

E-SMI Library: EPYC™ Systems Management Interface Library

Generated by Doxygen 1.8.17



<b>1 EPYC™ System Management Interface (E-SMI) In-band Library</b>	<b>1</b>
1.1 Important note about Versioning and Backward Compatibility	1
1.2 Building E-SMI	1
1.2.1 Downloading the source	1
1.2.2 Directory structure of the source	1
1.2.3 Building the library and tool	2
1.2.4 Building the Documentation	2
1.2.5 Building the package	2
1.3 Kernel version dependency	3
1.4 Kernel driver dependencies	3
1.4.1 amd_hsmp driver	3
1.4.2 amd_hsmp/msr_safe/amd_energy/msr	3
1.5 BIOS dependency	4
1.6 Supported hardware	4
1.7 Additional required software for building	4
1.8 Library Usage Basics	5
1.8.1 Hello E-SMI	5
1.9 Tool Usage	5
<b>2 Module Index</b>	<b>11</b>
2.1 Modules	11
<b>3 Data Structure Index</b>	<b>13</b>
3.1 Data Structures	13
<b>4 File Index</b>	<b>15</b>
4.1 File List	15
<b>5 Module Documentation</b>	<b>17</b>
5.1 Initialization and Shutdown	17
5.1.1 Detailed Description	17
5.1.2 Function Documentation	17
5.1.2.1 esmi_init()	17
5.2 Energy Monitor (RAPL MSR)	18
5.2.1 Detailed Description	18
5.2.2 Function Documentation	18
5.2.2.1 esmi_core_energy_get()	18
5.2.2.2 esmi_socket_energy_get()	19
5.2.2.3 esmi_all_energies_get()	19
5.2.2.4 esmi_rapl_units_hsmp_mailbox_get()	20
5.2.2.5 esmi_rapl_package_counter_hsmp_mailbox_get()	20
5.2.2.6 esmi_rapl_core_counter_hsmp_mailbox_get()	21
5.2.2.7 esmi_core_energy_hsmp_mailbox_get()	21
5.2.2.8 esmi_package_energy_hsmp_mailbox_get()	22

5.3 HSMP System Statistics . . . . .	23
5.3.1 Detailed Description . . . . .	23
5.3.2 Function Documentation . . . . .	23
5.3.2.1 esmi_hsmp_driver_version_get() . . . . .	23
5.3.2.2 esmi_smu_fw_version_get() . . . . .	24
5.3.2.3 esmi_prochot_status_get() . . . . .	24
5.3.2.4 esmi_fclk_mclk_get() . . . . .	25
5.3.2.5 esmi_cclk_limit_get() . . . . .	25
5.3.2.6 esmi_hsmp_proto_ver_get() . . . . .	26
5.3.2.7 esmi_socket_current_active_freq_limit_get() . . . . .	26
5.3.2.8 esmi_socket_freq_range_get() . . . . .	26
5.3.2.9 esmi_current_freq_limit_core_get() . . . . .	27
5.3.2.10 esmi_cpurail_isofreq_policy_get() . . . . .	27
5.3.2.11 esmi_dfc_ctrl_setting_get() . . . . .	28
5.4 Power Monitor . . . . .	29
5.4.1 Detailed Description . . . . .	29
5.4.2 Function Documentation . . . . .	29
5.4.2.1 esmi_socket_power_get() . . . . .	29
5.4.2.2 esmi_socket_power_cap_get() . . . . .	30
5.4.2.3 esmi_socket_power_cap_max_get() . . . . .	30
5.4.2.4 esmi_pwr_svi_telemetry_all_rails_get() . . . . .	31
5.4.2.5 esmi_pwr_efficiency_mode_get() . . . . .	31
5.5 Power Control . . . . .	32
5.5.1 Detailed Description . . . . .	32
5.5.2 Function Documentation . . . . .	32
5.5.2.1 esmi_socket_power_cap_set() . . . . .	32
5.5.2.2 esmi_pwr_efficiency_mode_set() . . . . .	33
5.5.2.3 esmi_cpurail_isofreq_policy_set() . . . . .	34
5.5.2.4 esmi_dfc_enable_set() . . . . .	34
5.6 Performance (Boost limit) Monitor . . . . .	36
5.6.1 Detailed Description . . . . .	36
5.6.2 Function Documentation . . . . .	36
5.6.2.1 esmi_core_boostlimit_get() . . . . .	36
5.6.2.2 esmi_socket_c0_residency_get() . . . . .	36
5.7 Performance (Boost limit) Control . . . . .	38
5.7.1 Detailed Description . . . . .	38
5.7.2 Function Documentation . . . . .	38
5.7.2.1 esmi_core_boostlimit_set() . . . . .	38
5.7.2.2 esmi_socket_boostlimit_set() . . . . .	39
5.8 ddr_bandwidth Monitor . . . . .	40
5.8.1 Detailed Description . . . . .	40
5.8.2 Function Documentation . . . . .	40

