E-SMI-OOB Library: EPYC™ Systems Management Interface Out-of-band Library

Generated by Doxygen 1.8.13

Contents

Chapter 1

EPYC™ System Management Interface Out-of-band (E-SMI-OOB) Library

The EPYC™ System Management Interface Out-of-band Library or E-SMI-OOB 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.

Important note about Versioning and Backward Compatibility

The E-SMI-OOB 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 E-SMI-OOB 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.

Building E-SMI-OOB

Additional Required software for building

In order to build the E-SMI-OOB 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)
- i2c-tools, libi2c-dev

Dowloading the source

The source code for E-SMI library is available on Github.

Directory stucture of the source

Once the E-SMI-OOB 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
- `$ example/` Contains esmi_oob_tool and esmi_oob_ex based on the E-SMI-OOB library
- `$ include/esmi_oob` Contains the header files used by the E-SMI-OOB library
- `$ src/esmi_oob` Contains library E-SMI-OOB source

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

```
$ mkdir -p build

$ cd build

$ cmake -DCMAKE_INSTALL_PREFIX=${PWD}/install <location of root of E-SMI-O↩
OB library CMakeLists.txt>

$ make
```

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

Building the Documentation

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

```
$ make doc

$ cd latex

$ make
```

The reference manual, `refman.pdf` will be in the `latex` directory and `refman.rtf` will be in the `rtf` directory upon a successful build.

Usage Basics

Device Indices

Many of the functions in the library take a "core or socket index". The core or socket index is a number greater than or equal to 0, and less than the number of cores or sockets on the system.

Hello E-SMI-OOB

The only required E-SMI-OOB call for any program that wants to use E-SMI-OOB is the `esmi_oob_init()` call. This call initializes some internal data structures that will be used by subsequent E-SMI-OOB calls.

When E-SMI-OOB is no longer being used, `esmi_oob_exit()` should be called. This provides a way to do any releasing of resources that E-SMI-OOB may have held. In many cases, this may have no effect, but may be necessary in future versions of the library.

Below is a simple "Hello World" type program that displays the Core energy of detected cores.

```
#include <stdio.h>
#include <stdint.h>
#include <esmi_oob/esmi_common.h>
#include <esmi_oob/esmi_mailbox.h>

int main() {
    oob_status_t ret;
    int package = 0, buffer = 0;

    ret = esmi_oob_init(1);
    if (ret != OOB_SUCCESS) {
        printf("I2CDEV INIT FAILED. Err[%d]\n", ret);
        return ret;
    }
    if (read_min_tdp(package, &buffer) < 0)
        goto x;
    printf("package[%d] Min TDP: %lx\n", package, buffer);

x:
    esmi_oob_exit();
    return ret;
}
```

Usage

Tool Usage

E-SMI tool is a C program based on the E-SMI Out-of-band Library, the executable "esmi_oob_tool" will be generated in the build/ folder. This tool provides options to Monitor and Control System Management functionality.

Below is a sample usage to dump the functionality, with default core/socket available.

```
esmi_oob_library/build> ./esmi_oob_tool
=============== APML System Management Interface ===============

--------------------------------------------------------------------------------
            SOCKET 0
--------------------------------------------------------------------------------
*** SB-RMI Mailbox Service Access ***
_POWER  (Watts)      | Avg : 56.068,  Limit : 200.000,  Max : 240.000
_TDP    (Watts)      | Avg : 225.000,  Minim : 225.000,  Max : 240.000
_CCLK_FREQ_LIMIT (MHz)  | 3200
_C0_RESIDENCY (in %)    | 0%
_BOOST_LIMIT (MHz)  | BIOS: 3200,  APML: 3200
_DRAM_THROTTLE  (in %)  | 50
_PROCHOT STATUS     | NOT_PROCHOT
_PROCHOT RESIDENCY (MHz)| 0
_VDDIOMem_POWER (mWatts)| 36460
_NBIO_Error_Logging_Reg | 0
_IOD_Bist_RESULT    | Bist pass
_CCD_Bist_RESULT    | Bist pass
_CCX_Bist_RESULT    | 0

*** SB-TSI UPDATE ***
_CPUTEMP        | 20.625 °C
_HIGH_THRESHOLD_TEMP    | 70.000 °C
_LOW_THRESHOLD_TEMP     | 2.125 °C
_TSI_UPDATERATE     | 0.125 Hz
```

```
_THRESHOLD_SAMPLE   | 1
_TEMP_OFFSET        | -21.125 °C
_STATUS         | No Temp Alert
_CONFIG         |
    ALERT_L pin | Enabled
    Runstop     | Enabled
    Atomic rd order | Disabled
    ARA response    | Enabled
_TIMEOUT_CONFIG     | Enabled
_TSI_ALERT_CONFIG   | Enabled
_TSI_MANUFACTURE_ID | 0
_TSI_REVISION       | 0x4
--------------------------------------------------------------------------------
Try './esmi_oob_tool --help' for more information.

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

For detailed and up to date usage information, we recommend consulting the help:

For convenience purposes, following is the output from the -h flag:

```
esmi_oob_library/build> ./esmi_oob_tool --help
Usage: ./esmi_oob_tool [Option<s>] SOURCES
Option<s>:
< MAILBOX COMMANDS >:
    -p, (--showpower)     [SOCKET]           Get Power for a given socket in Watt
    -t, (--showtdp)       [SOCKET]           Get TDP for a given socket in Watt
    -s, (--setpowerlimit)     [SOCKET][POWER]        Set powerlimit for a given socket in mWatt
    -c, (--showcclkfreqlimit)[SOCKET]           Get cclk freqlimit for a given socket in MHz
    -r, (--showc0residency)   [SOCKET]           Show socket c0_residency given socket
    -b, (--showboostlimit)    [SOCKET][THREAD]       Get APML and BIOS boostlimit for a given socket and
       core index in MHz
    -d, (--setapmlboostlimit)[SOCKET][THREAD][BOOSTLIMIT]   Set APML boostlimit for a given socket and core
       in MHz
    -a, (--setapmlsocketboostlimit)  [SOCKET][BOOSTLIMIT]   Set APML boostlimit for all cores in a socket
       in MHz
    --set_and_verify_dramthrottle    [SOCKET][0 to 80%]     Set DRAM THROTTLE for a given socket
< SB-RMI COMMANDS >:
    --showrmicommandregisters [SOCKET]          Get the values of different commands of SB-RMI registers
       for a given socket
< SB-TSI COMMANDS >:
    --showtsicommandregisters [SOCKET]          Get the values of different commands of SB-TSI registers
       for a given socket
    --set_verify_updaterate   [SOCKET][Hz]          Set APML Frequency Update rate for a socket
    --sethightempthreshold    [SOCKET][TEMP(°C)]       Set APML High Temp Threshold
    --setlowtempthreshold     [SOCKET][TEMP(°C)]       Set APML Low Temp Threshold
    --settempoffset       [SOCKET][VALUE]       Set APML CPU Temp Offset, VALUE = [-CPU_TEMP(°C), 127 °C]
    --settimeoutconfig    [SOCKET][VALUE]       Set/Reset APML CPU timeout config, VALUE = 0 or 1
    --setalertthreshold   [SOCKET][VALUE]       Set APML CPU alert threshold sample, VALUE = 1 to 8
    --setalertconfig      [SOCKET][VALUE]       Set/Reset APML CPU alert config, VALUE = 0 or 1
    --setalertmask        [SOCKET][VALUE]       Set/Reset APML CPU alert mask, VALUE = 0 or 1
    --setrunstop          [SOCKET][VALUE]       Set/Reset APML CPU runstop, VALUE = 0 or 1
    --setreadorder        [SOCKET][VALUE]       Set/Reset APML CPU read order, VALUE = 0 or 1
    --setara          [SOCKET][VALUE]       Set/Reset APML CPU ARA, VALUE = 0 or 1
    -h, (--help)                        Show this help message
```

Below is a sample usage to get the individual library functionality API's. We can pass arguments either any of the ways "./esmi_oob_tool -p 0" or "./esmi_oob_tool --showpower=0"

