
- oob_status_t validate_apml_dependency (uint8_t soc_num, bool ∗is_sbrmi, bool ∗is_sbtsi)

  Validates sbrmi and sbtsi modules are present for the given socket.

Variables

- const uint16_t sbrmi_addr [MAX_DEV_COUNT]

  SBRMI addresses //.
- const uint16_t sbtsi_addr [MAX_DEV_COUNT]

  SBTSI addresses //.

7.1.1 Detailed Description

Main header file for the APML library. All required function, structure, enum, etc. definitions should be defined in this file.

This header file contains the following: APIs prototype of the APIs exported by the APML library. Description of the API, arguments and return values. The Error codes returned by the API.

7.1.2 Enumeration Type Documentation

7.1.2.1 sbrmi_inbnd_msg

```
enum sbrmi_inbnd_msg
```

SBRMI inbound messages defined in the APML library.

Usage convention is: • SBRMI::InBndMsg_inst0 is command • SBRMI::InBndMsg_inst[4:1] are 32 bit data • SBR↩
MI::InBndMsg_inst[6:5] are reserved. • SBRMI::InBndMsg_inst7: [7] Must be 1'b1 to send message to firmware

7.1.3 Function Documentation

7.1.3.1 esmi_oob_read_byte()

```
oob_status_t esmi_oob_read_byte (
             uint8_t soc_num,
             uint8_t reg_offset,
             char * file_name,
             uint8_t * buffer )
```

Reads data for the given register.

This function will read the data for the given register.

Parameters

in	*soc_num*	Socket index.
in	*reg_offset*	Register offset for RMI/TSI I/F.
in	*file_name*	Character device file name for RMI/TSI I/F.
out	*buffer*	output value for the register.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.1.3.2 esmi_oob_write_byte()

```
oob_status_t esmi_oob_write_byte (
             uint8_t soc_num,
             uint8_t reg_offset,
             char * file_name,
             uint8_t value )
```

Writes data to the specified register.

This function will write the data to the specified register.

Parameters

in	*soc_num*	Socket index.
in	*file_name*	Character device file name for RMI/TSI I/F.
in	*reg_offset*	Register offset for RMI/TSI I/F.
in	*value*	data to write to the register.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.1.3.3 esmi_oob_read_mailbox()

```
oob_status_t esmi_oob_read_mailbox (
             uint8_t soc_num,
             uint32_t cmd,
             uint32_t input,
             uint32_t * buffer )
```

Reads mailbox command data.

This function will read mailbox command data.

Parameters

in	*soc_num*	Socket index.
in	*cmd*	mailbox command.
in	*input*	data.
out	*buffer*	output data for the given mailbox command.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.1.3.4 esmi_oob_write_mailbox()

```
oob_status_t esmi_oob_write_mailbox (
             uint8_t soc_num,
             uint32_t cmd,
             uint32_t data )
```

Writes data to the given mailbox command.

This function will writes data to mailbox command.

Parameters

in	*soc_num*	Socket index.
in	*cmd*	mailbox command.
in	*data*	input data.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.1.3.5 sbrmi_xfer_msg()

```
oob_status_t sbrmi_xfer_msg (
             uint8_t soc_num,
             char * file_name,
             struct apml_message * msg )
```

Writes data to device file.

This function will write data to character device file, through ioctl.

Parameters

in	*soc_num*	Socket index.
in	*file_name*	Character device file name for RMI/TSI I/F
in	*msg*	struct apml_message which contains information about the protocol, input/output data etc.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.1.3.6 validate_apml_dependency()

```
oob_status_t validate_apml_dependency (
             uint8_t soc_num,
             bool * is_sbrmi,
             bool * is_sbtsi )
```

Validates sbrmi and sbtsi modules are present for the given socket.

This function will validate sbrmi and sbtsi modules are present for the specified socket.

Parameters

in	*soc_num*	Socket index.
out	*is_sbrmi*	returns true if the sbrmi is present else false
out	*is_sbtsi*	returns true if the sbtsi is present else false

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.2 apml_err.h File Reference

Macros

- #define OOB_CPUID_MSR_ERR_BASE 0x800

    CPUID MSR FW error code.
- #define OOB_MAILBOX_ERR_BASE 0x900

    MAILBOX FW error code.

Enumerations

- enum oob_status_t {
  OOB_SUCCESS = 0, OOB_NOT_FOUND, OOB_PERMISSION, OOB_NOT_SUPPORTED,
  OOB_FILE_ERROR, OOB_INTERRUPTED, OOB_UNEXPECTED_SIZE, OOB_UNKNOWN_ERROR,
  OOB_ARG_PTR_NULL, OOB_NO_MEMORY, OOB_NOT_INITIALIZED, OOB_TRY_AGAIN,
  OOB_INVALID_INPUT, OOB_CMD_TIMEOUT, OOB_INVALID_MSGSIZE, **OOB_CPUID_MSR_ERR_S**↩
  TART,
  OOB_CPUID_MSR_CMD_TIMEOUT, OOB_CPUID_MSR_CMD_WARM_RESET, OOB_CPUID_MSR_CMD_UNKNOWN_FM
  OOB_CPUID_MSR_CMD_INVAL_RD_LEN,
  OOB_CPUID_MSR_CMD_EXCESS_DATA_LEN, OOB_CPUID_MSR_CMD_INVAL_THREAD, OOB_CPUID_MSR_CMD_UN
  OOB_CPUID_MSR_CMD_ABORTED,
  OOB_CPUID_MSR_ERR_END, **OOB_MAILBOX_ERR_START**, OOB_MAILBOX_CMD_ABORTED,
  OOB_MAILBOX_CMD_UNKNOWN,
  OOB_MAILBOX_CMD_INVAL_CORE, **OOB_MAILBOX_ERR_END** }

  Error codes retured by APML_ERR functions.

Functions

- oob_status_t errno_to_oob_status (int err)

  convert linux error to esmi error.
- char ∗ esmi_get_err_msg (oob_status_t oob_err)

  Get the error string message for esmi oob errors.

7.2.1 Detailed Description

Header file for the APML library error/return codes.

This header file has error/return codes for the API.

7.2.2 Enumeration Type Documentation

7.2.2.1 oob_status_t

```
enum oob_status_t
```

Error codes retured by APML_ERR functions.

Enumerator

OOB_SUCCESS	Operation was successful.
OOB_NOT_FOUND	An item was searched for but not found.
OOB_PERMISSION	many functions require root access to run. Permission denied/EACCESS file error.
OOB_NOT_SUPPORTED	The requested information or action is not available for the given input, on the given system

Enumerator

OOB_FILE_ERROR	Problem accessing a file. This may because the operation is not supported by the Linux kernel version running on the executing machine
OOB_INTERRUPTED	execution of function An interrupt occurred during
OOB_UNEXPECTED_SIZE	was read An unexpected amount of data
OOB_UNKNOWN_ERROR	An unknown error occurred.
OOB_ARG_PTR_NULL	Parsed argument ptr null.
OOB_NO_MEMORY	Not enough memory to allocate.
OOB_NOT_INITIALIZED	APML object not initialized.
OOB_TRY_AGAIN	No match Try again.
OOB_INVALID_INPUT	Input value is invalid.
OOB_CMD_TIMEOUT	Command timed out.
OOB_INVALID_MSGSIZE	Mesg size too long.
OOB_CPUID_MSR_CMD_TIMEOUT	RMI cmd timeout.
OOB_CPUID_MSR_CMD_WARM_RESET	Warm reset during RMI cmd.
OOB_CPUID_MSR_CMD_UNKNOWN_FMT	Cmd fmt field not recongnised.
OOB_CPUID_MSR_CMD_INVAL_RD_LEN	RMI cmd invalid read len.
OOB_CPUID_MSR_CMD_EXCESS_DATA_LEN	excess data
OOB_CPUID_MSR_CMD_INVAL_THREAD	Invalid thread selected.
OOB_CPUID_MSR_CMD_UNSUPP	Cmd not supported.
OOB_CPUID_MSR_CMD_ABORTED	Cmd aborted.
OOB_MAILBOX_CMD_ABORTED	Mailbox cmd aborted.
OOB_MAILBOX_CMD_UNKNOWN	Unknown mailbox cmd.
OOB_MAILBOX_CMD_INVAL_CORE	Invalid core.

7.3 apml_recovery.h File Reference

Enumerations

- enum apml_client { **DEV_SBRMI** = 0x0, **DEV_SBTSI** }

  APML Client Devices.

Functions

- oob_status_t apml_recover_dev (uint8_t soc_num, uint8_t client)

  Recover the APML client device for the given socket.

7.3.1 Detailed Description

Header file for the APML recovery flow

7.3.2 Function Documentation

7.3.2.1 apml_recover_dev()

```
oob_status_t apml_recover_dev (
             uint8_t soc_num,
             uint8_t client )
```

Recover the APML client device for the given socket.

This function will recover the APML client device and returns successful on recovery or error if recovery is unsuccessful

NOTE: This fix is a software workaround for the erratum, Erratum:1444, "Advanced Platform Management Link (APML) May Cease to Function After Incomplete Read Transaction" Further details can be found in mentioned tech doc `https://www.amd.com/system/files/TechDocs/57095-PUB_1.00.pdf`

Parameters

in	*soc_num*	Socket index.
in	*client*	DEV_SBRMI[0]/DEV_SBTSI[1] enum: apml_client

Return values

OOB_SUCCESS	is returned upon successful recovery
Non-zero	is returned upon failure.

7.4 esmi_cpuid_msr.h File Reference

```
#include "apml_err.h"
```

Data Structures

- struct processor_info
  Read Proccessor Info.

Enumerations

- enum cpuid_reg { **EAX** = 0, **EBX**, **ECX**, **EDX** }

Functions

- oob_status_t esmi_get_vendor_id (uint8_t soc_num, char ∗vendor_id)
  Get the number of logical cores per socket.
- oob_status_t esmi_get_processor_info (uint8_t soc_num, struct processor_info ∗proc_info)
  Get the number of logical cores per socket.
- oob_status_t esmi_get_logical_cores_per_socket (uint8_t soc_num, uint32_t ∗logical_cores_per_socket)
  Get the number of logical cores per socket.

- oob_status_t esmi_get_threads_per_socket (uint8_t soc_num, uint32_t ∗threads_per_socket)

  Get the number of threads per socket.
- oob_status_t esmi_get_threads_per_core (uint8_t soc_num, uint32_t ∗threads_per_core)

  Get number of threads per core.
- oob_status_t esmi_oob_read_msr (uint8_t soc_num, uint32_t thread, uint32_t msraddr, uint64_t ∗buffer)

  Read the MCA MSR register for a given thread.
- oob_status_t esmi_oob_cpuid (uint8_t soc_num, uint32_t thread, uint32_t ∗eax, uint32_t ∗ebx, uint32_t ∗ecx, uint32_t ∗edx)

  Read CPUID functionality for a particular thread in a system.
- oob_status_t esmi_oob_cpuid_eax (uint8_t soc_num, uint32_t thread, uint32_t fn_eax, uint32_t fn_ecx, uint32_t ∗eax)

  Read eax register on CPUID functionality.
- oob_status_t esmi_oob_cpuid_ebx (uint8_t soc_num, uint32_t thread, uint32_t fn_eax, uint32_t fn_ecx, uint32_t ∗ebx)

  Read ebx register on CPUID functionality.
- oob_status_t esmi_oob_cpuid_ecx (uint8_t soc_num, uint32_t thread, uint32_t fn_eax, uint32_t fn_ecx, uint32_t ∗ecx)

  Read ecx register on CPUID functionality.
- oob_status_t esmi_oob_cpuid_edx (uint8_t soc_num, uint32_t thread, uint32_t fn_eax, uint32_t fn_ecx, uint32_t ∗edx)

  Read edx register on CPUID functionality.
- oob_status_t read_max_threads_per_l3 (uint8_t soc_num, uint32_t ∗threads_l3)

  Read max threads per L3 cache.

Variables

- struct processor_info plat_info [1]

  Platform Info instance.

7.4.1 Detailed Description

Header file for the APML library cpuid and msr read functions. All required function, structure, enum and protocol specific data etc. definitions should be defined in this header.

