

E-SMI Library: EPYC™ Systems Management Interface Library

Generated by Doxygen 1.8.13



# Contents

<b>1</b>	<b>EPYC™ System Management Interface (E-SMI) In-band Library</b>	<b>1</b>
<b>2</b>	<b>Module Index</b>	<b>7</b>
2.1	Modules . . . . .	7
<b>3</b>	<b>Data Structure Index</b>	<b>9</b>
3.1	Data Structures . . . . .	9
<b>4</b>	<b>File Index</b>	<b>11</b>
4.1	File List . . . . .	11
<b>5</b>	<b>Module Documentation</b>	<b>13</b>
5.1	Initialization and Shutdown . . . . .	13
5.1.1	Detailed Description . . . . .	13
5.1.2	Function Documentation . . . . .	13
5.1.2.1	esmi_init() . . . . .	13
5.2	Energy Monitor (RAPL MSR) . . . . .	14
5.2.1	Detailed Description . . . . .	14
5.2.2	Function Documentation . . . . .	14
5.2.2.1	esmi_core_energy_get() . . . . .	14
5.2.2.2	esmi_socket_energy_get() . . . . .	15
5.2.2.3	esmi_all_energies_get() . . . . .	15
5.3	HSMP System Statistics . . . . .	16
5.3.1	Detailed Description . . . . .	16
5.3.2	Function Documentation . . . . .	16

5.3.2.1	<a href="#">esmi_smu_fw_version_get()</a>	16
5.3.2.2	<a href="#">esmi_prochot_status_get()</a>	17
5.3.2.3	<a href="#">esmi_df_pstate_set()</a>	17
5.3.2.4	<a href="#">esmi_fclk_mclk_get()</a>	17
5.3.2.5	<a href="#">esmi_cclk_limit_get()</a>	18
5.3.2.6	<a href="#">esmi_hsmp_proto_ver_get()</a>	18
5.4	<a href="#">Power Monitor</a>	20
5.4.1	<a href="#">Detailed Description</a>	20
5.4.2	<a href="#">Function Documentation</a>	20
5.4.2.1	<a href="#">esmi_socket_power_get()</a>	20
5.4.2.2	<a href="#">esmi_socket_power_cap_get()</a>	20
5.4.2.3	<a href="#">esmi_socket_power_cap_max_get()</a>	21
5.5	<a href="#">Power Control</a>	22
5.5.1	<a href="#">Detailed Description</a>	22
5.5.2	<a href="#">Function Documentation</a>	22
5.5.2.1	<a href="#">esmi_socket_power_cap_set()</a>	22
5.6	<a href="#">Performance (Boost limit) Monitor</a>	23
5.6.1	<a href="#">Detailed Description</a>	23
5.6.2	<a href="#">Function Documentation</a>	23
5.6.2.1	<a href="#">esmi_core_boostlimit_get()</a>	23
5.6.2.2	<a href="#">esmi_socket_c0_residency_get()</a>	23
5.7	<a href="#">Performance (Boost limit) Control</a>	25
5.7.1	<a href="#">Detailed Description</a>	25
5.7.2	<a href="#">Function Documentation</a>	25
5.7.2.1	<a href="#">esmi_core_boostlimit_set()</a>	25
5.7.2.2	<a href="#">esmi_socket_boostlimit_set()</a>	25
5.7.2.3	<a href="#">esmi_package_boostlimit_set()</a>	26
5.8	<a href="#">ddr_bandwidth Monitor</a>	27
5.8.1	<a href="#">Detailed Description</a>	27
5.8.2	<a href="#">Function Documentation</a>	27
5.8.2.1	<a href="#">esmi_ddr_bw_get()</a>	27
5.9	<a href="#">Auxiliary functions</a>	28
5.9.1	<a href="#">Detailed Description</a>	28
5.9.2	<a href="#">Function Documentation</a>	28
5.9.2.1	<a href="#">esmi_cpu_family_get()</a>	28
5.9.2.2	<a href="#">esmi_cpu_model_get()</a>	29
5.9.2.3	<a href="#">esmi_threads_per_core_get()</a>	29
5.9.2.4	<a href="#">esmi_number_of_cpus_get()</a>	29
5.9.2.5	<a href="#">esmi_number_of_sockets_get()</a>	30
5.9.2.6	<a href="#">esmi_first_online_core_on_socket()</a>	30
5.9.2.7	<a href="#">esmi_get_err_msg()</a>	31

<b>6</b>	<b>Data Structure Documentation</b>	<b>33</b>
6.1	ddr_bw_metrics Struct Reference . . . . .	33
6.1.1	Detailed Description . . . . .	33
6.2	smu_fw_version Struct Reference . . . . .	33
6.2.1	Detailed Description . . . . .	34
<b>7</b>	<b>File Documentation</b>	<b>35</b>
7.1	e_smi.h File Reference . . . . .	35
7.1.1	Detailed Description . . . . .	37
7.1.2	Enumeration Type Documentation . . . . .	37
7.1.2.1	esmi_status_t . . . . .	37
	<b>Index</b>	<b>39</b>



## Chapter 1

# EPYC™ System Management Interface (E-SMI) In-band Library

The EPYC™ System Management Interface In-band Library, or E-SMI library, is part of the EPYC™ System Management Inband software stack. It is a C library for Linux that provides a user space interface to monitor and control the CPU's power, energy, performance and other system management features.

### Important note about Versioning and Backward Compatibility

The E-SMI 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 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

### Downloading the source

The source code for E-SMI library is available on [Github](#).

### Directory structure of the source

Once the E-SMI 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 e-smi tool, based on the E-SMI library
- `$ include/` Contains the header files used by the E-SMI library
- `$ src/` Contains library E-SMI source

## Building the library and tool

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

```
$ mkdir -p build
```

```
$ cd build
```

```
$ cmake <location of root of E-SMI library CMakeLists.txt>
```

## Building the library for static linking

Building the library as a Static(.a) along with shared libraries(.so) is achieved by following sequence. The static library is part of RPM and DEB package when compiled with cmake as below and built with 'make package'. The next step can be skipped if static lib support is not required

```
$ cmake -DENABLE_STATIC_LIB=1 <location of root of E-SMI library CMakeLists.txt>
```

```
$ make
```

The built library `libe_smi64.so.X.Y` will appear in the build folder.

```
# Install library file and header; default location is /opt/esmi
```

```
$ sudo make install
```

## Building the Documentation

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

```
$ make doc
```

Upon a successful build, the `ESMI_Manual.pdf` and `ESMI_IB_Release_Notes.pdf` will be copied to the top directory of the source.