```
1.  esmi_oob_library/build> ./esmi_oob_tool -p 0
    ============== APML System Management Interface ==============

    socket[0]/power:           56.155 Watts
    socket[0]/powerlimit:      200.000 Watts
    socket[0]/max_power_limit: 240.000 Watts


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

2.  esmi_oob_library/build> ./esmi_oob_tool --setpowerlimit 0 220000
    ============== APML System Management Interface ==============

    Set socket[0]/power_limit :          220.000 Watts successfully

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

3.  esmi_oob_library/build> ./esmi_oob_tool --showtsicommandregisters 0
    ============== APML System Management Interface ==============
```

```
         *** SB-TSI UPDATE ***
Socket:0
--------------------------------------------------------------------------------
_CPUTEMP        | 20.625 °C
_HIGH_THRESHOLD_TEMP    | 70.000 °C
_LOW_THRESHOLD_TEMP     | 2.125 °C
_TSI_UPDATERATE     | 0.125 Hz
_THRESHOLD_SAMPLE   | 1
_TEMP_OFFSET        | -21.125 °C
_STATUS         | No Temp Alert
_CONFIG         |
    ALERT_L pin | Enabled
    Runstop     | Enabled
    Atomic rd order | Disabled
    ARA response    | Enabled
_TIMEOUT_CONFIG     | Enabled
_TSI_ALERT_CONFIG   | Enabled
_TSI_MANUFACTURE_ID | 0
_TSI_REVISION       | 0x4
--------------------------------------------------------------------------------

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

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 Initialization and Shutdown

Functions

- oob_status_t esmi_oob_init (int i2c_channel)

  maintain the file descriptor of i2c device.
- void esmi_oob_exit (void)

  Closes the i2c channel device which was opened during init.

5.1.1 Detailed Description

This function handles the i2c device file used by the APIs.

5.1.2 Function Documentation

5.1.2.1 esmi_oob_init()

```
oob_status_t esmi_oob_init (
             int i2c_channel )
```

maintain the file descriptor of i2c device.

get the file descriptor by opening the particular i2c channel.

Return values

OOB_SUCCESS	non-negative integer, file descriptor is returned upon successful call.
-1	is returned upon failure.

5.2 Auxiliary functions

Functions

- oob_status_t errno_to_oob_status (int err)
  convert linux error to esmi error.
- uint32_t esmi_get_logical_cores_per_socket (void)
  Get the number of logical cores per socket.
- uint32_t esmi_get_threads_per_socket (void)
  Get the number of threads per socket.
- uint32_t esmi_get_threads_per_core (void)
  Get number of threads per core.
- char * esmi_get_err_msg (oob_status_t oob_err)
  Get the error string message for esmi oob errors.

5.2.1 Detailed Description

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

5.2.2 Function Documentation

5.2.2.1 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

5.2.2.2 esmi_get_logical_cores_per_socket()

```
uint32_t esmi_get_logical_cores_per_socket (
            void  )
```

Get the number of logical cores per socket.

Get the total number of logical cores in a socket.

Return values

uint32_t	is returned upon successful call.

5.2.2.3 esmi_get_threads_per_socket()

```
uint32_t esmi_get_threads_per_socket (
            void  )
```

Get the number of threads per socket.

Get the total number of threads in a socket.

Return values

uint32_t	is returned upon successful call.

5.2.2.4 esmi_get_threads_per_core()

```
uint32_t esmi_get_threads_per_core (
            void  )
```

Get number of threads per core.

Get the number of threads per core.

Return values

uint32_t	is returned upon successful call.

5.2.2.5 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.

5.3 SB-RMI Mailbox Service

Modules

- Power Monitor
- Power Control
- Performance (Boost limit) Monitor
- Out-of-band Performance (Boost limit) Control
- Current, Min, Max TDP
- Prochot
- Dram and other features Query

5.3.1 Detailed Description

Below functions to support SB-RMI Mailbox messages to read, write, 'write and read' operations for a given socket.

5.4 Power Monitor

Functions

- oob_status_t read_socket_power (int socket_ind, uint32_t *ppower)

  Get the average power consumption of the socket with provided socket index.
- oob_status_t read_socket_power_limit (int socket_ind, uint32_t *pcap)

  Get the current power cap/limit value for a given socket.
- oob_status_t read_max_socket_power_limit (int socket_ind, uint32_t *pmax)

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

5.4.1 Detailed Description

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

5.4.2 Function Documentation

5.4.2.1 read_socket_power()

```
oob_status_t read_socket_power (
            int socket_ind,
            uint32_t * ppower )
```

Get the average power consumption of the socket with provided socket index.

Given a socket index `socket_ind` and a pointer to a uint32t `ppower`, this function will get the current average power consumption (in milliwatts) to the uint32t pointed to by `ppower`.

Parameters

`in`	*socket_ind*	a socket index
`in,out`	*ppower*	a pointer to uint32t to which the average power consumption will get

Return values

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

5.4.2.2 read_socket_power_limit()

```
oob_status_t read_socket_power_limit (
            int socket_ind,
            uint32_t * pcap )
```

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

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

Parameters

`in`	*socket_ind*	a socket index
`in,out`	*pcap*	a pointer to a uint32t that indicates the valid possible power cap/limit, in milliwatts

Return values

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

5.4.2.3 read_max_socket_power_limit()

```
oob_status_t read_max_socket_power_limit (
             int socket_ind,
             uint32_t * pmax )
```

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 `pmax` from a `socket_ind`.