5.8.2.1 esmi_ddr_bw_get()	40
5.9 Temperature Query	41
5.9.1 Detailed Description	41
5.9.2 Function Documentation	41
5.9.2.1 esmi_socket_temperature_get()	41
5.10 Dimm statistics	42
5.10.1 Detailed Description	42
5.10.2 Function Documentation	42
5.10.2.1 esmi_dimm_temp_range_and_refresh_rate_get()	42
5.10.2.2 esmi_dimm_power_consumption_get()	43
5.10.2.3 esmi_dimm_thermal_sensor_get()	43
5.11 xGMI bandwidth control	45
5.11.1 Detailed Description	45
5.11.2 Function Documentation	45
5.11.2.1 esmi_xgmi_width_set()	45
5.12 GMI3 width control	46
5.12.1 Detailed Description	46
5.12.2 Function Documentation	46
5.12.2.1 esmi_gmi3_link_width_range_set()	46
5.13 APB and LCLK level control	47
5.13.1 Detailed Description	47
5.13.2 Function Documentation	47
5.13.2.1 esmi_apb_enable()	47
5.13.2.2 esmi_apb_disable()	48
5.13.2.3 esmi_socket_lclk_dpm_level_set()	48
5.13.2.4 esmi_socket_lclk_dpm_level_get()	49
5.13.2.5 esmi_pcie_link_rate_set()	49
5.13.2.6 esmi_df_pstate_range_set()	50
5.13.2.7 esmi_xgmi_pstate_range_set()	50
5.14 Bandwidth Monitor	52
5.14.1 Detailed Description	52
5.14.2 Function Documentation	52
5.14.2.1 esmi_current_io_bandwidth_get()	52
5.14.2.2 esmi_current_xgmi_bw_get()	53
5.15 Metrics Table	54
5.15.1 Detailed Description	54
5.15.2 Function Documentation	54
5.15.2.1 esmi_metrics_table_version_get()	54
5.15.2.2 esmi_metrics_table_get()	54
5.15.2.3 esmi_dram_address_metrics_table_get()	55
5.16 Test HSMP mailbox	56
5.16.1 Detailed Description	56

5.16.2 Function Documentation	56
5.16.2.1 esmi_test_hsmp_mailbox()	56
5.17 Auxiliary functions	57
5.17.1 Detailed Description	57
5.17.2 Function Documentation	57
5.17.2.1 esmi_cpu_family_get()	57
5.17.2.2 esmi_cpu_model_get()	58
5.17.2.3 esmi_threads_per_core_get()	58
5.17.2.4 esmi_number_of_cpus_get()	58
5.17.2.5 esmi_number_of_sockets_get()	59
5.17.2.6 esmi_first_online_core_on_socket()	59
5.17.2.7 esmi_get_err_msg()	60
<b>6 Data Structure Documentation</b>	<b>61</b>
6.1 ddr_bw_metrics Struct Reference	61
6.1.1 Detailed Description	61
6.2 dimm_power Struct Reference	61
6.2.1 Detailed Description	62
6.3 dimm_thermal Struct Reference	62
6.3.1 Detailed Description	62
6.4 dpm_level Struct Reference	62
6.4.1 Detailed Description	63
6.5 hsmp_driver_version Struct Reference	63
6.5.1 Detailed Description	63
6.6 link_id_bw_type Struct Reference	63
6.6.1 Detailed Description	64
6.7 smu_fw_version Struct Reference	64
6.7.1 Detailed Description	64
6.8 temp_range_refresh_rate Struct Reference	64
6.8.1 Detailed Description	65
<b>7 File Documentation</b>	<b>67</b>
7.1 e_smi.h File Reference	67
7.1.1 Detailed Description	70
7.1.2 Enumeration Type Documentation	71
7.1.2.1 io_bw_encoding	71
7.1.2.2 esmi_status_t	71
<b>Index</b>	<b>73</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.

