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

Generated by Doxygen 1.8.13

Contents

1 EPYC™ System Management Interface Out-of-band (E-SMI-OOB) Library **1**

2 Module Index **7**

2.1 Modules 7

3 Data Structure Index **9**

3.1 Data Structures 9

4 File Index **11**

4.1 File List 11

5 Module Documentation **13**

5.1 Auxiliary functions 13

5.1.1 Detailed Description 13

5.1.2 Function Documentation 13

5.1.2.1 errno_to_oob_status() 13

5.1.2.2 esmi_get_err_msg() 14

5.2 SB-RMI Mailbox Service 15

5.2.1 Detailed Description 15

5.3 Power Monitor 16

5.3.1 Detailed Description 16

5.3.2 Function Documentation 16

5.3.2.1 read_socket_power() 16

5.3.2.2 read_socket_power_limit() 16

5.3.2.3 read_max_socket_power_limit() 17

5.4 Power Control 18
5.4.1 Detailed Description 18
5.4.2 Function Documentation 18
5.4.2.1 write_socket_power_limit() 18
5.5 Performance (Boost limit) Monitor 19
5.5.1 Detailed Description 19
5.5.2 Function Documentation 19
5.5.2.1 read_esb_boost_limit() 19
5.5.2.2 read_bios_boost_fmax() 19
5.6 Out-of-band Performance (Boost limit) Control 21
5.6.1 Detailed Description 21
5.6.2 Function Documentation 21
5.6.2.1 write_esb_boost_limit() 21
5.6.2.2 write_esb_boost_limit_allcores() 21
5.7 Current, Min, Max TDP 23
5.7.1 Detailed Description 23
5.7.2 Function Documentation 23
5.7.2.1 read_tdp() 23
5.7.2.2 read_max_tdp() 23
5.7.2.3 read_min_tdp() 24
5.8 Prochot 25
5.8.1 Detailed Description 25
5.8.2 Function Documentation 25
5.8.2.1 read_prochot_status() 25
5.8.2.2 read_prochot_residency() 25
5.9 Dram and other features Query 27
5.9.1 Detailed Description 27
5.9.2 Function Documentation 27
5.9.2.1 read_dram_throttle() 27
5.9.2.2 write_dram_throttle() 28

5.9.2.3 read_vddio_mem_power() 28
5.9.2.4 read_nbio_error_logging_register() 29
5.9.2.5 read_iod_bist() 29
5.9.2.6 read_ccd_bist_result() 31
5.9.2.7 read_ccx_bist_result() 31
5.9.2.8 read_cclk_freq_limit() 32
5.9.2.9 read_socket_c0_residency() 32
5.9.2.10 read_ddr_bandwidth() 33
5.10 SB_RMI Read Processor Register Access 34
5.10.1 Detailed Description 34
5.10.2 Function Documentation 34
5.10.2.1 esmi_oob_read_msr() 34
5.11 SB-RMI CPUID Register Access 35
5.11.1 Detailed Description 35
5.11.2 Function Documentation 35
5.11.2.1 esmi_oob_cpuid() 35
5.11.2.2 esmi_oob_cpuid_eax() 36
5.11.2.3 esmi_oob_cpuid_ebx() 36
5.11.2.4 esmi_oob_cpuid_ecx() 37
5.11.2.5 esmi_oob_cpuid_edx() 38
5.12 SB-RMI Register Read Byte Protocol 39
5.12.1 Detailed Description 39
5.12.2 Function Documentation 39
5.12.2.1 read_sbrmi_revision() 39
5.13 SBTSI Register Read Byte Protocol 41
5.13.1 Detailed Description 42
5.13.2 Function Documentation 42
5.13.2.1 read_sbtsi_cpuinttemp() 43
5.13.2.2 read_sbtsi_status() 43
5.13.2.3 read_sbtsi_config() 44

5.13.2.4 read_sbtsi_updaterate() 44
5.13.2.5 write_sbtsi_updaterate() 45
5.13.2.6 read_sbtsi_hitempint() 45
5.13.2.7 read_sbtsi_lotempint() 46
5.13.2.8 read_sbtsi_configwrite() 46
5.13.2.9 read_sbtsi_cputempdecimal() 47
5.13.2.10 read_sbtsi_cputempoffint() 47
5.13.2.11 read_sbtsi_cputempoffdec() 48
5.13.2.12 read_sbtsi_hitempdecimal() 48
5.13.2.13 read_sbtsi_lotempdecimal() 49
5.13.2.14 read_sbtsi_timeoutconfig() 49
5.13.2.15 read_sbtsi_alertthreshold() 49
5.13.2.16 read_sbtsi_alertconfig() 50
5.13.2.17 read_sbtsi_manufid() 50
5.13.2.18 read_sbtsi_revision() 52
5.13.2.19 sbtsi_get_cputemp() 52
5.13.2.20 sbtsi_get_temp_status() 53
5.13.2.21 sbtsi_get_config() 53
5.13.2.22 sbtsi_set_configwr() 54
5.13.2.23 sbtsi_get_timeout() 54
5.13.2.24 sbtsi_set_timeout_config() 55
5.13.2.25 sbtsi_set_hitemp_threshold() 55
5.13.2.26 sbtsi_set_lotemp_threshold() 56
5.13.2.27 sbtsi_get_hitemp_threshold() 56
5.13.2.28 sbtsi_get_lotemp_threshold() 58
5.13.2.29 read_sbtsi_cputempoffset() 58
5.13.2.30 write_sbtsi_cputempoffset() 59
5.13.2.31 sbtsi_set_alert_threshold() 59
5.13.2.32 sbtsi_set_alert_config() 60

6 Data Structure Documentation 61

6.1 processor_info Struct Reference 61

6.1.1 Detailed Description 61

6.2 sbrmi_indata Struct Reference 61

6.2.1 Detailed Description 62

6.2.2 Field Documentation 62

6.2.2.1 cmd 62

6.2.2.2 rd_ln 62

6.2.2.3 regcmd 63

6.2.2.4 ecx 63

6.3 sbrmi_outdata Struct Reference 63

6.3.1 Detailed Description 63

7 File Documentation 65

7.1 esmi_common.h File Reference 65

7.1.1 Detailed Description 65

7.1.2 Enumeration Type Documentation 66

7.1.2.1 oob_status_t 66

7.2 esmi_cpuid_msr.h File Reference 67

7.2.1 Detailed Description 69

7.2.2 Function Documentation 69

7.2.2.1 __attribute__() 69

7.2.2.2 esmi_get_vendor_id() 69

7.2.2.3 esmi_get_processor_info() 70

7.2.2.4 esmi_get_logical_cores_per_socket() 70

7.2.2.5 esmi_get_threads_per_socket() 71

7.2.2.6 esmi_get_threads_per_core() 71

7.2.3 Variable Documentation 72

7.2.3.1 cmd 72

7.2.3.2 rd_ln 72

7.2.3.3 regcmd 72

7.2.3.4 value 72

7.2.3.5 ecx 72

7.2.3.6 __attribute__ 72

7.3 esmi_mailbox.h File Reference 73

7.3.1 Detailed Description 74

7.4 esmi_rmi.h File Reference 74

7.4.1 Detailed Description 75

7.5 esmi_tsi.h File Reference 75

7.5.1 Detailed Description 77

Index 79

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)
- latex (pdfTeX 3.14159265-2.6-1.40.18)
- 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 for native build, as follows.

```
$ mkdir -p build
```

```
$ mkdir -p install
```

```
$ 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.

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
```