Parameters

`in`	*socket_ind*	a socket index
`in,out`	*pmax*	a pointer to a uint32t that indicates the maximum possible power cap/limit, in milliwatts

Return values

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

5.5 Power Control

Functions

- oob_status_t write_socket_power_limit (int socket_ind, uint32_t limit)

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

5.5.1 Detailed Description

This function provides a way to control Power Limit.

5.5.2 Function Documentation

5.5.2.1 write_socket_power_limit()

```
oob_status_t write_socket_power_limit (
             int socket_ind,
             uint32_t limit )
```

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

This function will set the power cap/limit to the provided value `cap`.

Parameters

in	*socket_ind*	a socket index
in	*limit*	uint32t that indicates the desired power cap/limit, in milliwatts

Return values

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

5.6 Performance (Boost limit) Monitor

Functions

- oob_status_t read_esb_boost_limit (int socket_ind, uint32_t cpu_ind, uint32_t *pboostlimit)
  Get the Out-of-band boostlimit value for a given core.
- oob_status_t read_bios_boost_fmax (int socket_ind, uint32_t value, uint32_t *buffer)
  Get the In-band maximum boostlimit value for a given core.

5.6.1 Detailed Description

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

5.6.2 Function Documentation

5.6.2.1 read_esb_boost_limit()

```
oob_status_t read_esb_boost_limit (
            int socket_ind,
            uint32_t cpu_ind,
            uint32_t * pboostlimit )
```

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

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

Parameters

in	*socket_ind*	a socket index
in	*cpu_ind*	a cpu index
in,out	*pboostlimit*	pointer to a uint32t that indicates the possible boost limit value

Return values

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

5.6.2.2 read_bios_boost_fmax()

```
oob_status_t read_bios_boost_fmax (
            int socket_ind,
```

```
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`	*socket_ind*	a socket index
`in`	*value*	is a cpu index
`in,out`	*buffer*	a pointer to a uint32t that indicates the maximum boost limit value set via In-band

Return values

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

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

Functions

- oob_status_t write_esb_boost_limit (int socket_ind, int cpu_id, uint32_t limit)

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

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

5.7.1 Detailed Description

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

5.7.2 Function Documentation

5.7.2.1 write_esb_boost_limit()

```
oob_status_t write_esb_boost_limit (
             int socket_ind,
             int cpu_id,
             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 via Out-of-band. NOTE: Currently the limit is setting for all the cores instead of a particular cpu. Testing in Progress.

Parameters

in	*socket_ind*	a socket index
in	*cpu_id*	a cpu index is a given core to set the boostlimit
in	*limit*	a uint32t that indicates the desired Out-of-band boostlimit value of a given core

Return values

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

5.7.2.2 write_esb_boost_limit_allcores()

```
oob_status_t write_esb_boost_limit_allcores (
             int socket_ind,
             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 whole socket.

Parameters

`in`	*socket_ind*	for detecting i2c address
`in`	*limit*	a uint32t that indicates the desired boostlimit value of the socket

Return values

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

5.8 Current, Min, Max TDP

Functions

- oob_status_t read_tdp (int socket_ind, uint32_t *ptdp)

  Get the Thermal Design Power limit TDP of the socket with provided socket index.
- oob_status_t read_max_tdp (int socket_ind, uint32_t *ptdp)

  Get the Maximum Thermal Design Power limit TDP of the socket with provided socket index.
- oob_status_t read_min_tdp (int socket_ind, uint32_t *ptdp)

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

5.8.1 Detailed Description

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

5.8.2 Function Documentation

5.8.2.1 read_tdp()

```
oob_status_t read_tdp (
            int socket_ind,
            uint32_t * ptdp )
```

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

Given a socket index `socket_ind` and a pointer to a uint32_t `ptdp`, this function will get the current TDP (in milliwatts) to the uint32_t pointed to by `ptdp`.

Parameters

`in`	*socket_ind*	a socket index
`in,out`	*ptdp*	a pointer to uint32_t to which the Current TDP value will be copied

Return values

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

5.8.2.2 read_max_tdp()

```
oob_status_t read_max_tdp (
```

```
            int socket_ind,
            uint32_t * ptdp )
```

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

Given a socket index `socket_ind` and a pointer to a uint32_t `ptdp`, this function will get the Maximum TDP (in milliwatts) to the uint32_t pointed to by `ptdp`.

Parameters

`in`	*socket_ind*	a socket index
`in,out`	*ptdp*	a pointer to uint32_t to which the Maximum TDP value will be copied

Return values

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

5.8.2.3 read_min_tdp()

```
oob_status_t read_min_tdp (
            int socket_ind,
            uint32_t * ptdp )
```

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

Given a socket index `socket_ind` and a pointer to a uint32_t `ptdp`, this function will get the Minimum TDP (in milliwatts) to the uint32_t pointed to by `ptdp`.

Parameters

`in`	*socket_ind*	a socket index
`in,out`	*ptdp*	a pointer to uint32_t to which the Minimum TDP value will be copied

Return values

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

5.9 Prochot

Functions

- oob_status_t read_prochot_status (int socket_ind, uint32_t *pstatus)

  Get the Prochot Status of the socket with provided socket index.
- oob_status_t read_prochot_residency (int socket_ind, uint32_t *presi)

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

5.9.1 Detailed Description

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

5.9.2 Function Documentation

5.9.2.1 read_prochot_status()

```
oob_status_t read_prochot_status (
            int socket_ind,
            uint32_t * pstatus )
```

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

Given a socket index `socket_ind` and a pointer to a uint32_t `pstatus`, this function will get the Prochot status as active/1 or inactive/0 to the bool pointed to by `pstatus`.

Parameters

`in`	*socket_ind*	a socket index
`in,out`	*pstatus*	a pointer to uint32_t to which the Prochot status will be copied

Return values

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

5.9.2.2 read_prochot_residency()

```
oob_status_t read_prochot_residency (
            int socket_ind,
            uint32_t * presi )
```

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

Given a socket index `socket_ind` and a pointer to a uint32_t `presi`, this function will get the Prochot residency as a percentage pointed to by `presi`.

Parameters

`in`	*socket_ind*	a socket index
`in,out`	*presi*	a pointer to uint32_t to which the Prochot residency will be copied

Return values

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

5.10 Dram and other features Query

Functions

- oob_status_t read_dram_throttle (int socket_ind, uint32_t *buffer)

  Read Dram Throttle will always read the lowest percentage value.
- oob_status_t write_dram_throttle (int socket_ind, uint32_t limit)

  Set Dram Throttle value in terms of percentage.
- oob_status_t read_vddio_mem_power (int socket_ind, uint32_t *buffer)

  Read VDDIOMem Power returns the estimated VDDIOMem power consumed within the socket.
- oob_status_t read_nbio_error_logging_register (int socket_ind, uint8_t quadrant, uint32_t offset, uint32_t *buffer)

  Read NBIO Error Logging Register.
- oob_status_t read_iod_bist (int socket_ind, uint32_t *buffer)

  Read IOD Bist status.
- oob_status_t read_ccd_bist_result (int socket_ind, 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 (int socket_ind, uint32_t input, uint32_t *buffer)

  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 (int socket_ind, uint32_t *buffer)

  Provides the socket's CPU core clock (CCLK) frequency limit from the most restrictive infrastructure limit at the time of the request.
- oob_status_t read_socket_c0_residency (int socket_ind, uint32_t *buffer)

  Provides the average C0 residency across all cores in the socket. 100% specifies that all enabled cores in the socket are runningin C0.

5.10.1 Detailed Description

5.10.2 Function Documentation

5.10.2.1 read_dram_throttle()