### 1.1 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.

### 1.2 Building E-SMI

#### 1.2.1 Downloading the source

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

#### 1.2.2 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
- `$ tools/` 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
- `$ cmake_modules/` Contains helper utilities for determining package and library version
- `$ DEBIAN/` Contains debian pre and post installation scripts
- `$ RPM/` Contains rpm pre and post installation scripts

### 1.2.3 Building the library and tool

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

- `$ cd <location of root of E-smi library>`
- `$ mkdir -p build`
- `$ cd build`
- `$ cmake ../`

#### 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'.

- `$ cmake -DENABLE_STATIC_LIB=1 ../`
- `$ make`

#### Building the library and tool using clang compiler

- `$ cmake -DUSE_CLANG=1 ../`
- `$ make`

The built library `libe_smi64_static.a`, `libe_smi64.so.X.Y` and `esmi_tool` will appear in the build directory

- `$ sudo make install`

By default library file, header and tool are installed at `/opt/e-sms`. To change the default installation path, build with `cmake -DCMAKE_INSTALL_PREFIX=xxxx`. Library will be installed at `xxxx/lib`, tool will be installed at `xxxx/bin`, header will be installed at `xxxx/include` and doc will be installed at `xxxx/e-sms/doc`. Example↵: If `-DCMAKE_INSTALL_PREFIX=/usr/local` then `esmi lib`, `esmi_tool` binary and headers are installed at `/usr/local/lib`, `/usr/local/bin`, `/usr/local/include` respectively.

**Note:** Library is dependent on `amd_hsmp.h` header and without this, compilation will break. Please follow the instruction in "Kernel dependencies" section

### 1.2.4 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.

### 1.2.5 Building the package

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



## 1.3 Kernel version dependency

- Family 0x19 model 00-0fh a0-afh are supported from v5.16-rc7 onwards
- Family 0x19 model 90-9fh are supported from v6.6-rc1 onwards
- Family 0x1A model 00-1fh are supported from v6.5-rc5 onwards

## 1.4 Kernel driver dependencies

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

### 1.4.1 amd\_hsmp driver

This is used to monitor and manage power metrics, boostlimits and other system management features. The power metrics, boostlimits and other features are managed by the SMU(System Management Unit of the processor) firmware and exposed via PCI config space and accessed through "Host System Management Port(HSMP)" at host/cpu side. AMD provides Linux kernel module(`amd_hsmp`) exposing this information to the user-space via `ioctl` interface.

- `amd_hsmp` driver is accepted in upstream kernel and is available at linux tree at `drivers/platform/x86/amd/hsmp.c` from version 5.17.rc1 onwards either it can be compiled as part of kernel as a module or built in driver or as an out of tree module which is available at [https://github.com/amd/amd\\_hsmp.git](https://github.com/amd/amd_hsmp.git)
- E-smi compilation has dependency on `amd_hsmp` header file from `uapi` header of `amd_hsmp` driver. It should be available at
  - `/usr/include/asm/` on RHEL systems
  - `/usr/include/x86_64-linux-gnu/asm/` on Ubuntu systems. If its not present, it can be copied from [amd\\_hsmp](#) github repo or from the kernel source `arch/x86/include/uapi/asm/amd_hsmp.h`
- There is always a dependency between E-smi and `amd_hsmp` driver versions. The new features of E-smi work only if there is a matching HSMP driver.

### 1.4.2 amd\_hsmp/msr\_safe/amd\_energy/msr

One of these drivers is needed to monitor energy counters.

- AMD family 19h, model 00-0fh and 30-3fh
  - These processors support energy monitoring through 32 bit RAPL MSR registers.
  - `amd_energy` driver, an out of tree kernel module, hosted at [amd\\_energy](#) can report per core and per socket counters via the `HWMON` sysfs entries.
  - This driver provides accumulation of energy for avoiding wrap around problem.
  - This is the only supported energy driver for 32bit RAPLS
- AMD family 19h, model 10-1fh, a0-afh and 90-9fh, AMD family 0x1A, model 00-1fh
  - These processors support energy monitoring through 64 bit RAPL MSR registers.
  - Because of 64 bit registers, there is no accumulation of energy needed.
  - For these processors either `msr_safe`, `amd_energy` or kernel's default `msr` driver can be used.