This header file contains the following: APIs prototype of the APIs exported by the APML library. Description of the API, arguments and return values. The Error codes returned by the API.

7.4.2 Enumeration Type Documentation

7.4.2.1 cpuid_reg

```
enum cpuid_reg
```

CPUID register indexes 0 for EAX, 1 for EBX, 2 ECX and 3 for EDX

7.5 esmi_mailbox.h File Reference

```
#include "apml_common.h"
#include "apml_err.h"
#include <stdbool.h>
```

Data Structures

- struct dimm_power

  DIMM power(mW), update rate(ms) and dimm address.
- struct dimm_thermal

  DIMM thermal sensor (degree C), update rate and dimm address.
- struct temp_refresh_rate

  DIMM temperature range and refresh rate, temperature update flag.
- struct pci_address

  PCI address information .PCI address includes 4 bit segment, 12 bit aligned offset, 8 bit bus, 5 bit device info and 3 bit function.
- struct dpm_level

  Max and min LCK DPM level on a given NBIO ID. Valid Max and min DPM level values are 0 - 1.
- struct lclk_dpm_level_range

  Max and Min Link frequency clock (LCLK) DPM level on a socket. 8 bit NBIO ID, dpm_level struct containing 8 bit max DPM level, 8 bit min DPM level.
- struct nbio_err_log

  NBIO quadrant(8 bit data) and NBIO register offset(24 bit) data.
- struct max_ddr_bw

  Structure for Max DDR bandwidth and utilization. It contains max bandwidth(12 bit data) in GBps, current utilization bandwidth(12 bit data) in GBps, current utilized bandwidth(8 bit data) in percentage.
- struct mca_bank

  MCA bank information.It contains 16 bit Index for MCA Bank and 16 bit offset.
- struct link_id_bw_type

  APML LINK ID and Bandwidth type Information.It contains APML LINK ID Encoding. Non-MI300 Platforms Valid Link ID encodings are 1(P0), 2(P1), 4(P2), 8(P3), 16(G0), 32(G1), 64(G2), 128(G3). Valid APML MI300 APML LINK ID Encoding. Valid Link ID encodings are 3(P2), 4(P3), 8(G0), 9(G1), 10(G2), 11(G3), 12(G4), 13(G5), 14(G6), 15(G7). IO Bandwidth types 1(Aggregate_BW), 2 (Read BW), 4 (Write BW).
- struct pstate_freq

  DF P-state frequency.It includes mem clock(16 bit data) frequency (DRAM memory clock), data fabric clock (12 bit data), UMC clock divider (UMC) (1 bit data).
- struct ras_df_err_chk

  RAS df err validity check output status. Structure contains the following members. df_block_instances number of block instance with error log to report (0 - 256) err_log_len length of error log in bytes per instance (0 - 256).
- union ras_df_err_dump

  RAS df error dump input. Union contains the following members. input[0] 4 byte alligned offset in error log (0 - 255) input[1] DF block ID (0 - 255) input[2] Zero based index of DF block instance (0 - 255) input[3] Reserved data_in 32-bit data input.
- struct ras_override_delay

  BMC RAS override delay reset CPU on sync flood. The structure contains delay value override in mins [5 -120 mins], disable delay counter and stop delay counter. If disable delay counter is set ResetCpuOnSyncFlood response will NOT be delayed in the next SyncFlood regardless of the value specified in delay_val_override. If StopDelayCounter is set it stops the active delay countdown which extends the DelayResetCpuOnSyncFlood indefinitely and system will not reset.

Macros

- #define MAX_ERR_LOG_LEN 256

  Max error log length //.
- #define MAX_DF_BLOCK_IDS 256

  Max DF block IDs //.
- #define MAX_DF_BLOCK_INSTS 256

  Max DF block instances //.

Enumerations

- enum esb_mailbox_commmands {
  READ_PACKAGE_POWER_CONSUMPTION = 0x1, **WRITE_PACKAGE_POWER_LIMIT**, **READ_PAC←KAGE_POWER_LIMIT**, **READ_MAX_PACKAGE_POWER_LIMIT**,
  READ_TDP, **READ_MAX_cTDP**, **READ_MIN_cTDP**, **READ_BIOS_BOOST_Fmax**,
  READ_APML_BOOST_LIMIT, **WRITE_APML_BOOST_LIMIT**, **WRITE_APML_BOOST_LIMIT_ALLCO←RES**, **READ_DRAM_THROTTLE**,
  WRITE_DRAM_THROTTLE, **READ_PROCHOT_STATUS**, **READ_PROCHOT_RESIDENCY**, **READ_N←BIO_ERROR_LOGGING_REGISTER** = 0x11,
  READ_IOD_BIST = 0x13, **READ_CCD_BIST_RESULT**, **READ_CCX_BIST_RESULT**, **READ_PACKAG←E_CCLK_FREQ_LIMIT**,
  READ_PACKAGE_C0_RESIDENCY, **READ_DDR_BANDWIDTH** = 0x18, **READ_PPIN_FUSE** = 0x1F, **G←ET_POST_CODE** = 0x20,
  WRITE_BMC_REPORT_DIMM_POWER = 0X40, **WRITE_BMC_REPORT_DIMM_THERMAL_SENSOR**, **READ_BMC_RAS_PCIE_CONFIG_ACCESS**, **READ_BMC_RAS_MCA_VALIDITY_CHECK**,
  READ_BMC_RAS_MCA_MSR_DUMP, **READ_BMC_RAS_FCH_RESET_REASON**, **READ_DIMM_TE←MP_RANGE_AND_REFRESH_RATE**, **READ_DIMM_POWER_CONSUMPTION**,
  READ_DIMM_THERMAL_SENSOR, **READ_PWR_CURRENT_ACTIVE_FREQ_LIMIT_SOCKET**, **READ←_PWR_CURRENT_ACTIVE_FREQ_LIMIT_CORE**, **READ_PWR_SVI_TELEMETRY_ALL_RAILS**,
  READ_SOCKET_FREQ_RANGE, **READ_CURRENT_IO_BANDWIDTH**, **READ_CURRENT_XGMI_BA←NDWIDTH**, **WRITE_GMI3_LINK_WIDTH_RANGE**,
  WRITE_XGMI_LINK_WIDTH_RANGE, **WRITE_APB_DISABLE**, **WRITE_APB_ENABLE**, **READ_CURR←ENT_DFPSTATE_FREQUENCY**,
  WRITE_LCLK_DPM_LEVEL_RANGE, **READ_BMC_RAPL_UNITS**, **READ_BMC_RAPL_CORE_LO_C←OUNTER**, **READ_BMC_RAPL_CORE_HI_COUNTER**,
  READ_BMC_RAPL_PKG_COUNTER, **READ_BMC_CPU_BASE_FREQUENCY**, **READ_BMC_CONTR←OL_PCIE_GEN5_RATE**, **READ_RAS_LAST_TRANS_ADDR_CHK**,
  READ_RAS_LAST_TRANS_ADDR_DUMP, **WRITE_PWR_EFFICIENCY_MODE**, **WRITE_DF_PSTATE←_RANGE**, **READ_LCLK_DPM_LEVEL_RANGE**,
  READ_UCODE_REVISION, **BMC_RAS_DELAY_RESET_ON_SYNCFLOOD_OVERRIDE** = 0x6A, **REA←D_BMC_RAS_RESET_ON_SYNC_FLOOD** }

  Mailbox message types defined in the APML library.
- enum apml_io_bw_encoding { **AGG_BW** = BIT(0), **RD_BW** = BIT(1), **WR_BW** = BIT(2) }

  APML IO Bandwidth Encoding defined in the APML library.
- enum apml_link_id_encoding {
  P0 = BIT(0), **P1** = BIT(1), **P2** = BIT(2), **P3** = BIT(3),
  G0 = BIT(4), **G1** = BIT(5), **G2** = BIT(6), **G3** = BIT(7) }

  APML IO LINK ID Encoding defined in the APML library.

Functions

- oob_status_t read_socket_power (uint8_t soc_num, uint32_t ∗buffer)

  Get the power consumption of the socket.
- oob_status_t read_socket_power_limit (uint8_t soc_num, uint32_t ∗buffer)

Get the current power cap/limit value for a given socket.
- oob_status_t read_max_socket_power_limit (uint8_t soc_num, uint32_t ∗buffer)

  Get the maximum value that can be assigned as a power cap/limit for a given socket.
- oob_status_t write_socket_power_limit (uint8_t soc_num, uint32_t limit)

  Set the power cap/limit value for a given socket.
- oob_status_t read_esb_boost_limit (uint8_t soc_num, uint32_t value, uint32_t ∗buffer)

  Get the Out-of-band boostlimit value for a given core.
- oob_status_t read_bios_boost_fmax (uint8_t soc_num, uint32_t value, uint32_t ∗buffer)

  Get the In-band maximum boostlimit value for a given core.
- oob_status_t write_esb_boost_limit (uint8_t soc_num, uint32_t cpu_ind, uint32_t limit)

  Set the Out-of-band boostlimit value for a given core.
- oob_status_t write_esb_boost_limit_allcores (uint8_t soc_num, uint32_t limit)

  Set the boostlimit value for the whole socket (whole system).
- oob_status_t read_tdp (uint8_t soc_num, uint32_t ∗buffer)

  Get the Thermal Design Power limit TDP of the socket with provided socket index.
- oob_status_t read_max_tdp (uint8_t soc_num, uint32_t ∗buffer)

  Get the Maximum Thermal Design Power limit TDP of the socket with provided socket index.
- oob_status_t read_min_tdp (uint8_t soc_num, uint32_t ∗buffer)

  Get the Minimum Thermal Design Power limit TDP of the socket.
- oob_status_t read_prochot_status (uint8_t soc_num, uint32_t ∗buffer)

  Get the Prochot Status of the socket with provided socket index.
- oob_status_t read_prochot_residency (uint8_t soc_num, float ∗buffer)

  Get the Prochot Residency (since the boot time or last read of Prochot Residency) of the socket.
- oob_status_t read_dram_throttle (uint8_t soc_num, uint32_t ∗buffer)

  Read Dram Throttle will always read the highest percentage value.
- oob_status_t write_dram_throttle (uint8_t soc_num, uint32_t limit)

  Set Dram Throttle value in terms of percentage.
- oob_status_t read_nbio_error_logging_register (uint8_t soc_num, struct nbio_err_log nbio, uint32_t ∗buffer)

  Read NBIO Error Logging Register.
- oob_status_t read_iod_bist (uint8_t soc_num, uint32_t ∗buffer)

  Read IOD Bist status.
- oob_status_t read_ccd_bist_result (uint8_t soc_num, uint32_t input, uint32_t ∗buffer)

  Read CCD Bist status. Results are read for each CCD present in the system.
- oob_status_t read_ccx_bist_result (uint8_t soc_num, uint32_t value, uint32_t ∗ccx_bist)

  Read CPU Core Complex Bist result. results are read for each Logical CCX instance number and returns a value which is the concatenation of L3 pass status and all cores in the complex(n:0).
- oob_status_t read_cclk_freq_limit (uint8_t soc_num, uint32_t ∗cclk_freq)

  Read CCLK frequency limit for the given socket.
- oob_status_t read_socket_c0_residency (uint8_t soc_num, uint32_t ∗c0_res)

  Read socket C0 residency.
- oob_status_t read_ddr_bandwidth (uint8_t soc_num, struct max_ddr_bw ∗max_ddr)

  Get the Theoretical maximum DDR Bandwidth of the system in GB/s, Current utilized DDR Bandwidth (Read + Write) in GB/s and Current utilized DDR Bandwidth as a percentage of theoretical maximum.
- oob_status_t write_bmc_report_dimm_power (uint8_t soc_num, struct dimm_power dp_info)

  Set DIMM Power consumption in mwatts.
- oob_status_t write_bmc_report_dimm_thermal_sensor (uint8_t soc_num, struct dimm_thermal dt_info)

  Set DIMM thermal Sensor in degree Celcius.
- oob_status_t read_bmc_ras_pcie_config_access (uint8_t soc_num, struct pci_address pci_addr, uint32_↩
t ∗out_buf)

  Read BMC RAS PCIE config access.