```
oob_status_t read_dram_throttle (
            int socket_ind,
            uint32_t * buffer )
```

Read Dram Throttle will always read the lowest percentage value.

Given a `socket_ind`, this function will read dram throttle.

Parameters

in	*socket_ind*	is a particular package in the system.
out	*buffer*	is to read the dram throttle in % (0 - 100).

Return values

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

5.10.2.2 write_dram_throttle()

```
oob_status_t write_dram_throttle (
             int socket_ind,
             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	*socket_ind*	is a given socket
in	*limit*	a uint32t that indicates the desired limit to write for a given socket

Return values

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

5.10.2.3 read_vddio_mem_power()

```
oob_status_t read_vddio_mem_power (
             int socket_ind,
             uint32_t * buffer )
```

Read VDDIOMem Power returns the estimated VDDIOMem power consumed within the socket.

Given a `socket_ind`, this function will read VDDIOMem Power.

Parameters

in	*socket_ind*	is a particular package in the system.
out	*buffer*	is to read VDDIOMem Power.

Return values

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

5.10.2.4 read_nbio_error_logging_register()

```
oob_status_t read_nbio_error_logging_register (
             int socket_ind,
             uint8_t quadrant,
             uint32_t offset,
             uint32_t * buffer )
```

Read NBIO Error Logging Register.

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

Parameters

`in`	*socket_ind*	is a particular package in the system.
`in`	*quadrant*	value is Quadrant[31:24] and register offset[23:0]
`in`	*offset*	value is Quadrant[31:24] and register offset[23:0]
`out`	*buffer*	is to read NBIOErrorLoggingRegiter(register value).

Return values

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

5.10.2.5 read_iod_bist()

```
oob_status_t read_iod_bist (
             int socket_ind,
             uint32_t * buffer )
```

Read IOD Bist status.

Given a `socket_ind`, this function will read IOD Bist result.

Parameters

`in`	*socket_ind*	is a particular package in the system.
`out`	*buffer*	is to read IODBistResult (0=Bist pass, 1= Bist fail).

Return values

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

5.10.2.6 read_ccd_bist_result()

```
oob_status_t read_ccd_bist_result (
            int socket_ind,
            uint32_t input,
            uint32_t * buffer )
```

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

Given a `socket_ind`, Logical CCD instance number as `input`, this function will read CCDBistResult.

Parameters

in	*socket_ind*	is a particular package in the system.
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.
None-zero	is returned upon failure.

5.10.2.7 read_ccx_bist_result()

```
oob_status_t read_ccx_bist_result (
            int socket_ind,
            uint32_t input,
            uint32_t * buffer )
```

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_ind`, Logical CCX instance number as `input`, this function will read CCXBistResult.

Parameters

in	*socket_ind*	is a particular package in the system.
in	*input*	is a Logical CCX instance number.
out	*buffer*	is to read CCXBistResult (L3pass, Core[n:0]Pass)

Return values

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

5.10.2.8 read_cclk_freq_limit()

```
oob_status_t read_cclk_freq_limit (
             int socket_ind,
             uint32_t * buffer )
```

Provides the socket's CPU core clock (CCLK) frequency limit from the most restrictive infrastructure limit at the time of the request.

Given a `socket_ind`, this function will read Frequency.

Parameters

`in`	*socket_ind*	is a particular package in the system.
`out`	*buffer*	is to read freequency[MHz]

Return values

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

5.10.2.9 read_socket_c0_residency()

```
oob_status_t read_socket_c0_residency (
             int socket_ind,
             uint32_t * buffer )
```

Provides the average C0 residency across all cores in the socket. 100% specifies that all enabled cores in the socket are runningin C0.

Given a `socket_ind`, this function will read Socket C0 residency[%].

Parameters

`in`	*socket_ind*	is a particular package in the system.
`out`	*buffer*	is to read Socket C0 residency[%].

Return values

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

5.11 SB_RMI Read Processor Register Access

Functions

- oob_status_t esmi_oob_read_msr (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 (
            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	*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

Functions

- oob_status_t esmi_oob_cpuid (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 (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 (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 (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 (uint32_t thread, uint32_t fn_eax, uint32_t fn_ecx, uint32_t *edx)
  Read edx register on CPUID functionality.

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 (
            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`	*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 (
            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`	*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 (
            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`	*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 (
             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`	*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 (
             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`	*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.

5.13 SB-RMI Register Read Byte Protocol

Functions

- oob_status_t read_sbrmi_revision (int socket, 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 (int socket, uint8_t *buffer)

  Read Control byte from SB_RMI register command.
- oob_status_t read_sbrmi_status (int socket, uint8_t *buffer)

  Read one byte of Status value from SB_RMI register command.
- oob_status_t read_sbrmi_readsize (int socket, 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 (int socket, uint8_t *buffer)

  Read one byte of Thread Status from SB_RMI register command.
- oob_status_t read_sbrmi_swinterrupt (int socket, 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 (int socket, uint8_t *buffer)

  This register indicates the maximum number of threads present.

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 (
            int socket,
            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`	*socket*	a socket index
`in,out`	*buffer*	a pointer to a uint8_t that indicates value to hold

Return values

OOB_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure. This value specifies the APML specification revision that the product is compliant to. 0x10 = 1.0x Revision.