- AMD family 1Ah, model 0x00-0x1f support RAPL reading using HSMP mailbox.
  - For these processors either amd\_hsmp driver or msr\_safe driver or amd\_energy driver or msr driver can be used.
  - The order of checking for the availability of drivers in e-smi is as follows.
    - \* If amd\_hsmp driver is present and supports RAPL reading, this is used for reading energy.
    - \* If amd\_hsmp driver is not present/not supports energy reading, and msr-safe driver is present, this is used for reading energy. Msr-safe driver needs allowlist file to be written to "/dev/cpu/msr\_↵allowlist" for allowing the read of those specific msr registers. Please follow below steps or use the tool option "writemsrallowlist" to write the allowlist file. Create "amd\_allowlist" file with below contents and run the command "sudo su" and "cat amd\_allowlist > /dev/cpu/msr\_allowlist".
      - 0xC0010299 0x0000000000000000 # "ENERGY\_PWR\_UNIT\_MSR"
      - 0xC001029A 0x0000000000000000 # "ENERGY\_CORE\_MSR"
      - 0xC001029B 0x0000000000000000 # "ENERGY\_PKG\_MSR"
      - Note: The first column above indicates MSR register address and 2nd column indicates write mask and the third column is name of the register.
    - \* If msr\_safe driver is not present, amd\_energy driver is present, this is used for reading energy.
    - \* If msr\_safe driver or amd\_energy driver not present, msr driver will be used for reading energy.
    - \* Any one of msr\_safe/amd\_energy/msr driver is sufficient

## 1.5 BIOS dependency

- To get HSMP working. PCIe interface needs to be enabled in the BIOS. On the reference BIOS please follow the sequence below for enabling HSMP.

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

**BIOS Default: "Auto" (Disabled) to BIOS Default: "Enabled"**

If the above HSMP support option is disabled, the related E-SMI APIs will return -ETIMEDOUT. The latest BIOS supports probing of HSMP driver through ACPI device. The ACPI supported `amd_hsmp` driver version is 2.2

## 1.6 Supported hardware

- AMD Zen3 based CPU Family 19h Models 0h-Fh and 30h-3Fh.
- AMD Zen4 based CPU Family 19h Models 10h-1Fh and A0-AFh.
- AMD Zen4 based CPU Family 19h Models 90-9Fh.
- AMD Zen5 based CPU Family 1Ah Models 00-1Fh.

## 1.7 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)
- gcc, g++, make
- build-essential

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

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

## 1.8 Library Usage Basics

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. Number of cores/sockets in a system can be obtained from esmi library APIs.

### 1.8.1 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;
}
```

## 1.9 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 core and socket metrics

```
e_smi_library/b$ sudo ./e_smi_tool
===== E-SMI =====
-----
| CPU Family          | 0x19 (25 ) |
| CPU Model           | 0x10 (16 ) |
| NR_CPUS              | 384         |
| NR_SOCKETS           | 2           |
| THREADS PER CORE    | 2 (SMT ON) |
-----
-----
| Sensor Name          | Socket 0    | Socket 1    |
-----
| Energy (K Joules)     | 14437.971   | 14087.151   |
| Power (Watts)         | 174.290     | 169.630     |
| PowerLimit (Watts)    | 400.000     | 320.000     |
-----
```

PowerLimitMax (Watts)	400.000	320.000
C0 Residency (%)	0	0
DDR Bandwidth		
DDR Max BW (GB/s)	58	58
DDR Utilized BW (GB/s)	0	0
DDR Utilized Percent(%)	0	0
Current Active Freq limit		
Freq limit (MHz)	3500	3500
Freq limit source	Refer below[*0]	Refer below[*1]
Socket frequency range		
Fmax (MHz)	3500	3500
Fmin (MHz)	400	400

---

CPU energies in Joules:

cpu [ 0] :	645.992	181.415	171.678	165.577	161.001	158.397	161.333	151.716
cpu [ 8] :	88.197	79.306	73.860	73.015	72.960	69.293	67.871	78.895
cpu [16] :	70.376	71.231	61.756	63.061	80.656	73.360	69.566	69.969
cpu [24] :	67.054	65.621	64.468	66.346	64.344	64.310	71.548	65.579
cpu [32] :	65.731	62.931	65.526	69.765	69.050	65.782	70.630	65.282
cpu [40] :	69.608	67.261	63.765	69.477	68.677	63.145	62.451	159.949
cpu [48] :	70.810	73.084	64.584	62.966	66.581	65.620	62.381	65.602
cpu [56] :	72.804	70.842	69.651	64.990	63.924	66.468	63.401	296.924
cpu [64] :	64.693	62.723	65.057	62.515	60.091	60.422	62.217	66.552
cpu [72] :	81.746	70.622	68.848	301.949	78.974	68.130	68.141	65.693
cpu [80] :	77.475	72.441	81.296	71.441	71.988	75.237	73.986	69.467
cpu [88] :	73.385	69.277	61.759	61.060	62.834	60.681	62.835	62.703
cpu [96] :	142.718	139.519	134.449	134.097	135.045	140.307	140.553	137.153
cpu [104] :	66.016	66.736	62.224	67.137	64.881	70.592	64.701	64.056
cpu [112] :	70.791	69.107	70.638	69.998	68.199	65.263	70.638	72.557
cpu [120] :	94.391	94.151	71.881	66.493	64.653	66.141	66.132	69.593
cpu [128] :	65.800	64.742	63.130	61.771	65.416	66.205	64.663	71.349
cpu [136] :	72.183	66.754	67.090	63.343	69.450	67.979	68.285	70.478
cpu [144] :	68.281	63.809	62.717	63.348	71.164	72.289	65.516	65.513
cpu [152] :	74.588	69.074	66.711	66.011	67.896	65.933	67.031	65.474
cpu [160] :	66.668	62.996	65.945	63.734	64.060	68.597	76.405	91.436
cpu [168] :	77.658	70.085	67.025	68.951	64.678	64.821	65.031	71.694
cpu [176] :	72.782	89.196	74.777	73.703	66.247	65.419	64.748	63.978
cpu [184] :	63.887	66.080	64.042	65.151	69.661	74.616	63.834	69.824

---

CPU boostlimit in MHz:

cpu [ 0] :	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500
cpu [16] :	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500
cpu [32] :	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500
cpu [48] :	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500
cpu [64] :	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500
cpu [80] :	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500
cpu [96] :	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500
cpu [112] :	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500
cpu [128] :	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500
cpu [144] :	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500	3500