## Building the package

The RPM and DEB packages can be created with the following steps (continued from the steps above):

```
$ make package
```

## Kernel dependencies

The E-SMI Library depends on the following device drivers from Linux to manage the system management features.



## Monitoring energy counters

The Energy counters reported by the RAPL MSRs, the AMD Energy driver exposes the per core and per socket counters via the HWMON sys entries. The AMD Energy driver is upstreamed and is available as part of Linux v5.8+. The kernel config symbol `SENSORS_AMD_ENERGY` needs to be selected, can be built and inserted as a module.

## Monitoring and managing power metrics, boostlimits and other system management features

The power metrics, boostlimits and other features are managed by the SMU firmware and exposed via PCI config space. AMD provides Linux kernel module exposing this information to the user-space via sys entries.

- `amd_hsmp` module will be made available at [https://github.com/amd/amd\\_hsmp.git](https://github.com/amd/amd_hsmp.git)
- PCIe interface needs to be enabled in the BIOS. On the reference BIOS, the CBS option may be found in the following path

```
Advanced > AMD CBS > NBIO Common Options > SMU Common Options > HSMP  
Support
```

BIOS Default: "Auto" (Disabled)

If the option is disabled, the related E-SMI APIs will return `-ETIMEDOUT`.

## Supported hardware

AMD Zen3 based CPU Family 19h Models 0h-Fh and 30h-3Fh.

## Additional required software for building

In order to build the E-SMI 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)

In order to build the latest documentation, the following are required:

- DOxygen (1.8.13)
- latex (pdfTeX 3.14159265-2.6-1.40.18)

## Usage Basics

### Device Indices

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

## Hello E-SMI

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

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

Below is a simple "Hello World" type program that display the Average Power of Sockets.

```
#include <stdio.h>
#include <stdint.h>
#include <e_smi/e_smi.h>
#include <e_smi/e_smi_monitor.h>

int main()
{
    esmi_status_t ret;
    unsigned int i;
    uint32_t power;
    uint32_t total_sockets = 0;

    ret = esmi_init();
    if (ret != ESMI_SUCCESS) {
        printf("ESMI Not initialized, drivers not found.\n"
            "Err[%d]: %s\n", ret, esmi_get_err_msg(ret));
        return ret;
    }

    total_sockets = esmi_get_number_of_sockets();
    for (i = 0; i < total_sockets; i++) {
        power = 0;
        ret = esmi_socket_power_get(i, &power);
        if (ret != ESMI_SUCCESS) {
            printf("Failed to get socket[%d] avg_power, "
                "Err[%d]: %s\n", i, ret, esmi_get_err_msg(ret));
        }
        printf("socket_%d_avgpower = %.3f Watts\n",
            i, (double)power/1000);
    }
    esmi_exit();

    return ret;
}
```

## Usage

### Tool Usage

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

Below is a sample usage to dump the functionality, with default core/socket/package as 0.

```
e_smi_library/build> sudo ./e_smi_tool

===== EPYC System Management Interface =====

-----
| CPU Family          | 0x19 (25 ) |
| CPU Model           | 0x0 (0 )  |
| NR_CPUS             | 128        |
| NR_SOCKETS          | 2          |
| THREADS PER CORE    | 1 (SMT OFF)|
-----

-----
| Sensor Name          | Socket 0    | Socket 1    |
|-----|-----|-----|
```

Energy (K Joules)	206.088	212.171	
Power (Watts)	42.224	42.634	
PowerLimit (Watts)	200.000	120.000	
PowerLimitMax (Watts)	225.000	225.000	
C0 Residency (%)	0	0	
-----			
Core[0] Energy (Joules)	6.123	5.520	
Core[0] boostlimit(MHz)	2500	2000	
-----			

Try './e\_smi\_tool --help' for more information.

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

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

```
e_smi_library/build> ./e_smi_tool --help

===== EPYC System Management Interface =====

Usage: ./e_smi_tool [Option]... <INPUT>...
Output Option<s>:
  -h, --help                Show this help message
  -A, --showall             Get all esmi parameter Values

Get Option<s>:
  -e, --showcoreenergy CORE    Get energy for a given CPU (Joules)
  -s, --showsocketenergy      Get energy for all sockets (KJoules)
  -p, --showsocketpower       Get power metrics for all sockets (mWatts)
  -L, --showcorebl CORE       Get Boostlimit for a given CPU (MHz)
  -r, --showsocketc0res SOCKET Get c0_residency for a given socket (%)
  -d, --showddrdbw            Show DDR bandwidth details (GBps)
  --showsmufwver              Show SMU FW Version
  --showhsmpprotover          Show HSMP Protocol Version
  --showprochotstatus         Show HSMP PROCHOT status (in/active)
  --showclocks                Show (CPU, Mem & Fabric) clock frequencies (MHz)

Set Option<s>:
  -C, --setpowerlimit SOCKET POWER Set power limit for a given socket (mWatts)
  -a, --setcorebl CORE BOOSTLIMIT  Set boost limit for a given core (MHz)
  --setsocketbl SOCKET BOOSTLIMIT  Set Boost limit for a given Socket (MHz)
  --setpkgbl BOOSTLIMIT            Set Boost limit for a given package (MHz)
  --setdfpstate SOCKET PSTATE      Set Data Fabric Pstate for a given socket
                                   (-1 for auto, 1, 2 and 3 for DF P-states)

===== End of EPYC SMI Log =====
```

Below is a sample usage to get the individual library functionality API's. We can pass arguments in short or long options ex: './e\_smi\_tool -e 0' or './e\_smi\_tool --showcoreenergy 0'

```
1. e_smi_library/build> ./e_smi_tool -e 0

===== EPYC System Management Interface =====

core[0]/energy :          25.733 Joules

===== End of EPYC SMI Log =====

2. e_smi_library/build> ./e_smi_tool --showcoreenergy 0

===== EPYC System Management Interface =====

core[0]/energy :          25.733 Joules

===== End of EPYC SMI Log =====

3. e_smi_library/build> ./e_smi_tool -e 12 --showsocketpower --setpowerlimit 1 220000 -p
===== EPYC System Management Interface =====

core[12]/energy :          19.334 Joules

-----
| Sensor Name          | Socket 0          | Socket 1          |
-----
| Power (Watts)        | 39.053            | 39.633            |
| PowerLimit (Watts)   | 200.000           | 210.000           |
-----
```

```
| PowerLimitMax (Watts) | 225.000          | 225.000          |
-----
Set socket[1]/power_limit :      220.000 Watts successfully
-----
| Sensor Name          | Socket 0          | Socket 1          |
-----
| Power (Watts)         | 39.767            | 40.340            |
| PowerLimit (Watts)    | 200.000           | 220.000           |
| PowerLimitMax (Watts) | 225.000           | 225.000           |
-----
===== End of EPYC SMI Log =====
```