The ESMI_OOB Library depends on the libi2c-dev library, libi2c-dev package needs to be installed.

```
$sudo apt-get install libi2c-dev
```

Compilation steps

```
$ mkdir -p build
```

```
$ cd build
```

```
$ cmake -DCMAKE_TOOLCHAIN_FILE=../cross-[arch..].cmake <location of root of
E-SMI-OOB 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 libesmi_oob64.so.0 root@10.x.x.x:/usr/lib
```

```
$ scp esmi_oob_tool root@10.x.x.x:/usr/bin
```

NOTE: For cross compilation, cross-$ARCH.cmake file is provided for below Architectures:

- armhf
- aarch64

Disclaimer

- User may not be able to use this library when the i2c addresses are reserved, this is observed when TSI driver is loaded
- Input arguments like i2c address and bus number passed by the user are not validated. It might result in unreliable system behavior

Building the Documentation

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

```
$ make doc
```

The reference manual (ESMI_OOB_Manual.pdf), release notes (ESMI_OOB_Release_Notes.pdf) upon a successful build.

Usage Basics

Device Indices

Many of the functions in the library take I2C Bus and 7-bit address as index.

Hello E-SMI-OOB

Below is a simple "Hello World" type program that displays power of required socket.

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

int main(int argc, char **argv) {
    uint32_t power_avg = 0;
    uint32_t i2c_bus, i2c_addr;
    char *end;

    i2c_bus = atoi(argv[1]);
    i2c_addr = strtoul(argv[2], &end, 16);
    if (*end || !*argv[2]) {
        printf("Require a valid i2c_address in Hexa\n");
        return 0;
    }

    read_socket_power(i2c_bus, i2c_addr, &power_avg);
    printf(" Avg:%.03f, ", (double)power_avg/1000);

    return 0;
}
```

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.

In execution platform, user can cross-verfiy "i2c-dev" module is loaded or not, if not follow the below step:

`$ lsmod | grep i2c-dev`

If not loaded, load the module as below

`$ insmod /lib/modules/‘uname -r‘/kernel/drivers/i2c/i2c-dev.ko` **or**

`$ modprobe i2c-dev.ko`

Check I2C addresses are enumerated as below, if not i2c connection is at fault. Pass the I2C bus number connected to socket for RMI or TSI

```
$ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- 0c -- -- --
10: 10 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: 20 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- 3c -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- 4c -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

For 2p targets, additional I2C addresses are enumerated as:

```
$ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- 0c -- -- --
10: 10 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: 20 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- 38 -- -- -- 3c -- -- --
40: -- -- -- -- -- -- -- -- 48 -- -- -- 4c -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

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

```
$ ./esmi_oob_tool

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

Usage: ./esmi_oob_tool <i2c_bus> <i2c_addr>
Where:  i2c_bus : 0 or 1
        Eg,  i2c_addr : SB-RMI addresses:       0x3c for Socket0 and 0x38 for Socket1
                        SB-TSI addresses:       0x4c for Socket0 and 0x48 for Socket1

====================== 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 or –help flag:

```
$ ./esmi_oob_tool --help

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

Usage: ./esmi_oob_tool [Option<s>] SOURCES
Option<s>:
< MAILBOX COMMANDS >:
  -p, (--showpower)               [I2C_BUS][I2C_ADDR]                    Get Power for a given socket in
       Watts
  -t, (--showtdp)                 [I2C_BUS][I2C_ADDR]                    Get TDP for a given socket in
       Watts
  -s, (--setpowerlimit)           [I2C_BUS][I2C_ADDR][POWER]             Set powerlimit for a given socket
       in mWatts
  -c, (--showcclkfreqlimit)       [I2C_BUS][I2C_ADDR]                    Get cclk freqlimit for a given
       socket in MHz
  -r, (--showc0residency)         [I2C_BUS][I2C_ADDR]                    Show c0_residency for a given
       socket
  -b, (--showboostlimit)          [I2C_BUS][I2C_ADDR][THREAD]            Get APML and BIOS boostlimit for a
       given core index in MHz
  -d, (--setapmlboostlimit)       [I2C_BUS][I2C_ADDR][THREAD][BOOSTLIMIT]Set APML boostlimit for a given
       core in MHz
  -a, (--setapmlsocketboostlimit) [I2C_BUS][I2C_ADDR][BOOSTLIMIT]        Set APML boostlimit for all cores
       in a socket in MHz
  --showddrbandwidth              [I2C_BUS][I2C_ADDR]                    Show DDR Bandwidth of a system
  --set_and_verify_dramthrottle   [I2C_BUS][I2C_ADDR][0 to 80%]          Set DRAM THROTTLE for a given
       socket
< SB-RMI COMMANDS >:
  --showrmicommandregisters       [I2C_BUS][I2C_ADDR]                    Get values of SB-RMI reg commands
       for a given socket
< SB-TSI COMMANDS >:
  --showtsicommandregisters       [I2C_BUS][I2C_ADDR]                    Get values of SB-TSI reg commands
       for a given socket
  --set_verify_updaterate         [I2C_BUS][I2C_ADDR][Hz]                Set APML Frequency Update rate for
       a given socket
  --sethightempthreshold          [I2C_BUS][I2C_ADDR][TEMP(°C)]          Set APML High Temp Threshold
  --setlowtempthreshold           [I2C_BUS][I2C_ADDR][TEMP(°C)]          Set APML Low Temp Threshold
  --settempoffset                 [I2C_BUS][I2C_ADDR][VALUE]             Set APML CPU Temp Offset, VALUE =
       [-CPU_TEMP(°C), 127 °C]
  --settimeoutconfig              [I2C_BUS][I2C_ADDR][VALUE]             Set/Reset APML CPU timeout config,
       VALUE = 0 or 1
  --setalertthreshold             [I2C_BUS][I2C_ADDR][VALUE]             Set APML CPU alert threshold
       sample, VALUE = 1 to 8
  --setalertconfig                [I2C_BUS][I2C_ADDR][VALUE]             Set/Reset APML CPU alert config,
       VALUE = 0 or 1
  --setalertmask                  [I2C_BUS][I2C_ADDR][VALUE]             Set/Reset APML CPU alert mask,
       VALUE = 0 or 1
  --setrunstop                    [I2C_BUS][I2C_ADDR][VALUE]             Set/Reset APML CPU runstop, VALUE
       = 0 or 1
  --setreadorder                  [I2C_BUS][I2C_ADDR][VALUE]             Set/Reset APML CPU read order,
       VALUE = 0 or 1
  --setara                        [I2C_BUS][I2C_ADDR][VALUE]             Set/Reset APML CPU ARA, VALUE = 0
       or 1
  -h, (--help)                                                           Show this help message

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

Below is a sample usage to get the individual library functionality API's. User can pass arguments either any of the ways "./esmi_oob_tool -p [bus_num] [7 bit adress]" or "./esmi_oob_tool --showpower [bus_num] [7 bit address]"