- oob_status_t read_bmc_ras_mca_validity_check (uint8_t soc_num, uint16_t ∗bytes_per_mca, uint16_t ∗mca_banks)

    Read number of MCA banks with valid status after a fatal error.
- oob_status_t read_bmc_ras_mca_msr_dump (uint8_t soc_num, struct mca_bank mca_dump, uint32_t ∗out_buf)

    Read data from mca bank reported by bmc ras mca validity check.
- oob_status_t read_bmc_ras_fch_reset_reason (uint8_t soc_num, uint32_t input, uint32_t ∗out_buf)

    Read FCH reason code from the previous reset.
- oob_status_t read_dimm_temp_range_and_refresh_rate (uint8_t soc_num, uint32_t dimm_addr, struct temp_refresh_rate ∗rate)

    Read DIMM temperature range and refresh rate.
- oob_status_t read_dimm_power_consumption (uint8_t soc_num, uint32_t dimm_addr, struct dimm_power ∗dimm_pow)

    Read DIMM power consumption.
- oob_status_t read_dimm_thermal_sensor (uint8_t soc_num, uint32_t dimm_addr, struct dimm_thermal ∗dimm_temp)

    Read DIMM thermal sensor.
- oob_status_t read_pwr_current_active_freq_limit_socket (uint8_t soc_num, uint16_t ∗freq, char ∗∗source_type)

    Read current active frequency limit per socket.
- oob_status_t read_pwr_current_active_freq_limit_core (uint8_t soc_num, uint32_t core_id, uint16_t ∗base_freq)

    Read current active frequency limit set per core.
- oob_status_t read_pwr_svi_telemetry_all_rails (uint8_t soc_num, uint32_t ∗power)

    Read SVR based telemtry for all rails.
- oob_status_t read_socket_freq_range (uint8_t soc_num, uint16_t ∗fmax, uint16_t ∗fmin)

    Read socket frequency range.
- oob_status_t read_current_io_bandwidth (uint8_t soc_num, struct link_id_bw_type link, uint32_t ∗io_bw)

    Read current bandwidth on IO Link.
- oob_status_t read_current_xgmi_bandwidth (uint8_t soc_num, struct link_id_bw_type link, uint32_t ∗xgmi_bw)

    Read current bandwidth on xGMI Link.
- oob_status_t write_gmi3_link_width_range (uint8_t soc_num, uint8_t min_link_width, uint8_t max_link_width)

    Set the max and min width of GMI3 link.
- oob_status_t write_xgmi_link_width_range (uint8_t soc_num, uint8_t min_link_width, uint8_t max_link_width)

    Set the max and min width of xGMI link.
- oob_status_t write_apb_disable (uint8_t soc_num, uint8_t df_pstate, bool ∗prochot_asserted)

    Set the APBDisabled.
- oob_status_t write_apb_enable (uint8_t soc_num, bool ∗prochot_asserted)

    Enable the DF p-state performance boost algorithm.
- oob_status_t read_current_dfpstate_frequency (uint8_t soc_num, struct pstate_freq ∗df_pstate)

    Read current DF p-state frequency .
- oob_status_t write_lclk_dpm_level_range (uint8_t soc_num, struct lclk_dpm_level_range lclk)

    Set the max and min LCK DPM Level on a given NBIO per socket.
- oob_status_t read_bmc_rapl_units (uint8_t soc_num, uint8_t ∗tu_value, uint8_t ∗esu_value)

    Read RAPL (Running Average Power Limit) Units.
- oob_status_t read_bmc_cpu_base_frequency (uint8_t soc_num, uint16_t ∗base_freq)

    Read RAPL base frequency per CPU socket.
- oob_status_t read_bmc_control_pcie_gen5_rate (uint8_t soc_num, uint8_t rate, uint8_t ∗mode)

    Control PCIe Rate on Gen5-Capable devices..
- oob_status_t read_rapl_core_energy_counters (uint8_t soc_num, uint32_t core_id, double ∗energy_counters)

Read RAPL core energy counters.
- oob_status_t read_rapl_pckg_energy_counters (uint8_t soc_num, double ∗energy_counters)

  Read RAPL package energy counters.
- oob_status_t write_pwr_efficiency_mode (uint8_t soc_num, uint8_t mode)

  Write power efficiency profile policy.
- oob_status_t write_df_pstate_range (uint8_t soc_num, uint8_t max_pstate, uint8_t min_pstate)

  Write df pstate range.
- oob_status_t read_lclk_dpm_level_range (uint8_t soc_num, uint8_t nbio_id, struct dpm_level ∗dpm)

  Read LCLK Max and Min DPM level range.
- oob_status_t read_ucode_revision (uint8_t soc_num, uint32_t ∗ucode_rev)

  Read ucode revision.
- oob_status_t read_ras_df_err_validity_check (uint8_t soc_num, uint8_t df_block_id, struct ras_df_err_chk ∗err_chk)

  Read number of instances of DF blocks of type DF_BLOCK_ID with errors.
- oob_status_t read_ras_df_err_dump (uint8_t soc_num, union ras_df_err_dump ras_err, uint32_t ∗data)

  Read RAS DF error dump.
- oob_status_t reset_on_sync_flood (uint8_t soc_num, uint32_t ∗ack_resp)

  Read BMC RAS reset on sync flood.
- oob_status_t override_delay_reset_on_sync_flood (uint8_t soc_num, struct ras_override_delay data_in, bool ∗ack_resp)

  Overrides delay reset cpu on sync flood value.
- oob_status_t get_post_code (uint8_t soc_num, uint32_t offset, uint32_t ∗post_code)

  Read post code.
- oob_status_t read_ppin_fuse (uint8_t soc_num, uint64_t ∗data)

  Get the 64 bit PPIN fuse.

Variables

- float esu_multiplier

  energy status multiplier value is 1/2^ ESU where ESU is [12:8] bit of the mailbox command 0x55h.

7.5.1 Detailed Description

Header file for the Mailbox messages supported by APML library. All required function, structure, enum, etc. definitions should be defined in this file.

This header file contains the following: APIs prototype of the Mailbox messages exported by the APML library. Description of the API, arguments and return values. The Error codes returned by the API.

7.5.2 Function Documentation

7.5.2.1 write_bmc_report_dimm_power()

```
oob_status_t write_bmc_report_dimm_power (
            uint8_t soc_num,
            struct dimm_power dp_info )
```

Set DIMM Power consumption in mwatts.

This function will set DIMM Power consumption periodically by BMC at specified update rate (10 ms or less) when bmc owns the SPD side-band bus.

Parameters

in	*soc_num*	Socket index.
in	*dp_info*	dimm_power Struct with power(mw), updaterate(ms) & dimm address

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.2 write_bmc_report_dimm_thermal_sensor()

```
oob_status_t write_bmc_report_dimm_thermal_sensor (
             uint8_t soc_num,
             struct dimm_thermal dt_info )
```

Set DIMM thermal Sensor in degree Celcius.

This function will set DIMM thermal sensor (in degree celcius) periodically by BMC at specified update rate (10 ms or less) when bmc owns the SPD side-band bus.

Parameters

in	*soc_num*	Socket index.
in	*dt_info*	struct with temp(ºC), updaterate(ms) & dimm address

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.3 read_bmc_ras_pcie_config_access()

```
oob_status_t read_bmc_ras_pcie_config_access (
             uint8_t soc_num,
             struct pci_address pci_addr,
             uint32_t * out_buf )
```

Read BMC RAS PCIE config access.

This function will read the 32 bit BMC RAS extended PCI config space.

Parameters

in	*soc_num*	Socket index.

Parameters

in	*pci_addr*	pci_address structure with fucntion(3 bit), device(4 bit) bus(8 bit), offset(12 bit), segment(4 bit). SEGMENT:0 BUS 0:DEVICE 18 and SEGMENT:0 BUS 0:DEVICE 19 are inaccessable.
out	*out_buf*	32 bit data from offset in PCI config space.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.4 read_bmc_ras_mca_validity_check()

```
oob_status_t read_bmc_ras_mca_validity_check (
             uint8_t soc_num,
             uint16_t * bytes_per_mca,
             uint16_t * mca_banks )
```

Read number of MCA banks with valid status after a fatal error.

This function returns the number of MCA banks with valid status after a fatal error.

Parameters

in	*soc_num*	Socket index.
out	*bytes_per_mca*	returns bytes per mca.
out	*mca_banks*	number of mca banks.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.5 read_bmc_ras_mca_msr_dump()

```
oob_status_t read_bmc_ras_mca_msr_dump (
             uint8_t soc_num,
             struct mca_bank mca_dump,
             uint32_t * out_buf )
```

Read data from mca bank reported by bmc ras mca validity check.

This function returns the data from mca bank reported by bmc ras mca validity check.

Parameters

in	*soc_num*	Socket index.
in	*mca_dump*	mca_bank Struct containing offset, index of MCA bank.
out	*out_buf*	32 bit data from offset in mca bank.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.6 read_bmc_ras_fch_reset_reason()

```
oob_status_t read_bmc_ras_fch_reset_reason (
              uint8_t soc_num,
              uint32_t input,
              uint32_t * out_buf )
```

Read FCH reason code from the previous reset.

This function reads the FCH reason code from the previous reset.

Parameters

in	*soc_num*	Socket index.
in	*input*	integer for id of FCH register.
out	*out_buf*	Data from FCH register.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.7 read_dimm_temp_range_and_refresh_rate()

```
oob_status_t read_dimm_temp_range_and_refresh_rate (
              uint8_t soc_num,
              uint32_t dimm_addr,
              struct temp_refresh_rate * rate )
```

Read DIMM temperature range and refresh rate.

This function returns the per DIMM temperature range and refresh rate from the MR4 register, per JEDEC spec.

Parameters

in	*soc_num*	Socket index.
in	*dimm_addr*	Encoded address of the dimm.
out	*rate*	temp_refresh_rate structure with refresh rate(1 bit) and range(3 bit). refresh rate: 0 = 1X, 1 = 2X. Temperature range: 001b = 1X, 101b = 2X.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.8 read_dimm_power_consumption()

```
oob_status_t read_dimm_power_consumption (
             uint8_t soc_num,
             uint32_t dimm_addr,
             struct dimm_power * dimm_pow )
```

Read DIMM power consumption.

This function returns the DIMM power consumption when bmc does not own the SPD side band bus.

Parameters

in	*soc_num*	Socket index.
in	*dimm_addr*	Encoded address of the dimm.
out	*dimm_pow*	struct dimm_power contains updaterate(ms): Time since last update (0-511ms). 0 means last update was $<$ 1ms, and 511 means update was $>=$ 511ms power consumption(mw): power consumed (0 - 32767 mW)

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.9 read_dimm_thermal_sensor()

```
oob_status_t read_dimm_thermal_sensor (
             uint8_t soc_num,
             uint32_t dimm_addr,
             struct dimm_thermal * dimm_temp )
```

Read DIMM thermal sensor.

This function returns the DIMM thermal sensor (2 sensors per DIMM) when bmc does not own the SPD side band bus.

Parameters

in	*soc_num*	Socket index.
in	*dimm_addr*	Encoded address of the dimm.
out	*dimm_temp*	struct dimm_thermal struct contains updaterate(ms): Time since last update (0-511ms). 0 means last update was < 1ms, and 511 means update was >= 511ms temperature (Degrees C): Temperature (-256 - 255.75 degree C)

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.10 read_pwr_current_active_freq_limit_socket()

```
oob_status_t read_pwr_current_active_freq_limit_socket (
             uint8_t soc_num,
             uint16_t * freq,
             char ** source_type )
```

Read current active frequency limit per socket.

This function returns the current active frequency limit per socket.

Parameters

in	*soc_num*	Socket index.
out	*freq*	Frequency (MHz).
out	*source_type*	Source of limit.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.11 read_pwr_current_active_freq_limit_core()

```
oob_status_t read_pwr_current_active_freq_limit_core (
             uint8_t soc_num,
             uint32_t core_id,
             uint16_t * base_freq )
```

Read current active frequency limit set per core.

This function returns the current active frequency limit per core.

Parameters

in	*soc_num*	Socket index.
in	*core_id*	index.
out	*base_freq*	Frequency (MHz).