## Chapter 2

# Module Index

### 2.1 Modules

Here is a list of all modules:

Initialization and Shutdown . . . . .	13
Energy Monitor (RAPL MSR) . . . . .	14
HSMP System Statistics . . . . .	16
Power Monitor . . . . .	20
Power Control . . . . .	22
Performance (Boost limit) Monitor . . . . .	23
Performance (Boost limit) Control . . . . .	25
ddr_bandwidth Monitor . . . . .	27
Auxiliary functions . . . . .	28



## Chapter 3

# Data Structure Index

### 3.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">ddr_bw_metrics</a>	
DDR bandwidth metrics . . . . .	33
<a href="#">smu_fw_version</a>	
Deconstruct raw uint32_t into SMU firmware major and minor version numbers . . . . .	33





## Chapter 4

# File Index

### 4.1 File List

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

<a href="#">e_smi.h</a> . . . . .	35
-----------------------------------	----



# Chapter 5

## Module Documentation

### 5.1 Initialization and Shutdown

#### Functions

- `esmi_status_t esmi_init` (void)  
*Initialize the library, validate the dependencies exists.*
- `void esmi_exit` (void)  
*Clean up allocation during init.*

#### 5.1.1 Detailed Description

This function validates the dependencies exists and initializes the library.

#### 5.1.2 Function Documentation

##### 5.1.2.1 `esmi_init()`

```
esmi_status_t esmi_init (  
    void )
```

Initialize the library, validate the dependencies exists.

Search the available dependency entries and initialize the library accordingly.

#### Return values

<code>ESMI_SUCCESS</code>	is returned upon successful call.
<code>None-zero</code>	is returned upon failure.

## 5.2 Energy Monitor (RAPL MSR)

### Functions

- `esmi_status_t esmi_core_energy_get` (uint32\_t core\_ind, uint64\_t \*penergy)  
*Get the core energy for a given core.*
- `esmi_status_t esmi_socket_energy_get` (uint32\_t socket\_idx, uint64\_t \*penergy)  
*Get the socket energy for a given socket.*
- `esmi_status_t esmi_all_energies_get` (uint64\_t \*penergy)  
*Get energies of all cores in the system.*

### 5.2.1 Detailed Description

Below functions provide interfaces to get the core energy value for a given core and to get the socket energy value for a given socket.

### 5.2.2 Function Documentation

#### 5.2.2.1 `esmi_core_energy_get()`

```
esmi_status_t esmi_core_energy_get (
    uint32_t core_ind,
    uint64_t * penergy )
```

Get the core energy for a given core.

Given a core index `core_ind`, and a `penergy` argument for 64bit energy counter of that particular cpu, this function will read the energy counter of the given core and update the `penergy` in micro Joules.

Note: The energy status registers are accessed at core level. In a system with SMT enabled in BIOS, the sibling threads would report duplicate values. Aggregating the energy counters of the sibling threads is incorrect.

#### Parameters

in	<code>core_ind</code>	is a core index
in, out	<code>penergy</code>	Input buffer to return the core energy.

#### Return values

<code>ESMI_SUCCESS</code>	is returned upon successful call.
<code>None-zero</code>	is returned upon failure.

## 5.2.2.2 esmi\_socket\_energy\_get()

```
esmi_status_t esmi_socket_energy_get (
    uint32_t socket_idx,
    uint64_t * penergy )
```

Get the socket energy for a given socket.

Given a socket index `socket_idx`, and a `penergy` argument for 64bit energy counter of a particular socket.

Updates the `penergy` with socket energy in micro Joules.

## Parameters

in	<i>socket_idx</i>	a socket index
in, out	<i>penergy</i>	Input buffer to return the socket energy.

## Return values

<a href="#"><i>ESMI_SUCCESS</i></a>	is returned upon successful call.
<i>None-zero</i>	is returned upon failure.

## 5.2.2.3 esmi\_all\_energies\_get()

```
esmi_status_t esmi_all_energies_get (
    uint64_t * penergy )
```

Get energies of all cores in the system.

Given an argument for energy profile `penergy`, This function will read all core energies in an array `penergy` in micro Joules.

## Parameters

in, out	<i>penergy</i>	Input buffer to return the energies of all cores. <code>penergy</code> should be allocated by user as below <code>(esmi_number_of_cpus_get()/esmi_threads_per_core_get()) * sizeof (uint64_t)</code>
---------	----------------	--

## Return values

<a href="#"><i>ESMI_SUCCESS</i></a>	is returned upon successful call.
<i>None-zero</i>	is returned upon failure.

## 5.3 HSMP System Statistics

### Functions

- [esmi\\_status\\_t esmi\\_smu\\_fw\\_version\\_get](#) (struct [smu\\_fw\\_version](#) \*smu\_fw)  
*Get the SMU Firmware Version.*
- [esmi\\_status\\_t esmi\\_prochot\\_status\\_get](#) (uint32\_t socket\_idx, uint32\_t \*prochot)  
*Get normalized status of the processor's PROCHOT status. 1 - PROCHOT active, 0 - PROCHOT inactive.*
- [esmi\\_status\\_t esmi\\_df\\_pstate\\_set](#) (uint32\_t socket\_idx, int32\_t pstate)  
*Set data fabric P-state and disable automatic P-state selection (analogous to the UEFI setup option APBDIS=1).*
- [esmi\\_status\\_t esmi\\_fclk\\_mclk\\_get](#) (uint32\_t socket\_idx, uint32\_t \*fclk, uint32\_t \*mclk)  
*Get the Data Fabric clock and Memory clock in MHz, for a given socket index.*
- [esmi\\_status\\_t esmi\\_cclk\\_limit\\_get](#) (uint32\_t socket\_idx, uint32\_t \*cclk)  
*Get the core clock (MHz) allowed by the most restrictive infrastructure limit at the time of the message.*
- [esmi\\_status\\_t esmi\\_hsmp\\_proto\\_ver\\_get](#) (uint32\_t \*proto\_ver)  
*Get the HSMP interface (protocol) version.*

### 5.3.1 Detailed Description

Below functions to get HSMP System Statistics.