5.14 SBTSI Register Read Byte Protocol

Functions

- oob_status_t read_sbtsi_cpuinttemp (int socket, int8_t ∗buffer)

  Read one byte from a given SB_TSI register with provided socket index and buffer to get the read data of a given command.
- oob_status_t read_sbtsi_status (int socket, int8_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 (int socket, int8_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 (int socket, int8_t ∗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 read_sbtsi_updateratehz (int socket, 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 (int socket, int8_t 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_updateratehz (int socket, 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 (int socket, int8_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 (int socket, int8_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 (int socket, int8_t ∗buffer)

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

  The value returns the decimal portion of the CPU temperature.
- oob_status_t read_sbtsi_cputempoffsethibyte (int socket, uint8_t ∗buffer)

  SBTSI::CpuTempOffInt and SBTSI::CpuTempOffDec combine to specify the CPU temperature offset.
- oob_status_t read_sbtsi_cputempoffsetdecimal (int socket, uint8_t ∗buffer)

  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 (int socket, uint8_t ∗buffer)

  This value specifies the decimal portion of the high temperature threshold.
- oob_status_t read_sbtsi_lotempdecimal (int socket, uint8_t ∗buffer)

  value specifies the decimal portion of the low temperature threshold.
- oob_status_t read_sbtsi_timeoutconfig (int socket, uint8_t ∗buffer)

  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 (int socket, int8_t ∗buffer)

  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 (int socket, int8_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_manufid (int socket, int8_t *buffer)

  Returns the AMD manufacture ID.
- oob_status_t read_sbtsi_revision (int socket, int8_t *buffer)

  Specifies the SBI temperature sensor interface revision.
- oob_status_t sbtsi_get_cputemp (int socket, float *temp_value)

  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 (int socket, 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 (int socket, 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_tsi_config (int socket, int value, int check)

  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 (int socket, uint8_t *timeout)

  To verify if timeout support enabled or disabled.
- oob_status_t sbtsi_set_timeout_config (int socket, int value)

  To enable/disable timeout support.
- oob_status_t sbtsi_set_hightemp_threshold (int socket, int temp_int, float temp_dec)

  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_lowtemp_threshold (int socket, int temp_int, float temp_dec)

  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_htemp_threshold (int socket, int8_t *integer, float *decimal)

  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_ltemp_threshold (int socket, int8_t *integer, float *decimal)

  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 (int socket, float *temp_offset)

  SBTSI::CpuTempOffInt and SBTSI::CpuTempOffDec combine to specify the CPU temperature offset.
- oob_status_t write_sbtsi_cputempoffset (uint32_t socket, float temp_offset)

  SBTSI::CpuTempOffInt and SBTSI::CpuTempOffDec combine to set the CPU temperature offset.
- oob_status_t sbtsi_set_alert_threshold (int socket, int value)

  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 (int socket, int value)

  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 (
            int socket,
            int8_t * buffer )
```

Read one byte from a given SB_TSI register with provided socket index and buffer to get the read data of a given command.

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_TSI register.

Parameters

`in`	*socket*	a socket index
`in,out`	*buffer*	a pointer to a int8_t that indicates value to hold

Return values

OOB_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure. integer CPU temperature value The CPU temperature is calculated by adding the CPU temperature offset(SBTSI::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`	*socket*	a socket index
`in,out`	*buffer*	a pointer to hold the cpu temperature

Return values

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

5.14.2.2 read_sbtsi_status()

```
oob_status_t read_sbtsi_status (
            int socket,
            int8_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	*socket*	a socket index
in,out	*buffer*	a pointer to hold the cpu temperature

Return values

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

5.14.2.3 read_sbtsi_config()

```
oob_status_t read_sbtsi_config (
             int socket,
             int8_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	*socket*	a socket index
in,out	*buffer*	a pointer to hold the cpu temperature

Return values

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

5.14.2.4 read_sbtsi_updaterate()

```
oob_status_t read_sbtsi_updaterate (
             int socket,
             int8_t * 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	*socket*	a socket index
in,out	*buffer*	a pointer to hold the cpu temperature

Return values

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

5.14.2.5 read_sbtsi_updateratehz()

```
oob_status_t read_sbtsi_updateratehz (
             int socket,
             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	*socket*	a socket index
in,out	*buffer*	a pointer to hold the cpu temperature

Return values

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

5.14.2.6 write_sbtsi_updaterate()

```
oob_status_t write_sbtsi_updaterate (
             int socket,
             int8_t 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	*socket*	a socket index
in	*buffer*	value to write in raw format

Return values

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

5.14.2.7 write_sbtsi_updateratehz()

```
oob_status_t write_sbtsi_updateratehz (
            int socket,
            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	*socket*	a socket index
in	*uprate*	value to write in raw format

Return values

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

5.14.2.8 read_sbtsi_hitempint()

```
oob_status_t read_sbtsi_hitempint (
            int socket,
            int8_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	*socket*	a socket index
in,out	*buffer*	a pointer to hold the cpu temperature

Return values

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

5.14.2.9 read_sbtsi_lotempint()

```
oob_status_t read_sbtsi_lotempint (
            int socket,
            int8_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	*socket*	a socket index
in,out	*buffer*	a pointer to hold the cpu temperature

Return values

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

5.14.2.10 read_sbtsi_configwrite()

```
oob_status_t read_sbtsi_configwrite (
             int socket,
             int8_t * buffer )
```

This register provides write access to SBTSI::Config.

Parameters

in	*socket*	a socket index
in,out	*buffer*	a pointer to hold the cpu temperature

Return values

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

5.14.2.11 read_sbtsi_cputempdecimal()

```
oob_status_t read_sbtsi_cputempdecimal (
             int socket,
             uint8_t * buffer )
```

The value returns the decimal portion of the CPU temperature.

Parameters

in	*socket*	a socket index
in,out	*buffer*	a pointer to hold the cpu temperature

Return values

OOB_SUCCESS	is returned upon successful call.

Return values

None-zero	is returned upon failure.

5.14.2.12 read_sbtsi_cputempoffsethibyte()

```
oob_status_t read_sbtsi_cputempoffsethibyte (
             int socket,
             uint8_t * buffer )
```

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

Parameters

in	*socket*	a socket index
in,out	*buffer*	a pointer to hold the cpu temperature

Return values

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

5.14.2.13 read_sbtsi_cputempoffsetdecimal()

```
oob_status_t read_sbtsi_cputempoffsetdecimal (
             int socket,
             uint8_t * buffer )
```

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

Parameters

in	*socket*	a socket index
in,out	*buffer*	a pointer to hold the cpu temperature

Return values

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

5.14.2.14 read_sbtsi_hitempdecimal()

```
oob_status_t read_sbtsi_hitempdecimal (
            int socket,
            uint8_t * buffer )
```

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

Parameters

in	*socket*	a socket index
in,out	*buffer*	a pointer to hold the cpu temperature

Return values

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

5.14.2.15 read_sbtsi_lotempdecimal()

```
oob_status_t read_sbtsi_lotempdecimal (
            int socket,
            uint8_t * buffer )
```

value specifies the decimal portion of the low temperature threshold.

Parameters

in	*socket*	a socket index
in,out	*buffer*	a pointer to hold the cpu temperature

Return values

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

5.14.2.16 read_sbtsi_timeoutconfig()

```
oob_status_t read_sbtsi_timeoutconfig (
            int socket,
            uint8_t * buffer )
```

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	*socket*	a socket index
in,out	*buffer*	a pointer to hold the cpu temperature

Return values

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

5.14.2.17 read_sbtsi_alertthreshold()

```
oob_status_t read_sbtsi_alertthreshold (
            int socket,
            int8_t * buffer )
```

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	*socket*	a socket index
in,out	*buffer*	a pointer to hold the cpu temperature

Return values

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

5.14.2.18 read_sbtsi_alertconfig()

```
oob_status_t read_sbtsi_alertconfig (
            int socket,
            int8_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	*socket*	a socket index
in,out	*buffer*	a pointer to hold the cpu temperature

Return values

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

5.14.2.19 read_sbtsi_manufid()

```
oob_status_t read_sbtsi_manufid (
            int socket,
            int8_t * buffer )
```

Returns the AMD manufacture ID.