```
1. $ ./esmi_oob_tool -p 1 0x3c

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

    --------------------------------------------
    | Power (Watts)          | 52.729          |
    | PowerLimit (Watts)     | 225.000         |
    | PowerLimitMax (Watts)  | 240.000         |
    --------------------------------------------

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

2. $ ./esmi_oob_tool --setpowerlimit 1 0x3c 200000

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

    Set i2c_addr[0x3c]/power_limit :          200.000 Watts successfully

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

3. $ ./esmi_oob_tool --showtsicommandregisters 1 0x4c

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

       ----------------------------------------------------------------

                          *** SB-TSI REGISTER SUMMARY ***
       ----------------------------------------------------------------
        _CPUTEMP                | 40.250 _C
    _HIGH_THRESHOLD_TEMP    | 70.000 _C
    _LOW_THRESHOLD_TEMP     | 0.000 _C
    _TSI_UPDATERATE         | 16.000 Hz
    _THRESHOLD_SAMPLE       | 1
    _TEMP_OFFSET            | 0.000 _C
    _STATUS                 | No Temp Alert
    _CONFIG                 |
            ALERT_L pin     | Enabled
            Runstop         | Comparison Enabled
            Atomic Rd order | Integer latches Decimal
            ARA response    | Enabled
    _TIMEOUT_CONFIG         | Enabled
    _TSI_ALERT_CONFIG       | Disabled
    _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:

Auxiliary functions 13
SB-RMI Mailbox Service 15
  Power Monitor 16
  Power Control 18
  Performance (Boost limit) Monitor 19
  Out-of-band Performance (Boost limit) Control 21
  Current, Min, Max TDP 23
  Prochot 25
  Dram and other features Query 27
SB_RMI Read Processor Register Access 34
SB-RMI CPUID Register Access 35
SB-RMI Register Read Byte Protocol 39
SBTSI Register Read Byte Protocol 41

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

processor_info
Read Proccessor Info 61
sbrmi_indata
SB-RMI Read Proccessor Register command protocol and read CPUID command protocol . . 61
sbrmi_outdata 63

Chapter 4

File Index

4.1 File List

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

esmi_common.h 65
esmi_cpuid_msr.h 67
esmi_mailbox.h 73
esmi_rmi.h 74
esmi_tsi.h 75

Chapter 5

Module Documentation

5.1 Auxiliary 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.

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

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

5.2.1 Detailed Description

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

5.3 Power Monitor

Functions

- oob_status_t read_socket_power (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t ∗buffer)

  Get the power consumption of the socket with provided i2c_bus and i2c_addr.
- oob_status_t read_socket_power_limit (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t ∗buffer)

  Get the current power cap/limit value for a given socket.
- oob_status_t read_max_socket_power_limit (uint32_t i2c_bus, uint32_t i2c_addr, 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 (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            uint32_t ∗ buffer )
```

Get the power consumption of the socket with provided i2c_bus and i2c_addr.

Given a i2c_bus and i2c_addr 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`	*i2c_bus*	is the Bus connected to the socket
`in`	*i2c_addr*	is the 7-bit socket address
`in,out`	*buffer*	a pointer to uint32_t value of power consumption

Return values

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

5.3.2.2 read_socket_power_limit()

```
oob_status_t read_socket_power_limit (
              uint32_t i2c_bus,
              uint32_t i2c_addr,
              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`	*i2c_bus*	is the Bus connected to the socket
`in`	*i2c_addr*	is the 7-bit socket address
`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.
None-zero	is returned upon failure.

5.3.2.3 read_max_socket_power_limit()

```
oob_status_t read_max_socket_power_limit (
              uint32_t i2c_bus,
              uint32_t i2c_addr,
              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`	*i2c_bus*	is the Bus connected to the socket
`in`	*i2c_addr*	is the 7-bit socket address
`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.
None-zero	is returned upon failure.

5.4 Power Control

Functions

- oob_status_t write_socket_power_limit (uint32_t i2c_bus, uint32_t i2c_addr, 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 (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             uint32_t limit )
```

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

This function will set the power cap/limit

Parameters

in	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in	*limit*	uint32_t that indicates the desired power cap/limit, in milliwatts

Return values

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

5.5 Performance (Boost limit) Monitor

Functions

- oob_status_t read_esb_boost_limit (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t value, uint32_t *buffer)

  Get the Out-of-band boostlimit value for a given core.
- oob_status_t read_bios_boost_fmax (uint32_t i2c_bus, uint32_t i2c_addr, 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 (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
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.
None-zero	is returned upon failure.

5.5.2.2 read_bios_boost_fmax()

```
oob_status_t read_bios_boost_fmax (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            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`	*i2c_bus*	is the Bus connected to the socket
`in`	*i2c_addr*	is the 7-bit socket address
`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.
None-zero	is returned upon failure.

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

Functions

- oob_status_t write_esb_boost_limit (uint32_t i2c_bus, uint32_t i2c_addr, int cpu_ind, uint32_t limit)

  Set the Out-of-band boostlimit value for a given core.
- oob_status_t write_esb_boost_limit_allcores (uint32_t i2c_bus, uint32_t i2c_addr, 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 (
              uint32_t i2c_bus,
              uint32_t i2c_addr,
              int 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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
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.
None-zero	is returned upon failure.

5.6.2.2 write_esb_boost_limit_allcores()

```
oob_status_t write_esb_boost_limit_allcores (
uint32_t i2c_bus,
uint32_t i2c_addr,
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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in	*limit*	a uint32_t that indicates the desired boostlimit value of the socket

Return values

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

5.7 Current, Min, Max TDP

Functions

- oob_status_t read_tdp (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t ∗buffer)

  Get the Thermal Design Power limit TDP of the socket with provided socket index.
- oob_status_t read_max_tdp (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t ∗buffer)

  Get the Maximum Thermal Design Power limit TDP of the socket with provided socket index.
- oob_status_t read_min_tdp (uint32_t i2c_bus, uint32_t i2c_addr, 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 (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            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`	*i2c_bus*	is the Bus connected to the socket
`in`	*i2c_addr*	is the 7-bit socket address
`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.
None-zero	is returned upon failure.

5.7.2.2 read_max_tdp()

```
oob_status_t read_max_tdp (
              uint32_t i2c_bus,
              uint32_t i2c_addr,
              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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
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.
None-zero	is returned upon failure.

5.7.2.3 read_min_tdp()

```
oob_status_t read_min_tdp (
              uint32_t i2c_bus,
              uint32_t i2c_addr,
              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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
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.
None-zero	is returned upon failure.