### 5.3.2 Function Documentation

#### 5.3.2.1 esmi\_smu\_fw\_version\_get()

```
esmi_status_t esmi_smu_fw_version_get (
    struct smu_fw_version * smu_fw )
```

Get the SMU Firmware Version.

This function will return the SMU FW version at `smu_fw`

#### Parameters

<code>in, out</code>	<code>smu_fw</code>	Input buffer to return the smu firmware version.
----------------------	---------------------	--

#### Return values

<a href="#">ESMI_SUCCESS</a>	is returned upon successful call.
<i>None-zero</i>	is returned upon failure.

## 5.3.2.2 esmi\_prochot\_status\_get()

```
esmi_status_t esmi_prochot_status_get (
    uint32_t socket_idx,
    uint32_t * prochot )
```

Get normalized status of the processor's PROCHOT status. 1 - PROCHOT active, 0 - PROCHOT inactive.

Given a socket index `socket_idx` and this function will get PROCHOT at `prochot`.

## Parameters

in	<i>socket_idx</i>	a socket index
in, out	<i>prochot</i>	Input buffer to return the PROCHOT status.

## Return values

<i>ESMI_SUCCESS</i>	is returned upon successful call.
<i>None-zero</i>	is returned upon failure.

## 5.3.2.3 esmi\_df\_pstate\_set()

```
esmi_status_t esmi_df_pstate_set (
    uint32_t socket_idx,
    int32_t pstate )
```

Set data fabric P-state and disable automatic P-state selection (analogous to the UEFI setup option APBDIS=1).

Acceptable values for the P-state are 0(highest) - 3 (lowest). DF P-state passing -1 will enable automatic P-state selection based on data fabric utilization (analogous to APBDIS=0).

This function will set the desired P-state at `pstate`.

## Parameters

in	<i>socket_idx</i>	a socket index
in	<i>pstate</i>	a int32_t that indicates the desired P-state to set.

## Return values

<i>ESMI_SUCCESS</i>	is returned upon successful call.
<i>None-zero</i>	is returned upon failure.

## 5.3.2.4 esmi\_fclk\_mclk\_get()

```
esmi_status_t esmi_fclk_mclk_get (
```

```
uint32_t socket_idx,
uint32_t * fclk,
uint32_t * mclk )
```

Get the Data Fabric clock and Memory clock in MHz, for a given socket index.

Given a socket index `socket_idx` and a pointer to a `uint32_t fclk` and `mclk`, this function will get the data fabric clock and memory clock.

#### Parameters

in	<i>socket_idx</i>	a socket index
in, out	<i>fclk</i>	Input buffer to return the data fabric clock.
in, out	<i>mclk</i>	Input buffer to return the memory clock.

#### Return values

<i>ESMI_SUCCESS</i>	is returned upon successful call.
<i>None-zero</i>	is returned upon failure.

#### 5.3.2.5 esmi\_cclk\_limit\_get()

```
esmi_status_t esmi_cclk_limit_get (
    uint32_t socket_idx,
    uint32_t * cclk )
```

Get the core clock (MHz) allowed by the most restrictive infrastructure limit at the time of the message.

Given a socket index `socket_idx` and a pointer to a `uint32_t cclk`, this function will get the core clock throttle limit.

#### Parameters

in	<i>socket_idx</i>	a socket index
in, out	<i>cclk</i>	Input buffer to return the core clock throttle limit.

#### Return values

<i>ESMI_SUCCESS</i>	is returned upon successful call.
<i>None-zero</i>	is returned upon failure.

#### 5.3.2.6 esmi\_hsmp\_proto\_ver\_get()

```
esmi_status_t esmi_hsmp_proto_ver_get (
    uint32_t * proto_ver )
```



Get the HSMP interface (protocol) version.

This function will get the HSMP interface version at `proto_ver`

#### Parameters

<code>in, out</code>	<code>proto_ver</code>	Input buffer to return the hsmg protocol version.
----------------------	------------------------	---

#### Return values

<code>ESMI_SUCCESS</code>	is returned upon successful call.
<code>None-zero</code>	is returned upon failure.

## 5.4 Power Monitor

### Functions

- [esmi\\_status\\_t esmi\\_socket\\_power\\_get](#) (uint32\_t socket\_idx, uint32\_t \*ppower)  
*Get the instantaneous power consumption of the provided socket.*
- [esmi\\_status\\_t esmi\\_socket\\_power\\_cap\\_get](#) (uint32\_t socket\_idx, uint32\_t \*pcap)  
*Get the current power cap value for a given socket.*
- [esmi\\_status\\_t esmi\\_socket\\_power\\_cap\\_max\\_get](#) (uint32\_t socket\_idx, uint32\_t \*pmax)  
*Get the maximum power cap value for a given socket.*

### 5.4.1 Detailed Description

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

### 5.4.2 Function Documentation

#### 5.4.2.1 esmi\_socket\_power\_get()

```
esmi_status_t esmi_socket_power_get (
    uint32_t socket_idx,
    uint32_t * ppower )
```

Get the instantaneous power consumption of the provided socket.

Given a socket index `socket_idx` and a pointer to a `uint32_t` `ppower`, this function will get the current power consumption (in milliwatts) to the `uint32_t` pointed to by `ppower`.

#### Parameters

in	<i>socket_idx</i>	a socket index
in, out	<i>ppower</i>	Input buffer to return power consumption in the socket.

#### Return values

<a href="#">ESMI_SUCCESS</a>	is returned upon successful call.
<i>None-zero</i>	is returned upon failure.

#### 5.4.2.2 esmi\_socket\_power\_cap\_get()

```
esmi_status_t esmi_socket_power_cap_get (
    uint32_t socket_idx,
    uint32_t * pcap )
```

Get the current power cap value for a given socket.

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

#### Parameters

in	<i>socket_idx</i>	a socket index
in, out	<i>pcap</i>	Input buffer to return power limit on the socket, in milliwatts.

#### Return values

<i>ESMI_SUCCESS</i>	is returned upon successful call.
<i>None-zero</i>	is returned upon failure.

#### 5.4.2.3 esmi\_socket\_power\_cap\_max\_get()

```
esmi_status_t esmi_socket_power_cap_max_get (
    uint32_t socket_idx,
    uint32_t * pmax )
```

Get the maximum power cap value for a given socket.