Parameters

in	*socket*	a socket index
in,out	*buffer*	a pointer to hold the cpu temperature

Return values

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

5.14.2.20 read_sbtsi_revision()

```
oob_status_t read_sbtsi_revision (
            int socket,
            int8_t * buffer )
```

Specifies the SBI temperature sensor interface revision.

Parameters

in	*socket*	a socket index
in,out	*buffer*	a pointer to hold the cpu temperature

Return values

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

5.14.2.21 sbtsi_get_cputemp()

```
oob_status_t sbtsi_get_cputemp (
            int socket,
            float * temp_value )
```

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

Parameters

in	*socket*	a socket index
in,out	*temp_value*	temperature of the CPU

Return values

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

5.14.2.22 sbtsi_get_temp_status()

```
oob_status_t sbtsi_get_temp_status (
            int socket,
            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	*socket*	a 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.
None-zero	is returned upon failure.

5.14.2.23 sbtsi_get_config()

```
oob_status_t sbtsi_get_config (
```

```
            int socket,
            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	*socket*	a 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.
None-zero	is returned upon failure.

5.14.2.24 sbtsi_set_tsi_config()

```
oob_status_t sbtsi_set_tsi_config (
            int socket,
            int value,
            int check )
```

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	*socket*	a socket index
in	*value*	value to update 0 or 1
in	*check*	which bit need to update

Return values

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

5.14.2.25 sbtsi_get_timeout()

```
oob_status_t sbtsi_get_timeout (
            int socket,
            uint8_t * timeout )
```

To verify if timeout support enabled or disabled.

Parameters

in	*socket*	a socket index
in,out	*timeout*	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.
None-zero	is returned upon failure.

5.14.2.26 sbtsi_set_timeout_config()

```
oob_status_t sbtsi_set_timeout_config (
            int socket,
            int value )
```

To enable/disable timeout support.

Parameters

in	*socket*	a socket index
in	*value*	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.
None-zero	is returned upon failure.

5.14.2.27 sbtsi_set_hightemp_threshold()

```
oob_status_t sbtsi_set_hightemp_threshold (
            int socket,
```

```
            int temp_int,
            float temp_dec )
```

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	*socket*	a socket index
in	*temp_int*	Specifies the integer part of threshold
in	*temp_dec*	Specifies the decimal part of threshold

Return values

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

5.14.2.28 sbtsi_set_lowtemp_threshold()

```
oob_status_t sbtsi_set_lowtemp_threshold (
            int socket,
            int temp_int,
            float temp_dec )
```

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	*socket*	a socket index
in	*temp_int*	Specifies the integer part of threshold
in	*temp_dec*	Specifies the decimal part of threshold

Return values

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

5.14.2.29 sbtsi_get_htemp_threshold()

```
oob_status_t sbtsi_get_htemp_threshold (
            int socket,
            int8_t * integer,
            float * decimal )
```

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	*socket*	a socket index
in,out	*integer*	Specifies the integer part of threshold
in,out	*decimal*	Specifies the decimal part of threshold

Return values

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

5.14.2.30 sbtsi_get_ltemp_threshold()

```
oob_status_t sbtsi_get_ltemp_threshold (
            int socket,
            int8_t * integer,
            float * decimal )
```

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	*socket*	a socket index
in,out	*integer*	Specifies the integer part of threshold
in,out	*decimal*	Specifies the decimal part of threshold

Return values

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

5.14.2.31 read_sbtsi_cputempoffset()

```
oob_status_t read_sbtsi_cputempoffset (
            int socket,
            float * temp_offset )
```

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

Parameters

in	*socket*	a socket index
in,out	*temp_offset*	to get the offset value for temperature

Return values

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

5.14.2.32 write_sbtsi_cputempoffset()

```
oob_status_t write_sbtsi_cputempoffset (
            uint32_t socket,
            float temp_offset )
```

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

Parameters

in	*socket*	a socket index
in	*temp_offset*	to set the offset value for temperature

Return values

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

5.14.2.33 sbtsi_set_alert_threshold()

```
oob_status_t sbtsi_set_alert_threshold (
            int socket,
            int value )
```

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	*socket*	a socket index
in	*value*	Number of samples 0h: 1 sample 6h-1h: (value + 1) sample 7h: 8 sample

Return values

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

5.14.2.34 sbtsi_set_alert_config()

```
oob_status_t sbtsi_set_alert_config (
            int socket,
            int value )
```

Alert comparator mode enable.

Parameters

in	*socket*	a socket index
in	*value*	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.
None-zero	is returned upon failure.

Chapter 6

Data Structure Documentation

6.1 sbrmi_indata Struct Reference

SB-RMI Read Proccessor Register command protocol and read CPUID command protocol.

`#include <esmi_cpuid_msr.h>`

Data Fields

- uint8_t cmd
  Read CPUID/Read Register Command Format is 0x73.
- uint8_t wr_ln
  0x8 bytes is WrDataLen.
- uint8_t rd_ln
- uint8_t regcmd
  read CPUID command is 0x91
- uint8_t thread
  bit 0 is reserved, bit 1:7 selects the 0x127 threads)
- union {
  uint32_t value
  value
  uint8_t reg [4]
  Register address or CPUID function.
  } __attribute__

  maximum 4 register address for CPUID function data to hold
- uint8_t ecx
  1b = Return edx:ecx.

6.1.1 Detailed Description

SB-RMI Read Proccessor Register command protocol and read CPUID command protocol.

I2C/SMBUS Input message packet data format for SB-RMI Read Processor Register Command and CPUID command Protocol.

6.1.2 Field Documentation

6.1.2.1 cmd

```
uint8_t sbrmi_indata::cmd
```

Read CPUID/Read Register Command Format is 0x73.

command protocol

6.1.2.2 rd_ln

```
uint8_t sbrmi_indata::rd_ln
```

Number of bytes to read from register, Valid values are 0x1 through 0x8. 0x8 bytes Number of CPUID bytes to read.

6.1.2.3 regcmd

```
uint8_t sbrmi_indata::regcmd
```

read CPUID command is 0x91

Read Processor Register command is 0x86

6.1.2.4 ecx

```
uint8_t sbrmi_indata::ecx
```

1b = Return edx:ecx.