5.8 Prochot

Functions

- oob_status_t read_prochot_status (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t *buffer)

    Get the Prochot Status of the socket with provided socket index.
- oob_status_t read_prochot_residency (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t *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 (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            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`	*i2c_bus*	is the Bus connected to the socket
`in`	*i2c_addr*	is the 7-bit socket address
`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.
None-zero	is returned upon failure.

5.8.2.2 read_prochot_residency()

```
oob_status_t read_prochot_residency (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            uint32_t * 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`	*i2c_bus*	is the Bus connected to the socket
`in`	*i2c_addr*	is the 7-bit socket address
`in,out`	*buffer*	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.

5.9 Dram and other features Query

Functions

- oob_status_t read_dram_throttle (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t ∗buffer)

  Read Dram Throttle will always read the lowest percentage value.
- oob_status_t write_dram_throttle (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t limit)

  Set Dram Throttle value in terms of percentage.
- oob_status_t read_vddio_mem_power (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t ∗buffer)

  Read VDDIOMem Power returns the estimated VDDIOMem power consumed within the socket.
- oob_status_t read_nbio_error_logging_register (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t quadrant, uint32_t offset, uint32_t ∗buffer)

  Read NBIO Error Logging Register.
- oob_status_t read_iod_bist (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t ∗buffer)

  Read IOD Bist status.
- oob_status_t read_ccd_bist_result (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t value, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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.
- oob_status_t read_ddr_bandwidth (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t ∗max_bw, uint32_↩
t ∗utilized_bw, uint32_t ∗utilized_pct)

  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 (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             uint32_t ∗ buffer )
```

Read Dram Throttle will always read the lowest percentage value.

This function will read dram throttle.

Parameters

in	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
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.

5.9.2.2 write_dram_throttle()

```
oob_status_t write_dram_throttle (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
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.
None-zero	is returned upon failure.

5.9.2.3 read_vddio_mem_power()

```
oob_status_t read_vddio_mem_power (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             uint32_t * buffer )
```

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

This function will read VDDIOMem Power for the given socket

Parameters

in	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
out	*buffer*	is to read VDDIOMem Power.

Return values

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

5.9.2.4 read_nbio_error_logging_register()

```
oob_status_t read_nbio_error_logging_register (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            uint8_t quadrant,
            uint32_t offset,
            uint32_t * buffer )
```

Read NBIO Error Logging Register.

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

Parameters

in	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in	*quadrant*	value is Quadrant[31:24] from NBIO register
in	*offset*	value is register offset[23:0] from NBIO register
out	*buffer*	is to read NBIOErrorLoggingRegiter(register value).

Return values

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

5.9.2.5 read_iod_bist()

```
oob_status_t read_iod_bist (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            uint32_t * buffer )
```

Read IOD Bist status.

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

Parameters

in	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
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.

5.9.2.6 read_ccd_bist_result()

```
oob_status_t read_ccd_bist_result (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
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.

5.9.2.7 read_ccx_bist_result()

```
oob_status_t read_ccx_bist_result (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            uint32_t value,
            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 bus number, address, Logical CCX instance number as `input`, this function will read CCXBist↩ Result.

Parameters

`in`	*i2c_bus*	is the Bus connected to the socket
`in`	*i2c_addr*	is the 7-bit socket address
`in`	*value*	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.

5.9.2.8 read_cclk_freq_limit()

```
oob_status_t read_cclk_freq_limit (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             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.

This function will read Frequency for the given socket

Parameters

`in`	*i2c_bus*	is the Bus connected to the socket
`in`	*i2c_addr*	is the 7-bit socket address
`out`	*buffer*	is to read freequency[MHz]

Return values

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

5.9.2.9 read_socket_c0_residency()

```
oob_status_t read_socket_c0_residency (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             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.

This function will read Socket C0 residency[%] for the given socket.

Parameters

in	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
out	*buffer*	is to read Socket C0 residency[%].

Return values

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

5.9.2.10 read_ddr_bandwidth()

```
oob_status_t read_ddr_bandwidth (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            uint32_t * max_bw,
            uint32_t * utilized_bw,
            uint32_t * utilized_pct )
```

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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
out	*max_bw*	is the maxium DDR Bandwidth in GB/s
out	*utilized_bw*	is the utilized DDR Bandwidth in GB/s
out	*utilized_pct*	is the utilized DDR Bandwidth in %.

Return values

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

5.10 SB_RMI Read Processor Register Access

Functions

- oob_status_t esmi_oob_read_msr (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t thread, uint32_t msraddr, uint64_t ∗buffer)

  Read the MCA MSR register for a given thread.

5.10.1 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.

5.10.2 Function Documentation

5.10.2.1 esmi_oob_read_msr()

```
oob_status_t esmi_oob_read_msr (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
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.

5.11 SB-RMI CPUID Register Access

Functions

- oob_status_t esmi_oob_cpuid (uint32_t i2c_bus, uint32_t i2c_addr, 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 i2c_bus, uint32_t i2c_addr, 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 i2c_bus, uint32_t i2c_addr, 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 i2c_bus, uint32_t i2c_addr, 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 i2c_bus, uint32_t i2c_addr, uint32_t thread, uint32_t fn_eax, uint32_t fn_ecx, uint32_t ∗edx)

  Read edx register on CPUID functionality.

5.11.1 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.

5.11.2 Function Documentation

5.11.2.1 esmi_oob_cpuid()

```
oob_status_t esmi_oob_cpuid (
              uint32_t i2c_bus,
              uint32_t i2c_addr,
              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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
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.

5.11.2.2 esmi_oob_cpuid_eax()

```
oob_status_t esmi_oob_cpuid_eax (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             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`	*i2c_bus*	is the Bus connected to the socket
`in`	*i2c_addr*	is the 7-bit socket address
`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.

5.11.2.3 esmi_oob_cpuid_ebx()

```
oob_status_t esmi_oob_cpuid_ebx (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
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.

5.11.2.4 esmi_oob_cpuid_ecx()

```
oob_status_t esmi_oob_cpuid_ecx (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
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.

5.11.2.5 esmi_oob_cpuid_edx()

```
oob_status_t esmi_oob_cpuid_edx (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
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 SB-RMI Register Read Byte Protocol

Functions

- oob_status_t read_sbrmi_revision (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗buffer)

  Read Control byte from SB_RMI register command.
- oob_status_t read_sbrmi_status (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗buffer)

  Read one byte of Status value from SB_RMI register command.
- oob_status_t read_sbrmi_readsize (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗buffer)

  Read one byte of Thread Status from SB_RMI register command.
- oob_status_t read_sbrmi_swinterrupt (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗buffer)

  This register indicates the maximum number of threads present.

5.12.1 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.

5.12.2 Function Documentation

5.12.2.1 read_sbrmi_revision()

```
oob_status_t read_sbrmi_revision (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            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	*i2c_bus*	i2c bus number
in	*i2c_addr*	device address on the i2c bus
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.