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.12 read_pwr_svi_telemetry_all_rails()

```
oob_status_t read_pwr_svi_telemetry_all_rails (
            uint8_t soc_num,
            uint32_t * power )
```

Read SVR based telemtry for all rails.

This function returns the SVR based telemetry (power and update rate) for all rails.

Parameters

in	*soc_num*	Socket index.
out	*power*	SVI-based Telemetry for all rails(mW)

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.13 read_socket_freq_range()

```
oob_status_t read_socket_freq_range (
            uint8_t soc_num,
            uint16_t * fmax,
            uint16_t * fmin )
```

Read socket frequency range.

This function returns the fmax and fmin frequency per socket.

Parameters

in	*soc_num*	Socket index.
out	*fmax*	maximum frequency (MHz).
out	*fmin*	minimum frequency (MHz).

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.14 read_current_io_bandwidth()

```
oob_status_t read_current_io_bandwidth (
              uint8_t soc_num,
              struct link_id_bw_type link,
              uint32_t * io_bw )
```

Read current bandwidth on IO Link.

This function returns the current IO bandwidth.

Parameters

in	*soc_num*	Socket index.
in	*link*	link_id_bw_type struct containing bandwidth type and Link ID encoding bandwidth type: 001b Aggregate BW Other Reserved MI300A APML Link ID Encoding: 00000011b: P2 00000100b: P3 00001000b: G0 00001001b: G1 00001010b: G2 00001011b: G3 00001100b: G4 00001101b: G5 00001110b: G6 00001111b: G7 For other platforms the APML Link ID Encoding: 00000001b: P0 00000010b: P1 00000100b: P2 00001000b: P3 00010000b: G0 00100000b: G1 01000000b: G2 10000000b: G3
out	*io_bw*	io bandwidth (Mbps).

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.15 read_current_xgmi_bandwidth()

```
oob_status_t read_current_xgmi_bandwidth (
              uint8_t soc_num,
              struct link_id_bw_type link,
              uint32_t * xgmi_bw )
```

Read current bandwidth on xGMI Link.

This function returns the current xGMI bandwidth.

Parameters

in	*soc_num*	Socket index.

Parameters

in	*link*	link_id_bw_type struct containing link id and bandwidth type info. Valid BW type are 001b Aggregate BW 010b Read BW 100b Write BW Other Reserved MI300A APML Link ID Encoding: 00000011b: P2 00000100b: P3 00001000b: G0 00001001b: G1 00001010b: G2 00001011b: G3 00001100b: G4 00001101b: G5 00001110b: G6 00001111b: G7 For other platforms the APML Link ID Encoding: 00000001b: P0 00000010b: P1 00000100b: P2 00001000b: P3 00010000b: G0 00100000b: G1 01000000b: G2 10000000b: G3
out	*xgmi_bw*	io bandwidth (Mbps).

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.16 write_gmi3_link_width_range()

```
oob_status_t write_gmi3_link_width_range (
            uint8_t soc_num,
            uint8_t min_link_width,
            uint8_t max_link_width )
```

Set the max and min width of GMI3 link.

This function will set the max and min width of GMI3 Link.

Parameters

in	*soc_num*	Socket index.
in	*min_link_width*	minimum link width. 0 = Quarter width 1 = Half width 2 = full width
in	*max_link_width*	maximum link width. 0 = Quarter width 1 = Half width 2 = full width NOTE: max value must be greater than or equal to min value.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.17 write_xgmi_link_width_range()

```
oob_status_t write_xgmi_link_width_range (
            uint8_t soc_num,
            uint8_t min_link_width,
            uint8_t max_link_width )
```

Set the max and min width of xGMI link.

This function will set the max and min width of xGMI Link. If this API is called from both the master and the slave sockets, then the largest width values from either calls are used.

Parameters

in	*soc_num*	Socket index.
in	*min_link_width*	minimum link width. 0 = X4 1 = X8 2 = X16
in	*max_link_width*	maximum link width. 0 = X4 1 = X8 2 = X16 NOTE: Max value must be greater than or equal to min value.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.18 write_apb_disable()

```
oob_status_t write_apb_disable (
            uint8_t soc_num,
            uint8_t df_pstate,
            bool * prochot_asserted )
```

Set the APBDisabled.

This function will set the APBDisabled by specifying the Data Fabric(DF) P-state. Messages APBEnable and A←PBDisable specify DF(Data Fabric) P-state behavior. DF P-states specify the frequency of clock domains from the CPU core boundary through to and including system memory, where 0 is the highest DF P-state and 2 is the lowest.

Parameters

in	*soc_num*	Socket index.
in	*df_pstate*	data fabric p-state.
out	*prochot_asserted*	prochot asserted status. True indicates asserted False indicates not-asserted.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.19 write_apb_enable()

```
oob_status_t write_apb_enable (
            uint8_t soc_num,
            bool * prochot_asserted )
```

Enable the DF p-state performance boost algorithm.

This function will enable the DF p-state performance boost algorithm.

Parameters

in	*soc_num*	Socket index.
out	*prochot_asserted*	prochot asserted status. True indicates asserted and false indicates not-asserted.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.20 read_current_dfpstate_frequency()

```
oob_status_t read_current_dfpstate_frequency (
            uint8_t soc_num,
            struct pstate_freq * df_pstate )
```

Read current DF p-state frequency .

This function returns the current DF p-state frequency. Returns the Fclck, DRAM memory clock(memclk),umc clock divider for the current socket DF P-state.

Parameters

in	*soc_num*	Socket index.
out	*df_pstate*	struct pstate_freq contains DRAM memory clock(mem clk) data fabric clock (Fclk) UMC clock divider Uclk = 0 means divide by 1 else divide by 2.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.21 write_lclk_dpm_level_range()

```
oob_status_t write_lclk_dpm_level_range (
            uint8_t soc_num,
            struct lclk_dpm_level_range lclk )
```

Set the max and min LCK DPM Level on a given NBIO per socket.

This function will set the LCK DPM Level on a given NBIO per socket.The DPM Level is an encoding to represent the PCIE Link Frequency (LCLK) under a root complex (NBIO).

Parameters

in	*soc_num*	Socket index.
in	*lclk*	lclk_dpm_level_range struct containing NBIOID (8 bit) Min dpm level (8 bit) and Max dpm level(8 bit). Valid NBIOID, min dpm level and max dpm level values are between 0 ∼ 3.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.22 read_bmc_rapl_units()

```
oob_status_t read_bmc_rapl_units (
              uint8_t soc_num,
              uint8_t * tu_value,
              uint8_t * esu_value )
```

Read RAPL (Running Average Power Limit) Units.

This function returns the RAPL (Running Average Power Limit) Units. Energy information (in Joules) is based on the multiplier: 1/(2^ESU). Time information (in Seconds) is based on the multiplier: 1/(2^TU).

Parameters

in	*soc_num*	Socket index.
out	*tu_value*	TU value.
out	*esu_value*	esu value.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.23 read_bmc_cpu_base_frequency()

```
oob_status_t read_bmc_cpu_base_frequency (
              uint8_t soc_num,
              uint16_t * base_freq )
```

Read RAPL base frequency per CPU socket.

This function returns the base frequency per CPU socket.

Parameters

in	*soc_num*	Socket index.
out	*base_freq*	base frequency.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.24 read_bmc_control_pcie_gen5_rate()

```
oob_status_t read_bmc_control_pcie_gen5_rate (
            uint8_t soc_num,
            uint8_t rate,
            uint8_t * mode )
```

Control PCIe Rate on Gen5-Capable devices..

This function returns the PCIe rate.

Parameters

in	*soc_num*	Socket index.
in	*rate*	PCIe gen rate. 0 indicates Auto-Detect BW and set link rate accordingly. 1 is for Limit at Gen4 Rate. 2 is for Limit at Gen5 rate.
out	*mode*	previous mode. 0 for Auto-Detect, 1 for Limit at Gen4 rate, 2 for limit at Gen5 rate.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.25 read_rapl_core_energy_counters()

```
oob_status_t read_rapl_core_energy_counters (
            uint8_t soc_num,
            uint32_t core_id,
            double * energy_counters )
```

Read RAPL core energy counters.

This function returns the RAPL core energy counters.

Parameters

in	*soc_num*	Soskcet index.
in	*core_id*	core id.
out	*energy_counters*	core energy.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.26 read_rapl_pckg_energy_counters()

```
oob_status_t read_rapl_pckg_energy_counters (
             uint8_t soc_num,
             double * energy_counters )
```

Read RAPL package energy counters.

This function returns the RAPL package energy counters.

Parameters

in	*soc_num*	Socket index.
out	*energy_counters*	core energy.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.27 write_pwr_efficiency_mode()

```
oob_status_t write_pwr_efficiency_mode (
             uint8_t soc_num,
             uint8_t mode )
```

Write power efficiency profile policy.

This function writes power efficiency mode

Parameters

in	*soc_num*	Socket index.
in	*mode*	power efficiency mode. 0 indicates High Performance Mode 1 indicates Power Efficiency Mode. 2 indicates I/O Performance Mode.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.28 write_df_pstate_range()

```
oob_status_t write_df_pstate_range (
             uint8_t soc_num,
             uint8_t max_pstate,
             uint8_t min_pstate )
```

Write df pstate range.

This function writes df pstate range

Parameters

in	*soc_num*	Socket index.
in	*max_pstate*	value.Max value must be less than or equal to min value. Valid values are 0 - 2.
in	*min_pstate*	value. Valid values are from 0 - 2.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.29 read_lclk_dpm_level_range()

```
oob_status_t read_lclk_dpm_level_range (
             uint8_t soc_num,
             uint8_t nbio_id,
             struct dpm_level * dpm )
```

Read LCLK Max and Min DPM level range.

This function returns the LCLK Max and Min DPM level range.

Parameters

in	*soc_num*	Socket index.
in	*nbio_id*	nbio for a socket.
out	*dpm*	struct dpm level containing max and min dpm levels. Valid max and min dpm levels are from 0 - 1.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.30 read_ucode_revision()

```
oob_status_t read_ucode_revision (
             uint8_t soc_num,
             uint32_t * ucode_rev )
```

Read ucode revision.

This function reads the micro code revision.

Parameters

in	*soc_num*	Socket index.
out	*ucode_rev*	micro code revision.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.31 read_ras_df_err_validity_check()

```
oob_status_t read_ras_df_err_validity_check (
             uint8_t df_block_id,
             uint8_t soc_num,
             struct ras_df_err_chk * err_chk )
```

Read number of instances of DF blocks of type DF_BLOCK_ID with errors.

This function reads the number instance of DF blocks of type DF_BLOCK_ID that have an error log to report.

Parameters

in	*soc_num*	Socket index.
in	*df_block↩_id*	DF block ID.
out	*err_chk*	ras_df_err_chk Struct containing number of DF block instances andclength of error log in bytes per instanace.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.32 read_ras_df_err_dump()

```
oob_status_t read_ras_df_err_dump (
             uint8_t soc_num,
             union ras_df_err_dump ras_err,
             uint32_t * data )
```

Read RAS DF error dump.

This function reads 32 bits of data from the offset provided for DF block instance retported by read_ras_df_err_↩ validity_check.

Parameters

in	*soc_num*	Socket index.
in	*ras_err*	ras_df_err_dump Union containing 4 byte offset, DF block ID and block ID instance.
out	*data*	output data from offset of DF block instance.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.33 reset_on_sync_flood()

```
oob_status_t reset_on_sync_flood (
             uint8_t soc_num,
             uint32_t * ack_resp )
```

Read BMC RAS reset on sync flood.

This function requests reset after sync flood. Reset only works during sync flood condition

Parameters

in	*soc_num*	Socket index. At present, only P0 handles this request.
out	*ack_resp*	acknowledgement response.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.34 override_delay_reset_on_sync_flood()

```
oob_status_t override_delay_reset_on_sync_flood (
            uint8_t soc_num,
            struct ras_override_delay data_in,
            bool * ack_resp )
```

Overrides delay reset cpu on sync flood value.

This function will override delay reset cpu on sync flood value for the current boot instance. Delay value reverts to BIOS config selection after reboot.Number of override requests is limited to 5 per boot instance.