0b = Return ebx:eax

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

- esmi_cpuid_msr.h

6.2 sbrmi_outdata Struct Reference

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

Data Fields

- uint8_t num_bytes

  Number of bytes returned = rd_ln + 1.
- uint8_t status

  status code
-

union {
  uint64_t value
    8bytes, [4,4] bytes of [eax, ebx] or [ecx, edx]
  uint8_t bytes [8]
    RdData 1 to RdData 8>
};

6.2.1 Detailed Description

I2C/SMBUS message Output message poacket data format for SB-RMI Read Processor Register Command and CPUID command Protocol for Output data.

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

- esmi_cpuid_msr.h

Chapter 7

File Documentation

7.1 esmi_common.h File Reference

Macros

- #define TOTAL_SOCKETS 2

  Total number of sockets in the system.
- #define FILEPATHSIZ 128

  Buffer to hold size of file path.
- #define P0_RMI_ADDR 0x3c

  I2C slave address for SB-RMI on socket P0 on SSP.
- #define P1_RMI_ADDR 0x38

  I2C slave address for SB-RMI on socket P1 on SSP.
- #define P0_TSI_ADDR 0x4c

  I2C slave address for SB-TSI on socket P0 on SSP.
- #define P1_TSI_ADDR 0x48

  I2C slave address for SB-TSI on socket P1 on SSP.

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_NO_I2C_ADDR, OOB_RD_LENGTH_ERR, OOB_RMI_STATUS_ERR, OOB_INVALID_INPUT }

  Error codes retured by ESMI_OOB_COMMON functions.

Functions

- oob_status_t esmi_oob_init (int i2c_channel)

  maintain the file descriptor of i2c device.
- void esmi_oob_exit (void)

  Closes the i2c channel device which was opened during init.
- oob_status_t errno_to_oob_status (int err)

  convert linux error to esmi error.

- uint32_t esmi_get_logical_cores_per_socket (void)

  Get the number of logical cores per socket.
- uint32_t esmi_get_threads_per_socket (void)

  Get the number of threads per socket.
- uint32_t esmi_get_threads_per_core (void)

  Get number of threads per core.
- char * esmi_get_err_msg (oob_status_t oob_err)

  Get the error string message for esmi oob errors.

7.1.1 Detailed Description

Header file for the ESMI-OOB library common functions. use of this library is to init the functionality and exit after use.

This header file has init and exit functionalities to open and close the particular i2c channel.

7.1.2 Enumeration Type Documentation

7.1.2.1 oob_status_t

```
enum oob_status_t
```

Error codes retured by ESMI_OOB_COMMON 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
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	ESMI-OOB object not initialized.
OOB_TRY_AGAIN	No match Try again.
OOB_NO_I2C_ADDR	i2c address not available
OOB_RD_LENGTH_ERR	read bytes from cpuid or msr failed
OOB_RMI_STATUS_ERR	cpuid or msr read status failed
OOB_INVALID_INPUT	Input value is invalid.

7.2 esmi_cpuid_msr.h File Reference

```
#include "esmi_common.h"
```

Data Structures

- struct sbrmi_indata

  SB-RMI Read Proccessor Register command protocol and read CPUID command protocol.
- struct sbrmi_outdata

Functions

- struct sbrmi_indata __attribute__ ((packed)) rmi_indata

  SB-RMI Read Proccessor Register command protocol and read CPUID command protocol.
- oob_status_t esmi_oob_read_msr (uint32_t thread, uint32_t msraddr, uint64_t *buffer)

  Read the MCA MSR register for a given thread.
- oob_status_t esmi_oob_cpuid (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 (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 (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 (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 (uint32_t thread, uint32_t fn_eax, uint32_t fn_ecx, uint32_t *edx)

  Read edx register on CPUID functionality.

Variables

- uint8_t cmd

  Read CPUID/Read Register Command Format is 0x73.
- uint8_t wr_ln

  0x8 bytes is WrDataLen.
- uint8_t rd_ln
- uint8_t regcmd

  read CPUID command is 0x91
- uint8_t thread

  bit 0 is reserved, bit 1:7 selects the 0x127 threads)
- union {
  uint32_t value
  value
  uint8_t reg [4]
  Register address or CPUID function.
  };

  maximum 4 register address for CPUID function data to hold
- uint8_t ecx

1b = Return edx:ecx.

- uint8_t num_bytes

  Number of bytes returned = rd_ln + 1.

- uint8_t status

  status code

-

union {
  uint64_t value
    8bytes, [4,4] bytes of [eax, ebx] or [ecx, edx]
  uint8_t bytes [8]
    RdData 1 to RdData 8>
};

7.2.1 Detailed Description

Header file for the ESMI-OOB 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 E-SMI-OOB library. Description of the API, arguments and return values. The Error codes returned by the API.

7.2.2 Function Documentation

7.2.2.1 __attribute__()

```
struct sbrmi_outdata __attribute__ (
            (packed)  )
```

SB-RMI Read Proccessor Register command protocol and read CPUID command protocol.

I2C/SMBUS Input message packet data format for SB-RMI Read Processor Register Command and CPUID command Protocol.

I2C/SMBUS message Output message poacket data format for SB-RMI Read Processor Register Command and CPUID command Protocol for Output data.

7.2.3 Variable Documentation

7.2.3.1 cmd

```
uint8_t cmd
```

Read CPUID/Read Register Command Format is 0x73.

command protocol

7.2.3.2 rd_ln

```
uint8_t rd_ln
```

Number of bytes to read from register, Valid values are 0x1 through 0x8. 0x8 bytes Number of CPUID bytes to read.

7.2.3.3 regcmd

```
uint8_t regcmd
```

read CPUID command is 0x91

Read Processor Register command is 0x86

7.2.3.4 value

```
uint64_t value
```

value

8bytes, [4,4] bytes of [eax, ebx] or [ecx, edx]

7.2.3.5 ecx

```
uint8_t ecx
```

1b = Return edx:ecx.

0b = Return ebx:eax

7.3 esmi_mailbox.h File Reference

```
#include "esmi_common.h"
```

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_VD↩DIO_MEM_POWER**,
  READ_NBIO_ERROR_LOGGING_REGISTER, **READ_IOD_BIST** = 0x13, **READ_CCD_BIST_RESULT**, **READ_CCX_BIST_RESULT**,
  READ_PACKAGE_CCLK_FREQ_LIMIT, **READ_PACKAGE_C0_RESIDENCY** }

  Mailbox message types defined in the E-SMI OOB library.