5.13 SBTSI Register Read Byte Protocol

Functions

- oob_status_t read_sbtsi_cpuinttemp (uint32_t i2c_bus, uint32_t i2c_addr, uint8_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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗buffer)

  This register provides write access to SBTSI::Config.
- oob_status_t read_sbtsi_cputempdecimal (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗buffer)

  The value returns the decimal portion of the CPU temperature.
- oob_status_t read_sbtsi_cputempoffint (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗temp_int)

  SBTSI::CpuTempOffInt and SBTSI::CpuTempOffDec combine to specify the CPU temperature offset.
- oob_status_t read_sbtsi_cputempoffdec (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗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 (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗temp_dec)

  This value specifies the decimal portion of the high temperature threshold.
- oob_status_t read_sbtsi_lotempdecimal (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗temp_dec)

  value specifies the decimal portion of the low temperature threshold.
- oob_status_t read_sbtsi_timeoutconfig (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗man_id)

  Returns the AMD manufacture ID.
- oob_status_t read_sbtsi_revision (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗rivision)

  Specifies the SBI temperature sensor interface revision.

- oob_status_t sbtsi_get_cputemp (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗timeout_en)

  To verify if timeout support enabled or disabled.
- oob_status_t sbtsi_set_timeout_config (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t mode)

  To enable/disable timeout support.
- oob_status_t sbtsi_set_hitemp_threshold (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, float ∗temp_offset)

  SBTSI::CpuTempOffInt and SBTSI::CpuTempOffDec combine to specify the CPU temperature offset.
- oob_status_t write_sbtsi_cputempoffset (uint32_t i2c_bus, uint32_t i2c_addr, float temp_offset)

  SBTSI::CpuTempOffInt and SBTSI::CpuTempOffDec combine to set the CPU temperature offset.
- oob_status_t sbtsi_set_alert_threshold (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t mode)

  Alert comparator mode enable.

5.13.1 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.

5.13.2 Function Documentation

5.13.2.1 read_sbtsi_cpuinttemp()

```
oob_status_t read_sbtsi_cpuinttemp (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            uint8_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,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`	*i2c_bus*	is the Bus connected to the socket
`in`	*i2c_addr*	is the 7-bit socket address
`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.

5.13.2.2 read_sbtsi_status()

```
oob_status_t read_sbtsi_status (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            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`	*i2c_bus*	is the Bus connected to the socket
`in`	*i2c_addr*	is the 7-bit socket address
`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.

5.13.2.3 read_sbtsi_config()

```
oob_status_t read_sbtsi_config (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             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`	*i2c_bus*	is the Bus connected to the socket
`in`	*i2c_addr*	is the 7-bit socket address
`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.

5.13.2.4 read_sbtsi_updaterate()

```
oob_status_t read_sbtsi_updaterate (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             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`	*i2c_bus*	is the Bus connected to the socket
`in`	*i2c_addr*	is the 7-bit socket address
`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.

5.13.2.5 write_sbtsi_updaterate()

```
oob_status_t write_sbtsi_updaterate (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in	*uprate*	value to write in raw format

Return values

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

5.13.2.6 read_sbtsi_hitempint()

```
oob_status_t read_sbtsi_hitempint (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in,out	*buffer*	a pointer to hold the integer part of high cpu temp

Return values

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

5.13.2.7 read_sbtsi_lotempint()

```
oob_status_t read_sbtsi_lotempint (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in,out	*buffer*	a pointer to hold the integer part of low cpu temp

Return values

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

5.13.2.8 read_sbtsi_configwrite()

```
oob_status_t read_sbtsi_configwrite (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            uint8_t * buffer )
```

This register provides write access to SBTSI::Config.

Parameters

in	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in,out	*buffer*	a pointer to hold the configuraion

Return values

OOB_SUCCESS	is returned upon successful call.

Return values

None-zero	is returned upon failure.

5.13.2.9 read_sbtsi_cputempdecimal()

```
oob_status_t read_sbtsi_cputempdecimal (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             uint8_t * buffer )
```

The value returns the decimal portion of the CPU temperature.

Parameters

in	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in,out	*buffer*	a pointer to hold the cpu temperature decimal

Return values

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

5.13.2.10 read_sbtsi_cputempoffint()

```
oob_status_t read_sbtsi_cputempoffint (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             uint8_t * temp_int )
```

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

Parameters

in	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in,out	*temp_int*	a pointer to hold the cpu offset interger

Return values

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

5.13.2.11 read_sbtsi_cputempoffdec()

```
oob_status_t read_sbtsi_cputempoffdec (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             uint8_t * temp_dec )
```

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

Parameters

in	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in,out	*temp_dec*	a pointer to hold the cpu offset decimal

Return values

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

5.13.2.12 read_sbtsi_hitempdecimal()

```
oob_status_t read_sbtsi_hitempdecimal (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             uint8_t * temp_dec )
```

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

Parameters

in	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in,out	*temp_dec*	a pointer to hold the decimal part of cpu high temp

Return values

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

5.13.2.13 read_sbtsi_lotempdecimal()

```
oob_status_t read_sbtsi_lotempdecimal (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            uint8_t * temp_dec )
```

value specifies the decimal portion of the low temperature threshold.

Parameters

in	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in,out	*temp_dec*	a pointer to hold the decimal part of cpu low temperature

Return values

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

5.13.2.14 read_sbtsi_timeoutconfig()

```
oob_status_t read_sbtsi_timeoutconfig (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in,out	*timeout*	a pointer to hold the cpu timeout configuration

Return values

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

5.13.2.15 read_sbtsi_alertthreshold()

```
oob_status_t read_sbtsi_alertthreshold (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            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`	*i2c_bus*	is the Bus connected to the socket
`in`	*i2c_addr*	is the 7-bit socket address
`in,out`	*samples*	a pointer to hold the cpu temperature alert threshold

Return values

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

5.13.2.16 read_sbtsi_alertconfig()

```
oob_status_t read_sbtsi_alertconfig (
              uint32_t i2c_bus,
              uint32_t i2c_addr,
              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`	*i2c_bus*	is the Bus connected to the socket
`in`	*i2c_addr*	is the 7-bit socket address
`in,out`	*mode*	a pointer to hold the cpu temperature alert configuration

Return values

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

5.13.2.17 read_sbtsi_manufid()

```
oob_status_t read_sbtsi_manufid (
              uint32_t i2c_bus,
              uint32_t i2c_addr,
              uint8_t * man_id )
```

Returns the AMD manufacture ID.

Parameters

in	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in,out	*man_id*	a pointer to hold the manufacture id

Return values

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

5.13.2.18 read_sbtsi_revision()

```
oob_status_t read_sbtsi_revision (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             uint8_t * rivision )
```

Specifies the SBI temperature sensor interface revision.

Parameters

in	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in,out	*rivision*	a pointer to hold the cpu temperature revision

Return values

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

5.13.2.19 sbtsi_get_cputemp()

```
oob_status_t sbtsi_get_cputemp (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in,out	*cpu_temp*	a pointer to get temperature of the CPU

Return values

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

5.13.2.20 sbtsi_get_temp_status()

```
oob_status_t sbtsi_get_temp_status (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             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`	*i2c_bus*	is the Bus connected to the socket
`in`	*i2c_addr*	is the 7-bit socket address
`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.