Parameters

in	*soc_num*	Socket index. At present, only P0 handles this request.
in	*data_in*	struct ras_override_delay_d_in containing delay value override [5 - 120 mins], disable delay counter bit and stop delay counter bit.
out	*ack_resp*	acknowledgment response.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.35 get_post_code()

```
oob_status_t get_post_code (
            uint8_t soc_num,
            uint32_t offset,
            uint32_t * post_code )
```

Read post code.

This function will read most recent post code at the specified offset.SMU caches 8 most recent post codes.When the input is 0 the SMU will refresh the cache before running the latest post code. Input as 0 refers to most recent post code and higher inputs refers to the older post code.

Parameters

in	*soc_num*	Socket index.
in	*offset*	post code offset
out	*post_code*	recent post code of error category and number of instances of error category.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.5.2.36 read_ppin_fuse()

```
oob_status_t read_ppin_fuse (
             uint8_t soc_num,
             uint64_t * data )
```

Get the 64 bit PPIN fuse.

This function will read the 64 bit PPIN fuse available via OPN_PPIN fuse.

Parameters

in	*soc_num*	Socket index.
out	*data*	PPIN fuse data

7.6 esmi_rmi.h File Reference

```
#include "apml_err.h"
```

Macros

- #define MAX_ALERT_REG 32

  Max alert register //.
- #define MAX_THREAD_REG_V20 32

  Max thread register for rev 20 //.
- #define MAX_THREAD_REG_V10 16

  Max thread register for rev 10 //.

Enumerations

- enum sbrmi_status_code {
  SBRMI_SUCCESS = 0x0, **SBRMI_CMD_TIMEOUT** = 0x11, **SBRMI_WARM_RESET** = 0x22, **SBRMI_UN↩ KNOWN_CMD_FORMAT** = 0x40,
  SBRMI_INVALID_READ_LENGTH = 0x41, **SBRMI_EXCESSIVE_DATA_LENGTH** = 0x42, **SBRMI_INV↩ ALID_THREAD** = 0x44, **SBRMI_UNSUPPORTED_CMD** = 0x45,
  SBRMI_CMD_ABORTED = 0x81 }

  Error codes retured by APML mailbox functions.

- enum sbrmi_registers {
SBRMI_REVISION = 0x0, **SBRMI_CONTROL**, **SBRMI_STATUS**, **SBRMI_READSIZE**,
SBRMI_THREADENABLESTATUS0, **SBRMI_ALERTSTATUS0** = 0x10, **SBRMI_ALERTSTATUS15** = 0x1F, **SBRMI_ALERTMASK0** = 0x20,
SBRMI_ALERTMASK15 = 0x2F, **SBRMI_SOFTWAREINTERRUPT** = 0x40, **SBRMI_THREADNUMBER**, **SBRMI_THREAD128CS** = 0x4B,
SBRMI_RASSTATUS, **SBRMI_THREADNUMBERLOW** = 0x4E, **SBRMI_THREADNUMBERHIGH** = 0x4F, **SBRMI_ALERTSTATUS16** = 0x50,
SBRMI_ALERTSTATUS31 = 0x5F, **SBRMI_MP0OUTBNDMSG0** = 0x80, **SBRMI_MP0OUTBNDMSG7** = 0x87, **SBRMI_ALERTMASK16** = 0xC0,
SBRMI_ALERTMASK31 = 0xCF }

  SB-RMI(Side-Band Remote Management Interface) features register access.

Functions

- oob_status_t read_sbrmi_revision (uint8_t soc_num, uint8_t ∗buffer)

  Read one byte from a given SB_RMI register number provided socket index and buffer to get the read data for a particular SB-RMI command register.
- oob_status_t read_sbrmi_control (uint8_t soc_num, uint8_t ∗buffer)

  Read Control byte from SB_RMI register command.
- oob_status_t read_sbrmi_status (uint8_t soc_num, uint8_t ∗buffer)

  Read one byte of Status value from SB_RMI register command.
- oob_status_t read_sbrmi_readsize (uint8_t soc_num, uint8_t ∗buffer)

  This register specifies the number of bytes to return when using the block read protocol to read SBRMI_x[4F:10].
- oob_status_t read_sbrmi_threadenablestatus (uint8_t soc_num, uint8_t ∗buffer)

  Read one byte of Thread Status from SB_RMI register command.
- oob_status_t read_sbrmi_multithreadenablestatus (uint8_t soc_num, uint8_t ∗buffer)

  Read one byte of Thread Status from SB_RMI register command.
- oob_status_t read_sbrmi_swinterrupt (uint8_t soc_num, uint8_t ∗buffer)

  This register is used by the SMBus master to generate an interrupt to the processor to indicate that a message is available..
- oob_status_t read_sbrmi_threadnumber (uint8_t soc_num, uint8_t ∗buffer)

  This register indicates the maximum number of threads present.
- oob_status_t read_sbrmi_mp0_msg (uint8_t soc_num, uint8_t ∗buffer)

  This register will read the message running on the MP0.
- oob_status_t read_sbrmi_alert_status (uint8_t soc_num, uint8_t num_of_alert_status_reg, uint8_t ∗∗buffer)

  This function will read bit vector for all the threads. Value of 1 indicates MCE occured for the thread and is set by hardware.
- oob_status_t read_sbrmi_alert_mask (uint8_t soc_num, uint8_t num_of_alert_mask_reg, uint8_t ∗∗buffer)

  This function will read bit vector for all the threads. Value of 1 indicates alert signaling disabled for corresponding SBRMI::AlertStatus[MceStat] for the thread.
- oob_status_t read_sbrmi_inbound_msg (uint8_t soc_num, uint8_t ∗buffer)

  This register will read the inbound message.
- oob_status_t read_sbrmi_outbound_msg (uint8_t soc_num, uint8_t ∗buffer)

  This register will read the outbound message.
- oob_status_t read_sbrmi_threadnumberlow (uint8_t soc_num, uint8_t ∗buffer)

  This register indicates the low part of maximum number of threads.
- oob_status_t read_sbrmi_threadnumberhi (uint8_t soc_num, uint8_t ∗buffer)

  This register indicates the upper part of maximum number of threads.
- oob_status_t read_sbrmi_thread_cs (uint8_t soc_num, uint8_t ∗buffer)

  This register is used to read the thread cs.
- oob_status_t read_sbrmi_ras_status (uint8_t soc_num, uint8_t ∗buffer)

  This register will read the ras status.
- oob_status_t clear_sbrmi_ras_status (uint8_t soc_num, uint8_t buffer)

  This API will clear ras status register.

Variables

- const uint8_t thread_en_reg_v10 [MAX_THREAD_REG_V10]

  thread enable register revision 0x10
- const uint8_t thread_en_reg_v20 [MAX_THREAD_REG_V20]

  thread enable register revision 0x20
- const uint8_t alert_status [MAX_ALERT_REG]

  alert status register
- const uint8_t alert_mask [MAX_ALERT_REG]

  alert mask

7.6.1 Detailed Description

Header file for the APML library for SB-RMI functionality access. All required function, structure, enum, etc. definitions should be defined in this file for SB-RMI Register accessing.

This header file contains the following: APIs prototype of the APIs exported by the APML library. Description of the API, arguments and return values. The Error codes returned by the API.

7.7 esmi_tsi.h File Reference

```
#include "apml_err.h"
```

Macros

- #define TEMP_INC 0.125

  Register encode the temperature to increase in 0.125 In decimal portion one increase in byte is equivalent to 0.125.

Enumerations

- enum sbtsi_registers {
  SBTSI_CPUTEMPINT = 0x1, **SBTSI_STATUS**, **SBTSI_CONFIGURATION**, **SBTSI_UPDATERATE**,
  SBTSI_HITEMPINT = 0x7, **SBTSI_LOTEMPINT**, **SBTSI_CONFIGWR**, **SBTSI_CPUTEMPDEC** = 0x10,
  SBTSI_CPUTEMPOFFINT, **SBTSI_CPUTEMPOFFDEC**, **SBTSI_HITEMPDEC**, **SBTSI_LOTEMPDEC**,
  SBTSI_TIMEOUTCONFIG = 0x22, **SBTSI_ALERTTHRESHOLD** = 0x32, **SBTSI_ALERTCONFIG** = 0xBF,
  SBTSI_MANUFID = 0xFE,
  SBTSI_REVISION = 0xFF }

  SB-TSI(Side-Band Temperature Sensor Interface) commands register access. The below registers mentioned as per Genessis PPR.
- enum sbtsi_config_write { **ARA_MASK** = 0x2, **READORDER_MASK** = 0x20, **RUNSTOP_MASK** = 0x40,
  ALERTMASK_MASK = 0x80 }

  Bitfield values to be set for SBTSI confirwr register [7] Alert mask [6] RunStop [5] ReadOrder [1] AraDis.

Functions

- oob_status_t read_sbtsi_cpuinttemp (uint8_t soc_num, uint8_t ∗buffer)

    integer CPU temperature value The CPU temperature is calculated by adding the CPU temperature offset(SBT←SI::CpuTempOffInt, SBTSI::CpuTempOffDec) to the processor control temperature (Tctl). SBTSI::CpuTempInt and SBTSI::CpuTempDec combine to return the CPU temperature.
- oob_status_t read_sbtsi_status (uint8_t soc_num, uint8_t ∗buffer)

    Status register is Read-only, volatile field If SBTSI::AlertConfig[AlertCompEn] == 0 , the temperature alert is latched high until the alert is read. If SBTSI::AlertConfig[AlertCompEn] == 1, the alert is cleared when the temperature does not meet the threshold conditions for temperature and number of samples.
- oob_status_t read_sbtsi_config (uint8_t soc_num, uint8_t ∗buffer)

    The bits in this register are Read-only and can be written by Writing to the corresponding bits in SBTSI::ConfigWr.
- oob_status_t read_sbtsi_updaterate (uint8_t soc_num, float ∗buffer)

    This register value specifies the rate at which CPU temperature is compared against the temperature thresholds to determine if an alert event has occurred.
- oob_status_t write_sbtsi_updaterate (uint8_t soc_num, float uprate)

    This register value specifies the rate at which CPU temperature is compared against the temperature thresholds to determine if an alert event has occurred.
- oob_status_t read_sbtsi_hitempint (uint8_t soc_num, uint8_t ∗buffer)

    This value specifies the integer portion of the high temperature threshold. The high temperature threshold specifies the CPU temperature that causes ALERT_L to assert if the CPU temperature is greater than or equal to the threshold.
- oob_status_t read_sbtsi_lotempint (uint8_t soc_num, uint8_t ∗buffer)

    This value specifies the integer portion of the low temperature threshold. The low temperature threshold specifies the CPU temperature that causes ALERT_L to assert if the CPU temperature is less than or equal to the threshold.
- oob_status_t read_sbtsi_configwrite (uint8_t soc_num, uint8_t ∗buffer)

    This register provides write access to SBTSI::Config.
- oob_status_t read_sbtsi_cputempdecimal (uint8_t soc_num, float ∗buffer)

    The value returns the decimal portion of the CPU temperature.
- oob_status_t read_sbtsi_cputempoffint (uint8_t soc_num, uint8_t ∗temp_int)

    SBTSI::CpuTempOffInt and SBTSI::CpuTempOffDec combine to specify the CPU temperature offset.
- oob_status_t read_sbtsi_cputempoffdec (uint8_t soc_num, float ∗temp_dec)

    This value specifies the decimal/fractional portion of the CPU temperature offset added to Tctl to calculate the CPU temperature.
- oob_status_t read_sbtsi_hitempdecimal (uint8_t soc_num, float ∗temp_dec)

    This value specifies the decimal portion of the high temperature threshold.
- oob_status_t read_sbtsi_lotempdecimal (uint8_t soc_num, float ∗temp_dec)

    value specifies the decimal portion of the low temperature threshold.
- oob_status_t read_sbtsi_timeoutconfig (uint8_t soc_num, uint8_t ∗timeout)

    value specifies 0=SMBus defined timeout support disabled. 1=SMBus defined timeout support enabled. SMBus timeout enable. If SB-RMI is in use, SMBus timeouts should be enabled or disabled in a consistent manner on both interfaces. SMBus defined timeouts are not disabled for SB-RMI when this bit is set to 0.
- oob_status_t read_sbtsi_alertthreshold (uint8_t soc_num, uint8_t ∗samples)

    Specifies the number of consecutive CPU temperature samples for which a temperature alert condition needs to remain valid before the corresponding alert bit is set.
- oob_status_t read_sbtsi_alertconfig (uint8_t soc_num, uint8_t ∗mode)

    Status register is Read-only, volatile field If SBTSI::AlertConfig[AlertCompEn] == 0 , the temperature alert is latched high until the alert is read. If SBTSI::AlertConfig[AlertCompEn] == 1, the alert is cleared when the temperature does not meet the threshold conditions for temperature and number of samples.
- oob_status_t read_sbtsi_manufid (uint8_t soc_num, uint8_t ∗man_id)

    Returns the AMD manufacture ID.
- oob_status_t read_sbtsi_revision (uint8_t soc_num, uint8_t ∗rivision)

    Specifies the SBI temperature sensor interface revision.
- oob_status_t sbtsi_get_cputemp (uint8_t soc_num, float ∗cpu_temp)

CPU temperature value The CPU temperature is calculated by adding SBTSI::CpuTempInt and SBTSI::CpuTempDec combine to return the CPU temperature.