This function will return the maximum possible valid power cap `pmax` from a `socket_idx`.

#### Parameters

in	<i>socket_idx</i>	a socket index
in, out	<i>pmax</i>	Input buffer to return maximum power limit on socket, in milliwatts.

#### Return values

<i>ESMI_SUCCESS</i>	is returned upon successful call.
<i>None-zero</i>	is returned upon failure.

## 5.5 Power Control

### Functions

- [esmi\\_status\\_t esmi\\_socket\\_power\\_cap\\_set](#) (uint32\_t socket\_idx, uint32\_t pcap)  
*Set the power cap value for a given socket.*

#### 5.5.1 Detailed Description

This function provides a way to control Power Limit.

#### 5.5.2 Function Documentation

##### 5.5.2.1 esmi\_socket\_power\_cap\_set()

```
esmi_status_t esmi_socket_power_cap_set (
    uint32_t socket_idx,
    uint32_t pcap )
```

Set the power cap value for a given socket.

This function will set the power cap to the provided value `pcap`. This cannot be more than the value returned by [esmi\\_socket\\_power\\_cap\\_max\\_get\(\)](#).

Note: The power limit specified will be clipped to the maximum cTDP range for the processor. There is a limit on the minimum power that the processor can operate at, no further power socket reduction occurs if the limit is set below that minimum.

##### Parameters

in	<i>socket_idx</i>	a socket index
in	<i>pcap</i>	a uint32_t that indicates the desired power cap, in milliwatts

##### Return values

<a href="#">ESMI_SUCCESS</a>	is returned upon successful call.
<i>None-zero</i>	is returned upon failure.

## 5.6 Performance (Boost limit) Monitor

### Functions

- [esmi\\_status\\_t esmi\\_core\\_boostlimit\\_get](#) (uint32\_t *cpu\_ind*, uint32\_t \**pboostlimit*)  
*Get the boostlimit value for a given core.*
- [esmi\\_status\\_t esmi\\_socket\\_c0\\_residency\\_get](#) (uint32\_t *socket\_idx*, uint32\_t \**pc0\_residency*)  
*Get the c0\_residency value for a given socket.*

### 5.6.1 Detailed Description

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

### 5.6.2 Function Documentation

#### 5.6.2.1 esmi\_core\_boostlimit\_get()

```
esmi_status_t esmi_core_boostlimit_get (
    uint32_t cpu_ind,
    uint32_t * pboostlimit )
```

Get the boostlimit value for a given core.

This function will return the core's current boost limit *pboostlimit* for a particular *cpu\_ind*

#### Parameters

in	<i>cpu_ind</i>	a cpu index
in, out	<i>pboostlimit</i>	Input buffer to return the boostlimit.

#### Return values

<a href="#">ESMI_SUCCESS</a>	is returned upon successful call.
<i>None-zero</i>	is returned upon failure.

#### 5.6.2.2 esmi\_socket\_c0\_residency\_get()

```
esmi_status_t esmi_socket_c0_residency_get (
    uint32_t socket_idx,
    uint32_t * pc0_residency )
```

Get the c0\_residency value for a given socket.

This function will return the socket's current c0\_residency *pc0\_residency* for a particular *socket\_idx*

## Parameters

in	<i>socket_idx</i>	a socket index provided.
in, out	<i>pc0_residency</i>	Input buffer to return the c0_residency.

## Return values

<i>ESMI_SUCCESS</i>	is returned upon successful call.
<i>None-zero</i>	is returned upon failure.

## 5.7 Performance (Boost limit) Control

### Functions

- `esmi_status_t esmi_core_boostlimit_set (uint32_t cpu_ind, uint32_t boostlimit)`  
*Set the boostlimit value for a given core.*
- `esmi_status_t esmi_socket_boostlimit_set (uint32_t socket_idx, uint32_t boostlimit)`  
*Set the boostlimit value for a given socket.*
- `esmi_status_t esmi_package_boostlimit_set (uint32_t boostlimit)`  
*Set the boostlimit value for the package (whole system).*

### 5.7.1 Detailed Description

Below functions provide ways to control Boost limit values.

### 5.7.2 Function Documentation

#### 5.7.2.1 `esmi_core_boostlimit_set()`

```
esmi_status_t esmi_core_boostlimit_set (
    uint32_t cpu_ind,
    uint32_t boostlimit )
```

Set the boostlimit value for a given core.

This function will set the boostlimit to the provided value `boostlimit` for a given cpu `cpu_ind`.

#### Parameters

in	<code>cpu_ind</code>	a cpu index is a given core to set the boostlimit
in	<code>boostlimit</code>	a <code>uint32_t</code> that indicates the desired boostlimit value of a given core

#### Return values

<code>ESMI_SUCCESS</code>	is returned upon successful call.
<code>None-zero</code>	is returned upon failure.

#### 5.7.2.2 `esmi_socket_boostlimit_set()`

```
esmi_status_t esmi_socket_boostlimit_set (
    uint32_t socket_idx,
    uint32_t boostlimit )
```

Set the boostlimit value for a given socket.

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

#### Parameters

in	<code>socket_idx</code>	a socket index to set boostlimit.
in	<code>boostlimit</code>	a <code>uint32_t</code> that indicates the desired boostlimit value of a particular socket.

#### Return values

<code>ESMI_SUCCESS</code>	is returned upon successful call.
<code>None-zero</code>	is returned upon failure.

#### 5.7.2.3 `esmi_package_boostlimit_set()`

```
esmi_status_t esmi_package_boostlimit_set (
    uint32_t boostlimit )
```

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

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

#### Parameters

in	<code>boostlimit</code>	a <code>uint32_t</code> that indicates the desired boostlimit value of the package.
----	-------------------------	---

#### Return values

<code>ESMI_SUCCESS</code>	is returned upon successful call.
<code>None-zero</code>	is returned upon failure.