Functions

- oob_status_t read_socket_power (int socket_ind, uint32_t *ppower)

  Get the average power consumption of the socket with provided socket index.
- oob_status_t read_socket_power_limit (int socket_ind, uint32_t *pcap)

  Get the current power cap/limit value for a given socket.
- oob_status_t read_max_socket_power_limit (int socket_ind, uint32_t *pmax)

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

  Set the power cap/limit value for a given socket.
- oob_status_t read_esb_boost_limit (int socket_ind, uint32_t cpu_ind, uint32_t *pboostlimit)

  Get the Out-of-band boostlimit value for a given core.
- oob_status_t read_bios_boost_fmax (int socket_ind, uint32_t value, uint32_t *buffer)

  Get the In-band maximum boostlimit value for a given core.
- oob_status_t write_esb_boost_limit (int socket_ind, int cpu_id, uint32_t limit)

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

  Set the boostlimit value for the whole socket (whole system).
- oob_status_t read_tdp (int socket_ind, uint32_t *ptdp)

  Get the Thermal Design Power limit TDP of the socket with provided socket index.
- oob_status_t read_max_tdp (int socket_ind, uint32_t *ptdp)

  Get the Maximum Thermal Design Power limit TDP of the socket with provided socket index.
- oob_status_t read_min_tdp (int socket_ind, uint32_t *ptdp)

  Get the Minimum Thermal Design Power limit TDP of the socket with provided socket index.
- oob_status_t read_prochot_status (int socket_ind, uint32_t *pstatus)

  Get the Prochot Status of the socket with provided socket index.
- oob_status_t read_prochot_residency (int socket_ind, uint32_t *presi)

  Get the Prochot Residency (since the boot time or last read of Prochot Residency) of the socket with provided socket index.
- oob_status_t read_dram_throttle (int socket_ind, uint32_t *buffer)

  Read Dram Throttle will always read the lowest percentage value.
- oob_status_t write_dram_throttle (int socket_ind, uint32_t limit)

  Set Dram Throttle value in terms of percentage.
- oob_status_t read_vddio_mem_power (int socket_ind, uint32_t *buffer)

  Read VDDIOMem Power returns the estimated VDDIOMem power consumed within the socket.
- oob_status_t read_nbio_error_logging_register (int socket_ind, uint8_t quadrant, uint32_t offset, uint32_t *buffer)

  Read NBIO Error Logging Register.
- oob_status_t read_iod_bist (int socket_ind, uint32_t *buffer)

  Read IOD Bist status.
- oob_status_t read_ccd_bist_result (int socket_ind, 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 (int socket_ind, uint32_t input, uint32_t *buffer)

  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 (int socket_ind, uint32_t *buffer)

  Provides the socket's CPU core clock (CCLK) frequency limit from the most restrictive infrastructure limit at the time of the request.
- oob_status_t read_socket_c0_residency (int socket_ind, uint32_t *buffer)

  Provides the average C0 residency across all cores in the socket. 100% specifies that all enabled cores in the socket are runningin C0.

7.3.1 Detailed Description

Header file for the Mailbox messages supported by E-SMI OOB 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 E-SMI OOB library. Description of the API, arguments and return values. The Error codes returned by the API.

7.4 esmi_rmi.h File Reference

```
#include "esmi_common.h"
```

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 E-SMI-OOB mailbox functions.
- enum sbrmi_registers {
  SBRMI_REVISION = 0x0, **SBRMI_CONTROL**, **SBRMI_STATUS**, **SBRMI_READSIZE**,
  SBRMI_THREADENABLESTATUS, **SBRMI_SOFTWAREINTERRUPT** = 0x40, **SBRMI_THREADNUMBER** }

  SB-RMI(Side-Band Remote Management Interface) features register access.

Functions

- oob_status_t read_sbrmi_revision (int socket, 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 (int socket, uint8_t ∗buffer)

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

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

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

  This register indicates the maximum number of threads present.

7.4.1 Detailed Description

Header file for the E-SMI-OOB 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 E-SMI-OOB library. Description of the API, arguments and return values. The Error codes returned by the API.

7.5 esmi_tsi.h File Reference

```
#include "esmi_common.h"
```

Macros

- #define TEMP_ENC 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_CPU_INT_TEMP = 0x1, **SBTSI_STATUS**, **SBTSI_CONFIGURATION**, **SBTSI_UPDATERATE**,
  SBTSI_HITEMPINT = 0x7, **SBTSI_LOTEMPINT**, **SBTSI_CONFIGWR**, **SBTSI_CPUTEMPDECIMAL** = 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.

Functions

- oob_status_t read_sbtsi_cpuinttemp (int socket, int8_t *buffer)

  Read one byte from a given SB_TSI register with provided socket index and buffer to get the read data of a given command.
- oob_status_t read_sbtsi_status (int socket, int8_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 (int socket, int8_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 (int socket, int8_t *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 read_sbtsi_updateratehz (int socket, 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 (int socket, int8_t 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_updateratehz (int socket, 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 (int socket, int8_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 (int socket, int8_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 (int socket, int8_t *buffer)

  This register provides write access to SBTSI::Config.

- oob_status_t read_sbtsi_cputempdecimal (int socket, uint8_t *buffer)

  The value returns the decimal portion of the CPU temperature.

- oob_status_t read_sbtsi_cputempoffsethibyte (int socket, uint8_t *buffer)

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

- oob_status_t read_sbtsi_cputempoffsetdecimal (int socket, uint8_t *buffer)

  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 (int socket, uint8_t *buffer)

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

- oob_status_t read_sbtsi_lotempdecimal (int socket, uint8_t *buffer)

  value specifies the decimal portion of the low temperature threshold.

- oob_status_t read_sbtsi_timeoutconfig (int socket, uint8_t *buffer)

  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 (int socket, int8_t *buffer)

  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 (int socket, int8_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_manufid (int socket, int8_t *buffer)

  Returns the AMD manufacture ID.

- oob_status_t read_sbtsi_revision (int socket, int8_t *buffer)

  Specifies the SBI temperature sensor interface revision.

- oob_status_t sbtsi_get_cputemp (int socket, float *temp_value)

  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 (int socket, 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 (int socket, 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_tsi_config (int socket, int value, int check)

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 (int socket, uint8_t *timeout)

  To verify if timeout support enabled or disabled.

- oob_status_t sbtsi_set_timeout_config (int socket, int value)

  To enable/disable timeout support.

- oob_status_t sbtsi_set_hightemp_threshold (int socket, int temp_int, float temp_dec)

  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_lowtemp_threshold (int socket, int temp_int, float temp_dec)

  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_htemp_threshold (int socket, int8_t *integer, float *decimal)

  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_ltemp_threshold (int socket, int8_t *integer, float *decimal)

  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 (int socket, float *temp_offset)

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

- oob_status_t write_sbtsi_cputempoffset (uint32_t socket, float temp_offset)

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

- oob_status_t sbtsi_set_alert_threshold (int socket, int value)

  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 (int socket, int value)

  Alert comparator mode enable.

7.5.1 Detailed Description

Header file for the E-SMI-OOB 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 E-SMI-OOB library. Description of the API, arguments and return values. The Error codes returned by the API.

Index