- oob_status_t sbtsi_get_temp_status (uint8_t soc_num, uint8_t ∗loalert, uint8_t ∗hialert)

  Status register is Read-only, volatile field If SBTSI::AlertConfig[AlertCompEn] == 0 , the temperature alert is latched high until the alert is read. If SBTSI::AlertConfig[AlertCompEn] == 1, the alert is cleared when the temperature does not meet the threshold conditions for temperature and number of samples.
- oob_status_t sbtsi_get_config (uint8_t soc_num, uint8_t ∗al_mask, uint8_t ∗run_stop, uint8_t ∗read_ord, uint8_t ∗ara)

  The bits in this register are Read-only and can be written by Writing to the corresponding bits in SBTSI::ConfigWr.
- oob_status_t sbtsi_set_configwr (uint8_t soc_num, uint8_t mode, uint8_t config_mask)

  The bits in this register are defined sbtsi_config_write and can be written by writing to the corresponding bits in SBTSI::ConfigWr.
- oob_status_t sbtsi_get_timeout (uint8_t soc_num, uint8_t ∗timeout_en)

  To verify if timeout support enabled or disabled.
- oob_status_t sbtsi_set_timeout_config (uint8_t soc_num, uint8_t mode)

  To enable/disable timeout support.
- oob_status_t sbtsi_set_hitemp_threshold (uint8_t soc_num, float hitemp_thr)

  This value set the high temperature threshold. The high temperature threshold specifies the CPU temperature that causes ALERT_L to assert if the CPU temperature is greater than or equal to the threshold.
- oob_status_t sbtsi_set_lotemp_threshold (uint8_t soc_num, float lotemp_thr)

  This value set the low temperature threshold. The low temperature threshold specifies the CPU temperature that causes ALERT_L to assert if the CPU temperature is less than or equal to the threshold.
- oob_status_t sbtsi_get_hitemp_threshold (uint8_t soc_num, float ∗hitemp_thr)

  This value specifies the high temperature threshold. The high temperature threshold specifies the CPU temperature that causes ALERT_L to assert if the CPU temperature is greater than or equal to the threshold.
- oob_status_t sbtsi_get_lotemp_threshold (uint8_t soc_num, float ∗lotemp_thr)

  This value specifies the low temperature threshold. The low temperature threshold specifies the CPU temperature that causes ALERT_L to assert if the CPU temperature is less than or equal to the threshold.
- oob_status_t read_sbtsi_cputempoffset (uint8_t soc_num, float ∗temp_offset)

  SBTSI::CpuTempOffInt and SBTSI::CpuTempOffDec combine to specify the CPU temperature offset.
- oob_status_t write_sbtsi_cputempoffset (uint8_t soc_num, float temp_offset)

  SBTSI::CpuTempOffInt and SBTSI::CpuTempOffDec combine to set the CPU temperature offset.
- oob_status_t sbtsi_set_alert_threshold (uint8_t soc_num, uint8_t samples)

  Specifies the number of consecutive CPU temperature samples for which a temperature alert condition needs to remain valid before the corresponding alert bit is set.
- oob_status_t sbtsi_set_alert_config (uint8_t soc_num, uint8_t mode)

  Alert comparator mode enable.

7.7.1 Detailed Description

Header file for the APML library for SB-TSI functionality access. All required function, structure, enum, etc. definitions should be defined in this file for SB-TSI Register accessing.

This header file contains the following: APIs prototype of the APIs exported by the APML library. Description of the API, arguments and return values. The Error codes returned by the API.

7.8 rmi_mailbox_mi300.h File Reference

```
#include "apml_err.h"
```

Data Structures

- struct max_mem_bw

  Structure for Max DDR/HBM bandwidth and utilization. It contains max bandwidth(16 bit data) in GBps, current utilization bandwidth(Read+Write)(16 bit data) in GBps.
- struct svi_port_domain

  struct containing port and slave address.
- struct freq_limits

  struct containing max frequency and min frequencey limit
- struct mclk_fclk_pstates

  struct containing memory clock and fabric clock pstate mappings.
- struct statistics

  struct containing statistics parameter of interest and output control pstate mappings.
- struct xgmi_speed_rate_n_width

  struct containing xgmi speed rate in MHZ and link width in units of Gpbs. If link_width[0] = 1 then XGMI link X2 is supported. If link_width[1] = 1 then XGMI link X4 is supported. If Link_width[2] = 1 then XGMI link X4 is supported and similarly if link_width[3] = 1 then XGMI link X8 is supported.
- struct host_status

  struct containing power management controlling status, driving running status.

Enumerations

- enum esb_mi300_mailbox_commmands {
  SET_MAX_GFX_CORE_CLOCK = 0x81, **SET_MIN_GFX_CORE_CLOCK**, **SET_MAX_PSTATE**, **GET_P↩**
  STATES,
  SET_XGMI_PSTATE = 0x86, **UNSET_XGMI_PSTATE**, **GET_XGMI_PSTATES**, **GET_XCC_IDLE_RESI↩**
  DENCY,
  GET_ENERGY_ACCUMULATOR = 0x90, **GET_RAS_ALARMS**, **GET_PM_ALARMS**, **GET_PSN**,
  GET_LINK_INFO, **GET_ABS_MAX_MIN_GFX_FREQ** = 0x96, **GET_SVI_TELEMETRY_BY_RAIL**, **GET↩**
  _DIE_TYPE,
  GET_ACT_GFX_FREQ_CAP_SELECTED = 0x9c, **GET_DIE_HOT_SPOT_INFO** = 0xA0, **GET_MEM_H↩**
  OT_SPOT_INFO, **GET_STATUS** = 0xA4,
  GET_MAX_MEM_BW_UTILIZATION = 0XB0, **GET_HBM_THROTTLE**, **SET_HBM_THROTTLE**, **GET_H↩**
  BM_STACK_TEMP,
  GET_GFX_CLK_FREQ_LIMITS, **GET_FCLK_FREQ_LIMITS**, **GET_SOCKETS_IN_SYSTEM**, **GET_BIS↩**
  T_RESULTS = 0xBC,
  QUERY_STATISTICS, **CLEAR_STATISTICS** }

  Mailbox message types defined in the APML library.
- enum range_type { **MIN** = 0, **MAX** }

  APML range type used by GFX core clock frequency. Max, MIN are the values. Min is 0 and Max is 1.
- enum clk_type { **GFX_CLK** = 0, **F_CLK** }

  APML clock frequency type. GFX_CLK or F_CLK GFX_CLK value is 0 and F_CLK value is 1.
- enum alarms_type { **PM** }

  APML alarms type. PM_ALARMS. PM is 0.

Functions

- oob_status_t set_gfx_core_clock (uint8_t soc_num, enum range_type freq_type, uint32_t freq)

  Set maximum/minimum gfx core clock frequency.
- oob_status_t set_mclk_fclk_max_pstate (uint8_t soc_num, uint32_t pstate)

  Set maximum mem and fclck pstate.
- oob_status_t get_mclk_fclk_pstates (uint8_t soc_num, uint8_t pstate_ind, struct mclk_fclk_pstates ∗pstate)

Get memory and fabric clock power state mappings.
- oob_status_t set_xgmi_pstate (uint8_t soc_num, uint32_t pstate)

  Sets the XGMI Pstate.
- oob_status_t unset_xgmi_pstate (uint8_t soc_num)

  Resets the XGMI Pstate.
- oob_status_t get_xgmi_pstates (uint8_t soc_num, uint8_t pstate_ind, struct xgmi_speed_rate_n_width ∗xgmi_pstate)

  Read XGMI power state mappings.
- oob_status_t get_xcc_idle_residency (uint8_t soc_num, uint32_t ∗gfx_cores_idle_res)

  Read xcc idle residency percentage.
- oob_status_t get_energy_accum_with_timestamp (uint8_t soc_num, uint64_t ∗energy, uint64_t ∗time_←stamp)

  Read energy accumulator with time stamp.
- oob_status_t get_alarms (uint8_t soc_num, enum alarms_type type, uint32_t ∗buffer)

  Read PM alarm status based on enumeration type alarms_type.
- oob_status_t get_psn (uint8_t soc_num, uint32_t die_index, uint64_t ∗buffer)

  Reads public serial number (PSN).
- oob_status_t get_link_info (uint8_t soc_num, uint8_t ∗link_config, uint8_t ∗module_id)

  Read link Info.
- oob_status_t get_max_min_gfx_freq (uint8_t soc_num, uint16_t ∗max_freq, uint16_t ∗min_freq)

  Read maximum and minimum allowed GFX engine frequency.
- oob_status_t get_act_gfx_freq_cap (uint8_t soc_num, uint16_t ∗freq)

  Read Actual GFX frequency cap selected.
- oob_status_t get_svi_rail_telemetry (uint8_t soc_num, struct svi_port_domain port, uint32_t ∗pow)

  Read SVI based telemetry for individual rails.
- oob_status_t get_die_hotspot_info (uint8_t soc_num, uint8_t ∗die_id, uint16_t ∗temp)

  Reads local ID of the hottest die and its temperature.
- oob_status_t get_mem_hotspot_info (uint8_t soc_num, uint8_t ∗hbm_stack_id, uint16_t ∗hbm_temp)

  Reads local ID of the HBM stack and its temperature.
- oob_status_t get_host_status (uint8_t soc_num, struct host_status ∗status)

  Reads the status in a bit vector.
- oob_status_t get_max_mem_bw_util (uint8_t soc_num, struct max_mem_bw ∗bw)

  Reads max memory bandwidth utilization.
- oob_status_t get_hbm_throttle (uint8_t soc_num, uint32_t ∗mem_th)

  Reads HBM throttle.
- oob_status_t set_hbm_throttle (uint8_t soc_num, uint32_t mem_th)

  writes HBM throttle.
- oob_status_t get_hbm_temperature (uint8_t soc_num, uint32_t index, uint16_t ∗temp)

  Reads hbm stack temperature.
- oob_status_t get_clk_freq_limits (uint8_t soc_num, enum clk_type type, struct freq_limits ∗limit)

  Reads GFX/F clk frequency limits based on enumeration type clk_type .
- oob_status_t get_sockets_in_system (uint8_t soc_num, uint32_t ∗sockets_count)

  Reads number of sockets in system.
- oob_status_t get_bist_results (uint8_t soc_num, uint8_t die_id, uint32_t ∗bist_result)

  Reads die level bist result status from package.
- oob_status_t get_statistics (uint8_t soc_num, struct statistics stat, uint32_t ∗param)

  Reads statistics for a given parameter.
- oob_status_t clear_statistics (uint8_t soc_num)

  Clears statistics.
- oob_status_t get_die_type (uint8_t soc_num, uint32_t data_in, uint32_t ∗data_out)

  Get die type.

7.8.1 Detailed Description

Header file for the MI300 mailbox messages supported by APML library. All required function, structure, enum, etc. definitions should be defined in this file.

This header file contains the following: APIs prototype of the Mailbox messages for MI300 exported by the APML library. Description of the API, arguments and return values. The Error codes returned by the API.