## 5.8 ddr\_bandwidth Monitor

### Functions

- `esmi_status_t esmi_ddr_bw_get (struct ddr_bw_metrics *ddr_bw)`

*Get the Theoretical maximum DDR Bandwidth in GB/s, Current utilized DDR Bandwidth in GB/s and Current utilized DDR Bandwidth as a percentage of theoretical maximum in a system.*

### 5.8.1 Detailed Description

This function provides the DDR Bandwidth for a system

### 5.8.2 Function Documentation

#### 5.8.2.1 esmi\_ddr\_bw\_get()

```
esmi_status_t esmi_ddr_bw_get (
    struct ddr_bw_metrics * ddr_bw )
```

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

This function will return the DDR Bandwidth metrics `ddr_bw`

#### Parameters

<code>in, out</code>	<code>ddr_bw</code>	Input buffer to return the DDR bandwidth metrics, contains <code>max_bw</code> , <code>utilized_bw</code> and <code>utilized_pct</code> .
----------------------	---------------------	---

#### Return values

<code>ESMI_SUCCESS</code>	is returned upon successful call.
<code>None-zero</code>	is returned upon failure.

## 5.9 Auxiliary functions

### Functions

- [esmi\\_status\\_t esmi\\_cpu\\_family\\_get](#) (uint32\_t \*family)  
*Get the CPU family.*
- [esmi\\_status\\_t esmi\\_cpu\\_model\\_get](#) (uint32\_t \*model)  
*Get the CPU model.*
- [esmi\\_status\\_t esmi\\_threads\\_per\\_core\\_get](#) (uint32\_t \*threads)  
*Get the number of threads per core in the system.*
- [esmi\\_status\\_t esmi\\_number\\_of\\_cpus\\_get](#) (uint32\_t \*cpus)  
*Get the number of cpus available in the system.*
- [esmi\\_status\\_t esmi\\_number\\_of\\_sockets\\_get](#) (uint32\_t \*sockets)  
*Get the total number of sockets available in the system.*
- [esmi\\_status\\_t esmi\\_first\\_online\\_core\\_on\\_socket](#) (uint32\_t socket\_idx, uint32\_t \*pcore\_ind)  
*Get the first online core on a given socket.*
- char \* [esmi\\_get\\_err\\_msg](#) ([esmi\\_status\\_t](#) esmi\_err)  
*Get the error string message for esmi errors.*

### 5.9.1 Detailed Description

Below functions provide interfaces to get the total number of cores and sockets available and also to get the first online core on a given socket in the system.

### 5.9.2 Function Documentation

#### 5.9.2.1 esmi\_cpu\_family\_get()

```
esmi_status_t esmi_cpu_family_get (
    uint32_t * family )
```

Get the CPU family.

#### Parameters

<i>in, out</i>	<i>family</i>	Input buffer to return the cpu family.
----------------	---------------	--

#### Return values

<a href="#">ESMI_SUCCESS</a>	is returned upon successful call.
<i>None-zero</i>	is returned upon failure.

## 5.9.2.2 esmi\_cpu\_model\_get()

```
esmi_status_t esmi_cpu_model_get (
    uint32_t * model )
```

Get the CPU model.

## Parameters

in, out	<i>model</i>	Input buffer to return the cpu model.
---------	--------------	---------------------------------------

## Return values

<i>ESMI_SUCCESS</i>	is returned upon successful call.
<i>None-zero</i>	is returned upon failure.

## 5.9.2.3 esmi\_threads\_per\_core\_get()

```
esmi_status_t esmi_threads_per_core_get (
    uint32_t * threads )
```

Get the number of threads per core in the system.

## Parameters

in, out	<i>threads</i>	input buffer to return number of SMT threads.
---------	----------------	---

## Return values

<i>ESMI_SUCCESS</i>	is returned upon successful call.
<i>None-zero</i>	is returned upon failure.

## 5.9.2.4 esmi\_number\_of\_cpus\_get()

```
esmi_status_t esmi_number_of_cpus_get (
    uint32_t * cpus )
```

Get the number of cpus available in the system.

## Parameters

in, out	<i>cpus</i>	input buffer to return number of cpus, reported by nproc (including threads in case of SMT enable).
---------	-------------	---

## Return values

<a href="#"><i>ESMI_SUCCESS</i></a>	is returned upon successful call.
<i>None-zero</i>	is returned upon failure.

## 5.9.2.5 esmi\_number\_of\_sockets\_get()

```
esmi_status_t esmi_number_of_sockets_get (
    uint32_t * sockets )
```

Get the total number of sockets available in the system.

## Parameters

in, out	<i>sockets</i>	input buffer to return number of sockets.
---------	----------------	---

## Return values

<a href="#"><i>ESMI_SUCCESS</i></a>	is returned upon successful call.
<i>None-zero</i>	is returned upon failure.

## 5.9.2.6 esmi\_first\_online\_core\_on\_socket()

```
esmi_status_t esmi_first_online_core_on_socket (
    uint32_t socket_idx,
    uint32_t * pcore_ind )
```

Get the first online core on a given socket.

## Parameters

in	<i>socket_idx</i>	a socket index provided.
in, out	<i>pcore_ind</i>	input buffer to return the index of first online core in the socket.

## Return values

<a href="#"><i>ESMI_SUCCESS</i></a>	is returned upon successful call.
<i>None-zero</i>	is returned upon failure.

5.9.2.7 `esmi_get_err_msg()`

```
char* esmi_get_err_msg (
    esmi_status_t esmi_err )
```

Get the error string message for esmi errors.

Get the error message for the esmi error numbers

**Parameters**

in	<i>esmi_err</i>	is a esmi error number
----	-----------------	------------------------

**Return values**

<i>char*</i>	value returned upon successful call.
--------------	--------------------------------------



## Chapter 6

# Data Structure Documentation

### 6.1 ddr\_bw\_metrics Struct Reference

DDR bandwidth metrics.

```
#include <e_smi.h>
```

#### Data Fields

- uint32\_t [max\\_bw](#)  
*DDR Maximum theoretical bandwidth in GB/s.*
- uint32\_t [utilized\\_bw](#)  
*DDR bandwidth utilization in GB/s.*
- uint32\_t [utilized\\_pct](#)  
*DDR bandwidth utilization in % of theoretical max.*

#### 6.1.1 Detailed Description

DDR bandwidth metrics.

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

- [e\\_smi.h](#)

### 6.2 smu\_fw\_version Struct Reference

Deconstruct raw uint32\_t into SMU firmware major and minor version numbers.

```
#include <e_smi.h>
```

## Data Fields

- `uint8_t debug`  
*SMU fw Debug version number.*
- `uint8_t minor`  
*SMU fw Minor version number.*
- `uint8_t major`  
*SMU fw Major version number.*
- `uint8_t unused`  
*reserved fields*

### 6.2.1 Detailed Description

Deconstruct raw `uint32_t` into SMU firmware major and minor version numbers.