5.13.2.21 sbtsi_get_config()

```
oob_status_t sbtsi_get_config (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             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`	*i2c_bus*	is the Bus connected to the socket

Parameters

in	*i2c_addr*	is the 7-bit socket address i∗
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.

5.13.2.22 sbtsi_set_configwr()

```
oob_status_t sbtsi_set_configwr (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
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.
None-zero	is returned upon failure.

5.13.2.23 sbtsi_get_timeout()

```
oob_status_t sbtsi_get_timeout (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            uint8_t * timeout_en )
```

To verify if timeout support enabled or disabled.

Parameters

in	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
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.
None-zero	is returned upon failure.

5.13.2.24 sbtsi_set_timeout_config()

```
oob_status_t sbtsi_set_timeout_config (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            uint8_t mode )
```

To enable/disable timeout support.

Parameters

in	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
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.
None-zero	is returned upon failure.

5.13.2.25 sbtsi_set_hitemp_threshold()

```
oob_status_t sbtsi_set_hitemp_threshold (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in	*hitemp_thr*	Specifies the high temperature threshold

Return values

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

5.13.2.26 sbtsi_set_lotemp_threshold()

```
oob_status_t sbtsi_set_lotemp_threshold (
              uint32_t i2c_bus,
              uint32_t i2c_addr,
              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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in	*lotemp_thr*	Specifies the low temperature threshold

Return values

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

5.13.2.27 sbtsi_get_hitemp_threshold()

```
oob_status_t sbtsi_get_hitemp_threshold (
              uint32_t i2c_bus,
              uint32_t i2c_addr,
              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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in	*hitemp_thr*	Specifies the high temperature threshold

Return values

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

5.13.2.28 sbtsi_get_lotemp_threshold()

```
oob_status_t sbtsi_get_lotemp_threshold (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in,out	*lotemp_thr*	Get the low temperature threshold

Return values

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

5.13.2.29 read_sbtsi_cputempoffset()

```
oob_status_t read_sbtsi_cputempoffset (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             float * temp_offset )
```

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

Parameters

in	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
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.

5.13.2.30 write_sbtsi_cputempoffset()

```
oob_status_t write_sbtsi_cputempoffset (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             float temp_offset )
```

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

Parameters

in	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
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.

5.13.2.31 sbtsi_set_alert_threshold()

```
oob_status_t sbtsi_set_alert_threshold (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
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.

Return values

None-zero	is returned upon failure.

5.13.2.32 sbtsi_set_alert_config()

```
oob_status_t sbtsi_set_alert_config (
            uint32_t i2c_bus,
            uint32_t i2c_addr,
            uint8_t mode )
```

Alert comparator mode enable.

Parameters

in	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
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.
None-zero	is returned upon failure.

Chapter 6

Data Structure Documentation

6.1 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.

6.1.1 Detailed Description

Read Proccessor Info.

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

- esmi_cpuid_msr.h

6.2 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 127 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.

6.2.1 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.

6.2.2 Field Documentation

6.2.2.1 cmd

```
uint8_t sbrmi_indata::cmd
```

Read CPUID/Read Register Command Format is 0x73.

command protocol

6.2.2.2 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.

6.2.2.3 regcmd

```
uint8_t sbrmi_indata::regcmd
```

read CPUID command is 0x91

Read Processor Register command is 0x86

6.2.2.4 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

6.3 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
      [4,4] bytes of [eax, ebx] or [ecx, edx]
    uint8_t bytes [8]
      [4,4] bytes of [eax, ebx] or [ecx, edx]
  };

6.3.1 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.

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 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.

7.1.1 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 bus.

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
- struct processor_info

  Read Proccessor Info.

Functions

- struct sbrmi_indata __attribute__ ((packed)) rmi_indata

  SB-RMI Read Proccessor Register command protocol and read CPUID command protocol.
- oob_status_t esmi_get_vendor_id (uint32_t i2c_bus, uint32_t i2c_addr, char ∗vendor_id)

  Get the number of logical cores per socket.
- oob_status_t esmi_get_processor_info (uint32_t i2c_bus, uint32_t i2c_addr, struct processor_info ∗proc_info)

  Get the number of logical cores per socket.
- oob_status_t esmi_get_logical_cores_per_socket (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t ∗logical_↩ cores_per_socket)

  Get the number of logical cores per socket.

- oob_status_t esmi_get_threads_per_socket (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t ∗threads_per_↩ socket)

  Get the number of threads per socket.
- oob_status_t esmi_get_threads_per_core (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t ∗threads_per_core)

  Get number of threads per core.
- oob_status_t esmi_oob_read_msr (uint32_t i2c_bus, uint32_t i2c_addr, 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 i2c_bus, uint32_t i2c_addr, 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 i2c_bus, uint32_t i2c_addr, 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 i2c_bus, uint32_t i2c_addr, 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 i2c_bus, uint32_t i2c_addr, 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 i2c_bus, uint32_t i2c_addr, 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 127 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
    [4,4] bytes of [eax, ebx] or [ecx, edx]
  uint8_t bytes [8]
    [4,4] bytes of [eax, ebx] or [ecx, edx]
};

- struct processor_info __attribute__

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_indata __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.

7.2.2.2 esmi_get_vendor_id()

```
oob_status_t esmi_get_vendor_id (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             char * vendor_id )
```

Get the number of logical cores per socket.

Get the processor vendor

Parameters

in	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
out	*vendor←_id*	to get the processor vendor, 12 byte RO value

Return values

uint32↩ _t	is returned upon successful call.

7.2.2.3 esmi_get_processor_info()

```
oob_status_t esmi_get_processor_info (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
out	*proc_info*	to get family, model & stepping identifier

Return values

uint32↩ _t	is returned upon successful call.

7.2.2.4 esmi_get_logical_cores_per_socket()

```
oob_status_t esmi_get_logical_cores_per_socket (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             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	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in,out	*logical_cores_per_socket*	is returned

Return values

logical_cores_per_socket	is returned upon successful call.

7.2.2.5 esmi_get_threads_per_socket()

```
oob_status_t esmi_get_threads_per_socket (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             uint32_t * threads_per_socket )
```

Get the number of threads per socket.

Get the total number of threads in a socket.

Parameters

in	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in,out	*threads_per_socket*	is returned

Return values

threads_per_socket	is returned upon successful call.

7.2.2.6 esmi_get_threads_per_core()

```
oob_status_t esmi_get_threads_per_core (
             uint32_t i2c_bus,
             uint32_t i2c_addr,
             uint32_t * threads_per_core )
```

Get number of threads per core.

Get the number of threads per core.

Parameters

in	*i2c_bus*	is the Bus connected to the socket
in	*i2c_addr*	is the 7-bit socket address
in,out	*threads_per_core*	is returned

Return values

threads_per_core	is returned upon successful call.

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

[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

7.2.3.6 __attribute__