7.8.2 Function Documentation

7.8.2.1 set_gfx_core_clock()

```
oob_status_t set_gfx_core_clock (
             uint8_t soc_num,
             enum range_type freq_type,
             uint32_t freq )
```

Set maximum/minimum gfx core clock frequency.

This function sets user provied frequency as MAX GFX/MIN GFX core clock frequency in MHZ based on enumeration type range_type.

Parameters

in	*soc_num*	Socket index.
in	*freq_type*	enumeration type range_type containing "MIN" = 0 or "MAX" = 1.
in	*freq*	frequency in MHZ.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.2 set_mclk_fclk_max_pstate()

```
oob_status_t set_mclk_fclk_max_pstate (
             uint8_t soc_num,
             uint32_t pstate )
```

Set maximum mem and fclck pstate.

This function sets the maximum memory and fabric clock power state. Mappings from memory and fabric clock pstate to MEMCLK/FCLK frequency can be found by issuing GetPstates command.

In case if the in-band has also set the maximum Pstate, then lower of the limits is used.

Parameters

in	*soc_num*	Socket index.
in	*pstate*	maximum pstate.Valid pstate range is 0 - 3.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.3 get_mclk_fclk_pstates()

```
oob_status_t get_mclk_fclk_pstates (
            uint8_t soc_num,
            uint8_t pstate_ind,
            struct mclk_fclk_pstates * pstate )
```

Get memory and fabric clock power state mappings.

This function returns the memory and fabric clock power state mappings. Returns MEMCLK/FCLK frequency in units of 1MHz for the available clock power states (Pstates). Each MEMCLK/FCLK frequency pair is returned independently for each pstate.

Parameters

in	*soc_num*	Socket index.
in	*pstate_ind*	index of the pstate.
out	*pstate*	struct mclk_fclk_pstates containing mem_clk and f_clk frequency in MHz.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.4 set_xgmi_pstate()

```
oob_status_t set_xgmi_pstate (
            uint8_t soc_num,
            uint32_t pstate )
```

Sets the XGMI Pstate.

This function sets the specified XGMI Pstate. This disables all active XGMI Pstate management although XGMI power down modes will still be supported. Only 2 XGMI states are supported (0/1).

Parameters

in	*soc_num*	Socket index.
in	*pstate*	power state. valid values are 0 - 1.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.5 unset_xgmi_pstate()

```
oob_status_t unset_xgmi_pstate (
              uint8_t soc_num )
```

Resets the XGMI Pstate.

This function resets the XGMI Pstate specified in the SetXgmiPstate, causing XGMI link speed/width to be actively managed by the GPU.

Parameters

in	*soc_num*	Socket index.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.6 get_xgmi_pstates()

```
oob_status_t get_xgmi_pstates (
              uint8_t soc_num,
              uint8_t pstate_ind,
              struct xgmi_speed_rate_n_width * xgmi_pstate )
```

Read XGMI power state mappings.

This function reads the XGMI power state mappings.Reads the supported XGMI link speeds and widths available to the SetXgmiPstate message.Link speeds reported in units of 1Gpbs.

Parameters

in	*soc_num*	Socket index.
in	*pstate_ind*	xgmi pstate index for speed rate.
out	*xgmi_pstate*	struct xgmi_speed_rate_n_width containing speed rate in MHz and link width

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.7 get_xcc_idle_residency()

```
oob_status_t get_xcc_idle_residency (
             uint8_t soc_num,
             uint32_t * gfx_cores_idle_res )
```

Read xcc idle residency percentage.

This function will provide the average xcc idle residency across all GFX cores in the socket.100% specifies that all enabled GFX cores in the socket are running in idle.

Parameters

in	*soc_num*	Socket index.
out	*gfx_cores_idle_res*	idle residency in percentage

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.8 get_energy_accum_with_timestamp()

```
oob_status_t get_energy_accum_with_timestamp (
             uint8_t soc_num,
             uint64_t * energy,
             uint64_t * time_stamp )
```

Read energy accumulator with time stamp.

This function will read 64 bits energy accumulator and the 56 bit time stamp.

Parameters

in	*soc_num*	Socket index.
out	*energy*	accumulator 2^ 16 J.
out	*time_stamp*	time stamp (units:10ns).

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.9 get_alarms()

```
oob_status_t get_alarms (
              uint8_t soc_num,
              enum alarms_type type,
              uint32_t * buffer )
```

Read PM alarm status based on enumeration type alarms_type.

This function provides PM alarm status if the enumeration type alarms_type is PM = 0 then it will retrieve PM alarm status . If buffer value is 1 the status is VRHOT. If the buffer value is 2 status is die over temp. If the buffer value is 4 status is HBM over temp and if the buffer is 8 then status is PWRBRK. Supported platforms: Fam-19h_Mod-90h-9Fh.

Parameters

in	*soc_num*	Socket index.
in	*type*	enumeration type alarms_type. PM = 0.
out	*buffer*	returns PSP fw return data.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.10 get_psn()

```
oob_status_t get_psn (
              uint8_t soc_num,
              uint32_t die_index,
              uint64_t * buffer )
```

Reads public serial number (PSN).

This function will return 64 bit public serial number (PSN) unique to each die.

Parameters

in	*soc_num*	Socket index.
in	*die_index*	core/die index.
out	*buffer*	returns 64 bit unique public serial number.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.11 get_link_info()

```
oob_status_t get_link_info (
             uint8_t soc_num,
             uint8_t * link_config,
             uint8_t * module_id )
```

Read link Info.

This function will read link info. Function will read the module ID and link config reflecting strapping pins.

Parameters

in	*soc_num*	Socket index.
out	*link_config*	link configuration.
out	*module↩ _id*	module ID.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.12 get_max_min_gfx_freq()

```
oob_status_t get_max_min_gfx_freq (
             uint8_t soc_num,
             uint16_t * max_freq,
             uint16_t * min_freq )
```

Read maximum and minimum allowed GFX engine frequency.

This function will read maximum and minimum allowed GFX engine frequency. Supported platforms: Fam-19h_↩ Mod-90h-9Fh.

Parameters

in	*soc_num*	Socket index.
out	*max_freq*	maximum GFX frequency in MHZ.
out	*min_freq*	minimum GFX frequency in MHZ.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.13 get_act_gfx_freq_cap()

```
oob_status_t get_act_gfx_freq_cap (
             uint8_t soc_num,
             uint16_t * freq )
```

Read Actual GFX frequency cap selected.

This function will read current seleted GFX engine clock frequency.It reflects minimum of all frequency caps seleted via in-band and out-of-band controls. Supported platforms: Fam-19h_Mod-90h-9Fh.

Parameters

in	*soc_num*	Socket index.
out	*freq*	maximum GFX frequency in MHZ.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.14 get_svi_rail_telemetry()

```
oob_status_t get_svi_rail_telemetry (
             uint8_t soc_num,
             struct svi_port_domain port,
             uint32_t * pow )
```

Read SVI based telemetry for individual rails.

This function will read SVI based telemetry for individual rails.

Parameters

in	*soc_num*	Socket index.
in	*port*	struct svi_telemetry_domain containing port and slave address.
out	*pow*	power in milliwatts.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.15 get_die_hotspot_info()

```
oob_status_t get_die_hotspot_info (
             uint8_t soc_num,
             uint8_t * die_id,
             uint16_t * temp )
```

Reads local ID of the hottest die and its temperature.

This function will read local ID of the hottest die and its corresponding die temperature. Measured in every 1 ms and the most recently measured temperature in ℃ is reported.

Parameters

in	*soc_num*	Socket index.
out	*die_id*	Hottest die ID.
out	*temp*	Die hot spot temperature in ℃.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.16 get_mem_hotspot_info()

```
oob_status_t get_mem_hotspot_info (
             uint8_t soc_num,
             uint8_t * hbm_stack_id,
             uint16_t * hbm_temp )
```

Reads local ID of the HBM stack and its temperature.

This function will read local ID of the HBM stack and its corresponding HBM stack temperature. Measured in every 1 ms and the most recently measured temperature is reported.

Parameters

in	*soc_num*	Socket index.
out	*hbm_stack↩_id*	Local ID of the hottest HBM stack.
out	*hbm_temp*	temperature in units of 1 ℃.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.17 get_host_status()

```
oob_status_t get_host_status (
            uint8_t soc_num,
            struct host_status * status )
```

Reads the status in a bit vector.

This function will read PM controller status and driver running status in a bit vector

Parameters

in	*soc_num*	Socket index.
out	*status*	struct host_status containing power management controller status and driver running status.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.18 get_max_mem_bw_util()

```
oob_status_t get_max_mem_bw_util (
            uint8_t soc_num,
            struct max_mem_bw * bw )
```

Reads max memory bandwidth utilization.

This function will provide theoretic.al maximum HBM/memory bandwidth of the system in GB/s, utilized bandwidth in GB/S.

Parameters

in	*soc_num*	Socket index.
out	*bw*	struct max_mem_bw containing max bw, utilized b/w.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.19 get_hbm_throttle()

```
oob_status_t get_hbm_throttle (
             uint8_t soc_num,
             uint32_t * mem_th )
```

Reads HBM throttle.

This function will read HBM throttle in percentage.

Parameters

in	*soc_num*	Socket index.
out	*mem_th*	hbm throttle in percentage (0 - 100%).

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.20 set_hbm_throttle()

```
oob_status_t set_hbm_throttle (
             uint8_t soc_num,
             uint32_t mem_th )
```

writes HBM throttle.

This function will write HBM throttle.

Parameters

in	*soc_num*	Socket index.
in	*mem_th*	hbm throttle in percentage (0 - 80%).

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.21 get_hbm_temperature()

```
oob_status_t get_hbm_temperature (
                uint8_t soc_num,
                uint32_t index,
                uint16_t * temp )
```

Reads hbm stack temperature.

This function will read particular hbm stack temperature.

Parameters

in	*soc_num*	Socket index.
in	*index*	hbm stack index (0 - 7 for MI300).
out	*temp*	temperature in units of 1 ℃.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.22 get_clk_freq_limits()

```
oob_status_t get_clk_freq_limits (
                uint8_t soc_num,
                enum clk_type type,
                struct freq_limits * limit )
```

Reads GFX/F clk frequency limits based on enumeration type clk_type .

This function will provide socket's GFX/F clk max and min frequnecy limits based on enumberation type clk_type . The function reads GFX clk frequency limits if the enumberation type clk_type is GFX_CLK = 0 else it will read F_CLK frequency limits.

Parameters

in	*soc_num*	Socket index.
in	*type*	enumeration type clk_type . Values are "GFX_CLK" = 0 or "F_CLK" = 1.
out	*limit*	struct freq_limits containing max and min GFX/F_clk frequency in MHZ.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.23 get_sockets_in_system()

```
oob_status_t get_sockets_in_system (
            uint8_t soc_num,
            uint32_t * sockets_count )
```

Reads number of sockets in system.

This function will read number of sockets in system.

Parameters

in	*soc_num*	Socket index.
out	*sockets_count*	Numbers of sockets in system

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.24 get_bist_results()

```
oob_status_t get_bist_results (
            uint8_t soc_num,
            uint8_t die_id,
            uint32_t * bist_result )
```

Reads die level bist result status from package.

This function will read die level bist result status from package.

Parameters

in	*soc_num*	Socket index.
in	*die_id*	die level id.
out	*bist_result*	constituent bist results depending on MI300X/A/C configuration.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.25 get_statistics()

```
oob_status_t get_statistics (
            uint8_t soc_num,
            struct statistics stat,
            uint32_t * param )
```

Reads statistics for a given parameter.

This function will read statistics for a given parameter since the last clear statistics command.

Parameters

in	*soc_num*	Socket index.
in	*stat*	struct statistics containing statistics parameter of interest and output control.
out	*param*	parameter or timestamp HI/Lo value.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.26 clear_statistics()

```
oob_status_t clear_statistics (
             uint8_t soc_num )
```

Clears statistics.

This function will clear all stored query statistics timestamps and then resumes data collection or aggregation.