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

- `e_smi.h`



## Chapter 7

# File Documentation

### 7.1 e\_smi.h File Reference

#### Data Structures

- struct [smu\\_fw\\_version](#)  
*Deconstruct raw uint32\_t into SMU firmware major and minor version numbers.*
- struct [ddr\\_bw\\_metrics](#)  
*DDR bandwidth metrics.*

#### Macros

- #define [ENERGY\\_DEV\\_NAME](#) "amd\_energy"  
*Supported Energy driver name.*
- #define [HSMP\\_DEV\\_NAME](#) "amd\_hsmp"  
*Supported HSMP driver name.*

#### Enumerations

- enum [esmi\\_status\\_t](#) {  
    [ESMI\\_SUCCESS](#) = 0, [ESMI\\_INITIALIZED](#) = 0, [ESMI\\_NO\\_ENERGY\\_DRV](#), [ESMI\\_NO\\_HSMP\\_DRV](#),  
    [ESMI\\_NO\\_HSMP\\_SUP](#), [ESMI\\_NO\\_DRV](#), [ESMI\\_FILE\\_NOT\\_FOUND](#), [ESMI\\_DEV\\_BUSY](#),  
    [ESMI\\_PERMISSION](#), [ESMI\\_NOT\\_SUPPORTED](#), [ESMI\\_FILE\\_ERROR](#), [ESMI\\_INTERRUPTED](#),  
    [ESMI\\_IO\\_ERROR](#), [ESMI\\_UNEXPECTED\\_SIZE](#), [ESMI\\_UNKNOWN\\_ERROR](#), [ESMI\\_ARG\\_PTR\\_NULL](#),  
    [ESMI\\_NO\\_MEMORY](#), [ESMI\\_NOT\\_INITIALIZED](#), [ESMI\\_INVALID\\_INPUT](#) }  
*Error codes returned by E-SMI functions.*

## Functions

- [esmi\\_status\\_t esmi\\_init](#) (void)  
*Initialize the library, validate the dependencies exists.*
- void [esmi\\_exit](#) (void)  
*Clean up allocation during init.*
- [esmi\\_status\\_t esmi\\_core\\_energy\\_get](#) (uint32\_t core\_ind, uint64\_t \*penergy)  
*Get the core energy for a given core.*
- [esmi\\_status\\_t esmi\\_socket\\_energy\\_get](#) (uint32\_t socket\_idx, uint64\_t \*penergy)  
*Get the socket energy for a given socket.*
- [esmi\\_status\\_t esmi\\_all\\_energies\\_get](#) (uint64\_t \*penergy)  
*Get energies of all cores in the system.*
- [esmi\\_status\\_t esmi\\_smu\\_fw\\_version\\_get](#) (struct [smu\\_fw\\_version](#) \*smu\_fw)  
*Get the SMU Firmware Version.*
- [esmi\\_status\\_t esmi\\_prochot\\_status\\_get](#) (uint32\_t socket\_idx, uint32\_t \*prochot)  
*Get normalized status of the processor's PROCHOT status. 1 - PROCHOT active, 0 - PROCHOT inactive.*
- [esmi\\_status\\_t esmi\\_df\\_pstate\\_set](#) (uint32\_t socket\_idx, int32\_t pstate)  
*Set data fabric P-state and disable automatic P-state selection (analogous to the UEFI setup option APBDIS=1).*
- [esmi\\_status\\_t esmi\\_fclk\\_mclk\\_get](#) (uint32\_t socket\_idx, uint32\_t \*fclk, uint32\_t \*mclk)  
*Get the Data Fabric clock and Memory clock in MHz, for a given socket index.*
- [esmi\\_status\\_t esmi\\_cclk\\_limit\\_get](#) (uint32\_t socket\_idx, uint32\_t \*cclk)  
*Get the core clock (MHz) allowed by the most restrictive infrastructure limit at the time of the message.*
- [esmi\\_status\\_t esmi\\_hsmv\\_proto\\_ver\\_get](#) (uint32\_t \*proto\_ver)  
*Get the HSMP interface (protocol) version.*
- [esmi\\_status\\_t esmi\\_socket\\_power\\_get](#) (uint32\_t socket\_idx, uint32\_t \*ppower)  
*Get the instantaneous power consumption of the provided socket.*
- [esmi\\_status\\_t esmi\\_socket\\_power\\_cap\\_get](#) (uint32\_t socket\_idx, uint32\_t \*pcap)  
*Get the current power cap value for a given socket.*
- [esmi\\_status\\_t esmi\\_socket\\_power\\_cap\\_max\\_get](#) (uint32\_t socket\_idx, uint32\_t \*pmax)  
*Get the maximum power cap value for a given socket.*
- [esmi\\_status\\_t esmi\\_socket\\_power\\_cap\\_set](#) (uint32\_t socket\_idx, uint32\_t pcap)  
*Set the power cap value for a given socket.*
- [esmi\\_status\\_t esmi\\_core\\_boostlimit\\_get](#) (uint32\_t cpu\_ind, uint32\_t \*pboostlimit)  
*Get the boostlimit value for a given core.*
- [esmi\\_status\\_t esmi\\_socket\\_c0\\_residency\\_get](#) (uint32\_t socket\_idx, uint32\_t \*pc0\_residency)  
*Get the c0\_residency value for a given socket.*
- [esmi\\_status\\_t esmi\\_core\\_boostlimit\\_set](#) (uint32\_t cpu\_ind, uint32\_t boostlimit)  
*Set the boostlimit value for a given core.*
- [esmi\\_status\\_t esmi\\_socket\\_boostlimit\\_set](#) (uint32\_t socket\_idx, uint32\_t boostlimit)  
*Set the boostlimit value for a given socket.*
- [esmi\\_status\\_t esmi\\_package\\_boostlimit\\_set](#) (uint32\_t boostlimit)  
*Set the boostlimit value for the package (whole system).*
- [esmi\\_status\\_t esmi\\_ddr\\_bw\\_get](#) (struct [ddr\\_bw\\_metrics](#) \*ddr\_bw)  
*Get the Theoretical maximum DDR Bandwidth in GB/s, Current utilized DDR Bandwidth in GB/s and Current utilized DDR Bandwidth as a percentage of theoretical maximum in a system.*
- [esmi\\_status\\_t esmi\\_cpu\\_family\\_get](#) (uint32\_t \*family)  
*Get the CPU family.*
- [esmi\\_status\\_t esmi\\_cpu\\_model\\_get](#) (uint32\_t \*model)  
*Get the CPU model.*
- [esmi\\_status\\_t esmi\\_threads\\_per\\_core\\_get](#) (uint32\_t \*threads)  
*Get the number of threads per core in the system.*