```
struct sbrmi_outdata __attribute__
```

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

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**, **READ_DDR_BANDWID←TH** }

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

Functions

- oob_status_t read_socket_power (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t ∗buffer)

  Get the power consumption of the socket with provided i2c_bus and i2c_addr.
- oob_status_t read_socket_power_limit (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t ∗buffer)

  Get the current power cap/limit value for a given socket.
- oob_status_t read_max_socket_power_limit (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t limit)

  Set the power cap/limit value for a given socket.
- oob_status_t read_esb_boost_limit (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t value, uint32_t ∗buffer)

  Get the Out-of-band boostlimit value for a given core.
- oob_status_t read_bios_boost_fmax (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t value, uint32_t ∗buffer)

  Get the In-band maximum boostlimit value for a given core.
- oob_status_t write_esb_boost_limit (uint32_t i2c_bus, uint32_t i2c_addr, int cpu_ind, uint32_t limit)

  Set the Out-of-band boostlimit value for a given core.
- oob_status_t write_esb_boost_limit_allcores (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t limit)

  Set the boostlimit value for the whole socket (whole system).
- oob_status_t read_tdp (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t ∗buffer)

  Get the Thermal Design Power limit TDP of the socket with provided socket index.
- oob_status_t read_max_tdp (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t ∗buffer)

  Get the Maximum Thermal Design Power limit TDP of the socket with provided socket index.
- oob_status_t read_min_tdp (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t ∗buffer)

  Get the Minimum Thermal Design Power limit TDP of the socket.
- oob_status_t read_prochot_status (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t ∗buffer)

  Get the Prochot Status of the socket with provided socket index.
- oob_status_t read_prochot_residency (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t ∗buffer)

  Get the Prochot Residency (since the boot time or last read of Prochot Residency) of the socket.
- oob_status_t read_dram_throttle (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t ∗buffer)

Read Dram Throttle will always read the lowest percentage value.

- oob_status_t write_dram_throttle (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t limit)

    Set Dram Throttle value in terms of percentage.
- oob_status_t read_vddio_mem_power (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t ∗buffer)

    Read VDDIOMem Power returns the estimated VDDIOMem power consumed within the socket.
- oob_status_t read_nbio_error_logging_register (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t quadrant, uint32_t offset, uint32_t ∗buffer)

    Read NBIO Error Logging Register.
- oob_status_t read_iod_bist (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t ∗buffer)

    Read IOD Bist status.
- oob_status_t read_ccd_bist_result (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t value, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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.
- oob_status_t read_ddr_bandwidth (uint32_t i2c_bus, uint32_t i2c_addr, uint32_t ∗max_bw, uint32_←
t ∗utilized_bw, uint32_t ∗utilized_pct)

    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.

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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗buffer)

  Read Control byte from SB_RMI register command.
- oob_status_t read_sbrmi_status (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗buffer)

  Read one byte of Status value from SB_RMI register command.
- oob_status_t read_sbrmi_readsize (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗buffer)

  Read one byte of Thread Status from SB_RMI register command.
- oob_status_t read_sbrmi_swinterrupt (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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_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 (uint32_t i2c_bus, uint32_t i2c_addr, uint8_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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗buffer)

    This register provides write access to SBTSI::Config.
- oob_status_t read_sbtsi_cputempdecimal (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗buffer)

    The value returns the decimal portion of the CPU temperature.
- oob_status_t read_sbtsi_cputempoffint (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗temp_int)

    SBTSI::CpuTempOffInt and SBTSI::CpuTempOffDec combine to specify the CPU temperature offset.
- oob_status_t read_sbtsi_cputempoffdec (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗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 (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗temp_dec)

    This value specifies the decimal portion of the high temperature threshold.
- oob_status_t read_sbtsi_lotempdecimal (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗temp_dec)

    value specifies the decimal portion of the low temperature threshold.
- oob_status_t read_sbtsi_timeoutconfig (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗man_id)

    Returns the AMD manufacture ID.
- oob_status_t read_sbtsi_revision (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗rivision)

    Specifies the SBI temperature sensor interface revision.
- oob_status_t sbtsi_get_cputemp (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t ∗timeout_en)

  To verify if timeout support enabled or disabled.
- oob_status_t sbtsi_set_timeout_config (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t mode)

  To enable/disable timeout support.
- oob_status_t sbtsi_set_hitemp_threshold (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, float ∗temp_offset)

  SBTSI::CpuTempOffInt and SBTSI::CpuTempOffDec combine to specify the CPU temperature offset.
- oob_status_t write_sbtsi_cputempoffset (uint32_t i2c_bus, uint32_t i2c_addr, float temp_offset)

  SBTSI::CpuTempOffInt and SBTSI::CpuTempOffDec combine to set the CPU temperature offset.
- oob_status_t sbtsi_set_alert_threshold (uint32_t i2c_bus, uint32_t i2c_addr, 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 (uint32_t i2c_bus, uint32_t i2c_addr, uint8_t mode)

  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