Parameters

in	*soc_num*	Socket index.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.8.2.27 get_die_type()

```
oob_status_t get_die_type (
             uint8_t soc_num,
             uint32_t data_in,
             uint32_t * data_out )
```

Get die type.

This function will read die-type, counts and AID base die based on die-ID input. If the bit[0] of input/data_in is 1 then it will get maximum die-ID (0 - 255). If the bit[0] of input/data_in is 0 then the data_out[7:0] will be die type i.e. 0 Not available, 1 means AID, 2 means XCD, 3 means CCD and 4 means HBM stack and 5- 255 are reserved. data_out[15:8] means maximum coumt of currently indexed die type. Data_out[19:16] means AID associated with specified die-ID.

Parameters

in	*soc_num*	Socket index.
in	*data_in*	input to get the die-type, counts and AID base.
out	*data_out*	maximum die id idnex or the die type based on die-id input.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.9 tsi_mi300.h File Reference

```
#include "apml_err.h"
```

Enumerations

- enum sbtsi_mi300_registers {
  SBTSI_HBM_HITEMPINT_LIMIT = 0x40, **SBTSI_HBM_HITEMPDEC_LIMIT** = 0x44, **SBTSI_HBM_LOT↩ EMPINT_LIMIT** = 0x48, **SBTSI_HBM_LOTEMPDEC_LIMIT** = 0x4C,
  SBTSI_MAX_HBMTEMPINT = 0x50, **SBTSI_MAX_HBMTEMPDEC** = 0x54, **SBTSI_HBMTEMPINT** = 0x5C,
  SBTSI_HBMTEMPDEC = 0x60 }

  SB-TSI(Side-Band Temperature Sensor Interface) commands register access.

Functions

- oob_status_t read_sbtsi_hbm_hi_temp_int_th (uint8_t soc_num, uint8_t ∗buffer)

  Get high hbm temperature integer threshold in ℃.
- oob_status_t write_sbtsi_hbm_hi_temp_th (uint8_t soc_num, float hi_temp_th)

  Set high hbm temperature integer and decimal threshold in ℃.
- oob_status_t read_sbtsi_hbm_hi_temp_dec_th (uint8_t soc_num, float ∗buffer)

  Get high hbm temperature decimal threshold.
- oob_status_t read_sbtsi_hbm_hi_temp_th (uint8_t soc_num, float ∗buffer)

  Get high hbm temperature threshold.
- oob_status_t read_sbtsi_hbm_lo_temp_int_th (uint8_t soc_num, uint8_t ∗buffer)

  Get low hbm temperature integer threshold.
- oob_status_t read_sbtsi_hbm_lo_temp_dec_th (uint8_t soc_num, float ∗buffer)

  Get low hbm temperature decimal threshold.
- oob_status_t write_sbtsi_hbm_lo_temp_th (uint8_t soc_num, float temp_th)

  Set low hbm temperature threshold.
- oob_status_t read_sbtsi_max_hbm_temp_int (uint8_t soc_num, uint8_t ∗buffer)

  Get max hbm integer temperature.
- oob_status_t read_sbtsi_max_hbm_temp_dec (uint8_t soc_num, float ∗buffer)

  Get max hbm decimal temperature.
- oob_status_t read_sbtsi_hbm_temp_int (uint8_t soc_num, uint8_t ∗buffer)

  Get hbm integer temperature.

- oob_status_t read_sbtsi_hbm_temp_dec (uint8_t soc_num, float ∗buffer)

  Get hbm decimal temperature.
- oob_status_t read_sbtsi_hbm_lo_temp_th (uint8_t soc_num, float ∗buffer)

  Get hbm low temperature threshold.
- oob_status_t read_sbtsi_max_hbm_temp (uint8_t soc_num, float ∗buffer)

  Get hbm maximum temperature.
- oob_status_t read_sbtsi_hbm_temp (uint8_t soc_num, float ∗buffer)

  Get hbm temperature.
- oob_status_t read_sbtsi_hbm_alertthreshold (uint8_t soc_num, uint8_t ∗samples)

  Get hbm alert threshold.
- oob_status_t sbtsi_set_hbm_alert_threshold (uint8_t soc_num, uint8_t samples)

  Set hbm alert samples.
- oob_status_t get_sbtsi_hbm_alertconfig (uint8_t soc_num, uint8_t ∗mode)

  Get the sbtsi hbm alert config.
- oob_status_t set_sbtsi_hbm_alertconfig (uint8_t soc_num, uint8_t mode)

  Set hbm alert config.

7.9.1 Detailed Description

Header file for the APML library for SB-TSI functionality access for MI300. All required function, structure, enum, etc. definitions should be defined in this file for SB-TSI Register accessing.

This header file contains the following: APIs prototype of the APIs exported by the APML library. Description of the API, arguments and return values. The Error codes returned by the API.

7.9.2 Function Documentation

7.9.2.1 read_sbtsi_hbm_hi_temp_int_th()

```
oob_status_t read_sbtsi_hbm_hi_temp_int_th (
            uint8_t soc_num,
            uint8_t ∗ buffer )
```

Get high hbm temperature integer threshold in ℃.

This function will read high hbm temperature interger threshold in ℃.

Parameters

in	*soc_num*	Socket index.
out	*buffer*	a pointer to hold the high hbm temperature integer threshold in ℃.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.9.2.2 write_sbtsi_hbm_hi_temp_th()

```
oob_status_t write_sbtsi_hbm_hi_temp_th (
            uint8_t soc_num,
            float hi_temp_th )
```

Set high hbm temperature integer and decimal threshold in ℃.

This function will set high hbm temperature interger and decimal threshold in ℃.

Parameters

in	*soc_num*	Socket index.
in	*hi_temp←↩ _th*	high hbm temperature threshold containing integer and decimal part in ℃.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.9.2.3 read_sbtsi_hbm_hi_temp_dec_th()

```
oob_status_t read_sbtsi_hbm_hi_temp_dec_th (
            uint8_t soc_num,
            float * buffer )
```

Get high hbm temperature decimal threshold.

This function will read high hbm temperature decimal threshold in ℃.

Parameters

in	*soc_num*	Socket index.
out	*buffer*	a pointer to hold the high hbm temperature decimal threshold in ℃.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.9.2.4 read_sbtsi_hbm_hi_temp_th()

```
oob_status_t read_sbtsi_hbm_hi_temp_th (
            uint8_t soc_num,
            float * buffer )
```

Get high hbm temperature threshold.

This function will read high hbm temperature threshold in ℃.

Parameters

in	*soc_num*	Socket index.
out	*buffer*	a pointer to hold the high hbm temperature in ℃

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.9.2.5 read_sbtsi_hbm_lo_temp_int_th()

```
oob_status_t read_sbtsi_hbm_lo_temp_int_th (
            uint8_t soc_num,
            uint8_t * buffer )
```

Get low hbm temperature integer threshold.

This function will read low hbm temperature interger threshold in ℃.

Parameters

in	*soc_num*	Socket index.
out	*buffer*	a pointer to hold the low hbm temperature integer threshold in ℃.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.9.2.6 read_sbtsi_hbm_lo_temp_dec_th()

```
oob_status_t read_sbtsi_hbm_lo_temp_dec_th (
            uint8_t soc_num,
            float * buffer )
```

Get low hbm temperature decimal threshold.

This function will low high hbm temperature decimal threshold in ℃.

Parameters

in	*soc_num*	Socket index.
out	*buffer*	a pointer to hold the low hbm temperature decimal threshold in ℃.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.9.2.7 write_sbtsi_hbm_lo_temp_th()

```
oob_status_t write_sbtsi_hbm_lo_temp_th (
            uint8_t soc_num,
            float temp_th )
```

Set low hbm temperature threshold.

This function will set low hbm temperature threshold in ℃.

Parameters

in	*soc_num*	Socket index.
in	*temp_th*	low hbm temperature threshold in ℃.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.9.2.8 read_sbtsi_max_hbm_temp_int()

```
oob_status_t read_sbtsi_max_hbm_temp_int (
            uint8_t soc_num,
            uint8_t * buffer )
```

Get max hbm integer temperature.

This function will read max hbm interger temperature in ℃

Parameters

in	*soc_num*	Socket index.
out	*buffer*	a pointer to hold the max hbm integer temperature in ℃

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.9.2.9 read_sbtsi_max_hbm_temp_dec()

```
oob_status_t read_sbtsi_max_hbm_temp_dec (
            uint8_t soc_num,
            float * buffer )
```

Get max hbm decimal temperature.

This function will read max hbm decimal temperature in ℃

Parameters

in	*soc_num*	Socket index.
out	*buffer*	a pointer to hold the max hbm decimal temperature in ℃.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.9.2.10 read_sbtsi_hbm_temp_int()

```
oob_status_t read_sbtsi_hbm_temp_int (
            uint8_t soc_num,
            uint8_t * buffer )
```

Get hbm integer temperature.

This function will read hbm integer temperature in ℃.

Parameters

in	*soc_num*	Socket index.
out	*buffer*	a pointer to hold the hbm integer temperature in ℃

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.9.2.11 read_sbtsi_hbm_temp_dec()

```
oob_status_t read_sbtsi_hbm_temp_dec (
             uint8_t soc_num,
             float * buffer )
```

Get hbm decimal temperature.

This function will read hbm decimal temperature in ℃.

Parameters

in	*soc_num*	Socket index.
out	*buffer*	a pointer to hold the hbm decimal temperature in ℃

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.9.2.12 read_sbtsi_hbm_lo_temp_th()

```
oob_status_t read_sbtsi_hbm_lo_temp_th (
             uint8_t soc_num,
             float * buffer )
```

Get hbm low temperature threshold.

This function will read hbm low threshold temperature in ℃.

Parameters

in	*soc_num*	Socket index.
out	*buffer*	a pointer to hold the low hbm temperature threshold in ℃.

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.9.2.13 read_sbtsi_max_hbm_temp()

```
oob_status_t read_sbtsi_max_hbm_temp (
              uint8_t soc_num,
              float * buffer )
```

Get hbm maximum temperature.

This function will read maximum hbm temperature in ℃.

Parameters

in	*soc_num*	Socket index.
out	*buffer*	a pointer to hold the max hbm temperature in ℃

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.9.2.14 read_sbtsi_hbm_temp()

```
oob_status_t read_sbtsi_hbm_temp (
              uint8_t soc_num,
              float * buffer )
```

Get hbm temperature.

This function will read maximum hbm temperature in ℃.

Parameters

in	*soc_num*	Socket index.
out	*buffer*	a pointer to hold the hbm temperature in ℃

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.9.2.15 read_sbtsi_hbm_alertthreshold()

```
oob_status_t read_sbtsi_hbm_alertthreshold (
            uint8_t soc_num,
            uint8_t * samples )
```

Get hbm alert threshold.

This function will read hbm alert threshold.

Parameters

in	*soc_num*	Socket index.
out	*samples*	hbm threshold samples

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.9.2.16 sbtsi_set_hbm_alert_threshold()

```
oob_status_t sbtsi_set_hbm_alert_threshold (
            uint8_t soc_num,
            uint8_t samples )
```

Set hbm alert samples.

This function will set hbm alert samples.

Parameters

in	*soc_num*	Socket index.
in	*samples*	hbm threshold samples

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.9.2.17 get_sbtsi_hbm_alertconfig()

```
oob_status_t get_sbtsi_hbm_alertconfig (
            uint8_t soc_num,
            uint8_t * mode )
```

Get the sbtsi hbm alert config.

This function will read hbm alert config. 1 indicates Enable hbm high and low temperature alert and 0 indicates disbale hbm high and low temperature alert.

Parameters

`in`	*soc_num*	Socket index.
`in,out`	*mode*	HBM alert enable or disable

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

7.9.2.18 set_sbtsi_hbm_alertconfig()

```
oob_status_t set_sbtsi_hbm_alertconfig (
            uint8_t soc_num,
            uint8_t mode )
```

Set hbm alert config.

This function will set hbm alert config. 1 indicates enable hbm high and low temperature alert and 0 indicates disable hbm high and low temperature alert.

Parameters

`in`	*soc_num*	Socket index.
`in`	*mode*	1 indicates enable and 0 indicates disable

Return values

OOB_SUCCESS	is returned upon successful call.
Non-zero	is returned upon failure.

Index