- `esmi_status_t esmi_number_of_cpus_get` (uint32\_t \*cpus)  
*Get the number of cpus available in the system.*
- `esmi_status_t esmi_number_of_sockets_get` (uint32\_t \*sockets)  
*Get the total number of sockets available in the system.*
- `esmi_status_t esmi_first_online_core_on_socket` (uint32\_t socket\_idx, uint32\_t \*pcore\_ind)  
*Get the first online core on a given socket.*
- `char * esmi_get_err_msg` (esmi\_status\_t esmi\_err)  
*Get the error string message for esmi errors.*

### 7.1.1 Detailed Description

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

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

### 7.1.2 Enumeration Type Documentation

#### 7.1.2.1 esmi\_status\_t

enum `esmi_status_t`

Error codes returned by E-SMI functions.

#### Enumerator

ESMI_SUCCESS	Operation was successful.
ESMI_INITIALIZED	ESMI initialized successfully.
ESMI_NO_ENERGY_DRV	Energy driver not found.
ESMI_NO_HSMP_DRV	HSMP driver not found.
ESMI_NO_HSMP_SUP	HSMP feature not supported.
ESMI_NO_DRV	No Energy and HSMP driver present.
ESMI_FILE_NOT_FOUND	file or directory not found
ESMI_DEV_BUSY	Device or resource busy.
ESMI_PERMISSION	Many functions require root access to run. Permission denied/EACCESS file error.
ESMI_NOT_SUPPORTED	The requested information or action is not available for the given input, on the given system
ESMI_FILE_ERROR	Problem accessing a file. This may because the operation is not supported by the Linux kernel version running on the executing machine
ESMI_INTERRUPTED	execution of function An interrupt occurred during
ESMI_IO_ERROR	An input or output error.
ESMI_UNEXPECTED_SIZE	was read An unexpected amount of data
ESMI_UNKNOWN_ERROR	An unknown error occurred.
ESMI_ARG_PTR_NULL	Parsed argument is invalid.
ESMI_NO_MEMORY	Not enough memory to allocate.
ESMI_NOT_INITIALIZED	ESMI path not initialized.
ESMI_INVALID_INPUT	Input value is invalid.



# Index

## Auxiliary functions, 28

- esmi\_cpu\_family\_get, 28
- esmi\_cpu\_model\_get, 28
- esmi\_first\_online\_core\_on\_socket, 30
- esmi\_get\_err\_msg, 30
- esmi\_number\_of\_cpus\_get, 29
- esmi\_number\_of\_sockets\_get, 30
- esmi\_threads\_per\_core\_get, 29

## ddr\_bandwidth Monitor, 27

- esmi\_ddr\_bw\_get, 27

## ddr\_bw\_metrics, 33

## e\_smi.h, 35

- esmi\_status\_t, 37

## Energy Monitor (RAPL MSR), 14

- esmi\_all\_energies\_get, 15
- esmi\_core\_energy\_get, 14
- esmi\_socket\_energy\_get, 14

## esmi\_all\_energies\_get

- Energy Monitor (RAPL MSR), 15

## esmi\_cclk\_limit\_get

- HSMP System Statistics, 18

## esmi\_core\_boostlimit\_get

- Performance (Boost limit) Monitor, 23

## esmi\_core\_boostlimit\_set

- Performance (Boost limit) Control, 25

## esmi\_core\_energy\_get

- Energy Monitor (RAPL MSR), 14

## esmi\_cpu\_family\_get

- Auxiliary functions, 28

## esmi\_cpu\_model\_get

- Auxiliary functions, 28

## esmi\_ddr\_bw\_get

- ddr\_bandwidth Monitor, 27

## esmi\_df\_pstate\_set

- HSMP System Statistics, 17

## esmi\_fclk\_mclk\_get

- HSMP System Statistics, 17

## esmi\_first\_online\_core\_on\_socket

- Auxiliary functions, 30

## esmi\_get\_err\_msg

- Auxiliary functions, 30

## esmi\_hsmptproto\_ver\_get

- HSMP System Statistics, 18

## esmi\_init

- Initialization and Shutdown, 13

## esmi\_number\_of\_cpus\_get

- Auxiliary functions, 29

## esmi\_number\_of\_sockets\_get

## Auxiliary functions, 30

## esmi\_package\_boostlimit\_set

- Performance (Boost limit) Control, 26

## esmi\_prochot\_status\_get

- HSMP System Statistics, 16

## esmi\_smu\_fw\_version\_get

- HSMP System Statistics, 16

## esmi\_socket\_boostlimit\_set

- Performance (Boost limit) Control, 25

## esmi\_socket\_c0\_residency\_get

- Performance (Boost limit) Monitor, 23

## esmi\_socket\_energy\_get

- Energy Monitor (RAPL MSR), 14

## esmi\_socket\_power\_cap\_get

- Power Monitor, 20

## esmi\_socket\_power\_cap\_max\_get

- Power Monitor, 21

## esmi\_socket\_power\_cap\_set

- Power Control, 22

## esmi\_socket\_power\_get

- Power Monitor, 20

## esmi\_status\_t

- e\_smi.h, 37

## esmi\_threads\_per\_core\_get

- Auxiliary functions, 29

## HSMP System Statistics, 16

- esmi\_cclk\_limit\_get, 18

- esmi\_df\_pstate\_set, 17

- esmi\_fclk\_mclk\_get, 17

- esmi\_hsmptproto\_ver\_get, 18

- esmi\_prochot\_status\_get, 16

- esmi\_smu\_fw\_version\_get, 16

## Initialization and Shutdown, 13

- esmi\_init, 13

## Performance (Boost limit) Control, 25

- esmi\_core\_boostlimit\_set, 25

- esmi\_package\_boostlimit\_set, 26

- esmi\_socket\_boostlimit\_set, 25

## Performance (Boost limit) Monitor, 23

- esmi\_core\_boostlimit\_get, 23

- esmi\_socket\_c0\_residency\_get, 23

## Power Control, 22

- esmi\_socket\_power\_cap\_set, 22

## Power Monitor, 20

- esmi\_socket\_power\_cap\_get, 20

- esmi\_socket\_power\_cap\_max\_get, 21

- esmi\_socket\_power\_get, 20

smu\_fw\_version, [33](#)