__attribute__
esmi_cpuid_msr.h, 69, 72

Auxiliary functions, 13
errno_to_oob_status, 13
esmi_get_err_msg, 14

cmd
esmi_cpuid_msr.h, 72
sbrmi_indata, 62
Current, Min, Max TDP, 23
read_max_tdp, 23
read_min_tdp, 24
read_tdp, 23

Dram and other features Query, 27
read_ccd_bist_result, 31
read_cclk_freq_limit, 32
read_ccx_bist_result, 31
read_ddr_bandwidth, 33
read_dram_throttle, 27
read_iod_bist, 29
read_nbio_error_logging_register, 29
read_socket_c0_residency, 32
read_vddio_mem_power, 28
write_dram_throttle, 28

ecx
esmi_cpuid_msr.h, 72
sbrmi_indata, 63
errno_to_oob_status
Auxiliary functions, 13
esmi_common.h, 65
oob_status_t, 66
esmi_cpuid_msr.h, 67
__attribute__, 69, 72
cmd, 72
ecx, 72
esmi_get_logical_cores_per_socket, 70
esmi_get_processor_info, 70
esmi_get_threads_per_core, 71
esmi_get_threads_per_socket, 71
esmi_get_vendor_id, 69
rd_ln, 72
regcmd, 72
value, 72
esmi_get_err_msg
Auxiliary functions, 14
esmi_get_logical_cores_per_socket
esmi_cpuid_msr.h, 70
esmi_get_processor_info
esmi_cpuid_msr.h, 70
esmi_get_threads_per_core
esmi_cpuid_msr.h, 71
esmi_get_threads_per_socket
esmi_cpuid_msr.h, 71
esmi_get_vendor_id
esmi_cpuid_msr.h, 69
esmi_mailbox.h, 73
esmi_oob_cpuid
SB-RMI CPUID Register Access, 35
esmi_oob_cpuid_eax
SB-RMI CPUID Register Access, 36
esmi_oob_cpuid_ebx
SB-RMI CPUID Register Access, 36
esmi_oob_cpuid_ecx
SB-RMI CPUID Register Access, 37
esmi_oob_cpuid_edx
SB-RMI CPUID Register Access, 37
esmi_oob_read_msr
SB_RMI Read Processor Register Access, 34
esmi_rmi.h, 74
esmi_tsi.h, 75

oob_status_t
esmi_common.h, 66
Out-of-band Performance (Boost limit) Control, 21
write_esb_boost_limit, 21
write_esb_boost_limit_allcores, 21

Performance (Boost limit) Monitor, 19
read_bios_boost_fmax, 19
read_esb_boost_limit, 19
Power Control, 18
write_socket_power_limit, 18
Power Monitor, 16
read_max_socket_power_limit, 17
read_socket_power, 16
read_socket_power_limit, 16
processor_info, 61
Prochot, 25
read_prochot_residency, 25
read_prochot_status, 25

rd_ln
esmi_cpuid_msr.h, 72
sbrmi_indata, 62
read_bios_boost_fmax
Performance (Boost limit) Monitor, 19
read_ccd_bist_result

Dram and other features Query, 31
read_cclk_freq_limit
Dram and other features Query, 32
read_ccx_bist_result
Dram and other features Query, 31
read_ddr_bandwidth
Dram and other features Query, 33
read_dram_throttle
Dram and other features Query, 27
read_esb_boost_limit
Performance (Boost limit) Monitor, 19
read_iod_bist
Dram and other features Query, 29
read_max_socket_power_limit
Power Monitor, 17
read_max_tdp
Current, Min, Max TDP, 23
read_min_tdp
Current, Min, Max TDP, 24
read_nbio_error_logging_register
Dram and other features Query, 29
read_prochot_residency
Prochot, 25
read_prochot_status
Prochot, 25
read_sbrmi_revision
SB-RMI Register Read Byte Protocol, 39
read_sbtsi_alertconfig
SBTSI Register Read Byte Protocol, 50
read_sbtsi_alertthreshold
SBTSI Register Read Byte Protocol, 49
read_sbtsi_config
SBTSI Register Read Byte Protocol, 44
read_sbtsi_configwrite
SBTSI Register Read Byte Protocol, 46
read_sbtsi_cpuinttemp
SBTSI Register Read Byte Protocol, 42
read_sbtsi_cputempdecimal
SBTSI Register Read Byte Protocol, 47
read_sbtsi_cputempoffdec
SBTSI Register Read Byte Protocol, 48
read_sbtsi_cputempoffint
SBTSI Register Read Byte Protocol, 47
read_sbtsi_cputempoffset
SBTSI Register Read Byte Protocol, 58
read_sbtsi_hitempdecimal
SBTSI Register Read Byte Protocol, 48
read_sbtsi_hitempint
SBTSI Register Read Byte Protocol, 45
read_sbtsi_lotempdecimal
SBTSI Register Read Byte Protocol, 48
read_sbtsi_lotempint
SBTSI Register Read Byte Protocol, 46
read_sbtsi_manufid
SBTSI Register Read Byte Protocol, 50
read_sbtsi_revision
SBTSI Register Read Byte Protocol, 52
read_sbtsi_status
SBTSI Register Read Byte Protocol, 43
read_sbtsi_timeoutconfig
SBTSI Register Read Byte Protocol, 49
read_sbtsi_updaterate
SBTSI Register Read Byte Protocol, 44
read_socket_c0_residency
Dram and other features Query, 32
read_socket_power
Power Monitor, 16
read_socket_power_limit
Power Monitor, 16
read_tdp
Current, Min, Max TDP, 23
read_vddio_mem_power
Dram and other features Query, 28
regcmd
esmi_cpuid_msr.h, 72
sbrmi_indata, 62

SB-RMI CPUID Register Access, 35
esmi_oob_cpuid, 35
esmi_oob_cpuid_eax, 36
esmi_oob_cpuid_ebx, 36
esmi_oob_cpuid_ecx, 37
esmi_oob_cpuid_edx, 37
SB-RMI Mailbox Service, 15
SB-RMI Register Read Byte Protocol, 39
read_sbrmi_revision, 39
SB_RMI Read Processor Register Access, 34
esmi_oob_read_msr, 34
SBTSI Register Read Byte Protocol, 41
read_sbtsi_alertconfig, 50
read_sbtsi_alertthreshold, 49
read_sbtsi_config, 44
read_sbtsi_configwrite, 46
read_sbtsi_cpuinttemp, 42
read_sbtsi_cputempdecimal, 47
read_sbtsi_cputempoffdec, 48
read_sbtsi_cputempoffint, 47
read_sbtsi_cputempoffset, 58
read_sbtsi_hitempdecimal, 48
read_sbtsi_hitempint, 45
read_sbtsi_lotempdecimal, 48
read_sbtsi_lotempint, 46
read_sbtsi_manufid, 50
read_sbtsi_revision, 52
read_sbtsi_status, 43
read_sbtsi_timeoutconfig, 49
read_sbtsi_updaterate, 44
sbtsi_get_config, 53
sbtsi_get_cputemp, 52
sbtsi_get_hitemp_threshold, 56
sbtsi_get_lotemp_threshold, 58
sbtsi_get_temp_status, 53
sbtsi_get_timeout, 54
sbtsi_set_alert_config, 60
sbtsi_set_alert_threshold, 59
sbtsi_set_configwr, 54
sbtsi_set_hitemp_threshold, 55

sbtsi_set_lotemp_threshold, 56
sbtsi_set_timeout_config, 55
write_sbtsi_cputempoffset, 59
write_sbtsi_updaterate, 45
sbrmi_indata, 61
cmd, 62
ecx, 63
rd_ln, 62
regcmd, 62
sbrmi_outdata, 63
sbtsi_get_config
SBTSI Register Read Byte Protocol, 53
sbtsi_get_cputemp
SBTSI Register Read Byte Protocol, 52
sbtsi_get_hitemp_threshold
SBTSI Register Read Byte Protocol, 56
sbtsi_get_lotemp_threshold
SBTSI Register Read Byte Protocol, 58
sbtsi_get_temp_status
SBTSI Register Read Byte Protocol, 53
sbtsi_get_timeout
SBTSI Register Read Byte Protocol, 54
sbtsi_set_alert_config
SBTSI Register Read Byte Protocol, 60
sbtsi_set_alert_threshold
SBTSI Register Read Byte Protocol, 59
sbtsi_set_configwr
SBTSI Register Read Byte Protocol, 54
sbtsi_set_hitemp_threshold
SBTSI Register Read Byte Protocol, 55
sbtsi_set_lotemp_threshold
SBTSI Register Read Byte Protocol, 56
sbtsi_set_timeout_config
SBTSI Register Read Byte Protocol, 55

value
esmi_cpuid_msr.h, 72

write_dram_throttle
Dram and other features Query, 28
write_esb_boost_limit
Out-of-band Performance (Boost limit) Control, 21
write_esb_boost_limit_allcores
Out-of-band Performance (Boost limit) Control, 21
write_sbtsi_cputempoffset
SBTSI Register Read Byte Protocol, 59
write_sbtsi_updaterate
SBTSI Register Read Byte Protocol, 45
write_socket_power_limit
Power Control, 18