```

| cpu [160] : 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500
| cpu [176] : 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500
|
-----
| CPU core clock current frequency limit in MHz:
|
| cpu [ 0] : 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500 3500
| cpu [ 16] : NA NA NA NA NA NA NA NA NA 3500 3500 3500 3500 3500 3500 3500 3500
| cpu [ 32] : 3500 3500 3500 3500 3500 3500 3500 3500 3500 NA NA NA NA NA NA NA NA
| cpu [ 48] : 3500 3500 3500 3500 3500 3500 3500 3500 3500 NA NA NA NA NA NA NA NA
| cpu [ 64] : NA NA NA NA NA NA NA NA NA 3500 3500 3500 3500 3500 3500 3500 3500
| cpu [ 80] : NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
| cpu [ 96] : NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
| cpu [112] : NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
| cpu [128] : NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
| cpu [144] : NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
| cpu [160] : NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
| cpu [176] : NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
|
-----
*0 Frequency limit source names:
OPN Max
*1 Frequency limit source names:
OPN Max
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 on hsmf protocol version 7 based system:

```

===== E-SMI =====
Usage: ./e_smi_tool [Option]... <INPUT>...
Output Option<s>:
  -h, --help                Show this help message
  -A, --showall             Show all esmi parameter values
  -V, --version             Show e-smi library version
  --testmailbox [SOCKET] [VALUE<0-0xFFFFFFFF>] Test HSMP mailbox interface
  --writemsrallowlist       Write msr-safe allowlist file
Get Option<s>:
  --showcoreenergy [CORE]   Show energy for a given CPU (Joules)
  --showsockenergy         Show energy for all sockets (KJoules)
  --showsockpower          Show power metrics for all sockets (Watts)
  --showcorebl [CORE]      Show Boostlimit for a given CPU (MHz)
  --showsock0res [SOCKET]  Show c0_residency for a given socket (%)
  --showsmufwver           Show SMU FW Version
  --showhsmpprotover       Show HSMP Protocol Version
  --showprochotstatus      Show HSMP PROCHOT status for all sockets
  --showclocks             Show Clock Metrics (MHz) for all sockets
  --showddrbw              Show DDR bandwidth details (Gbps)
  --showdimmtemprange [SOCKET] [DIMM_ADDR] Show dimm temperature range and refresh rate
                           for a given socket and dimm address
  --showdimmthermal [SOCKET] [DIMM_ADDR] Show dimm thermal values for a given socket
                           and dimm address
  --showdimpower [SOCKET] [DIMM_ADDR] Show dimm power consumption for a given
                           socket and dimm address
  --showcclkfreqlimit [CORE] Show current clock frequency limit(MHz) for
                           a given core
  --showsvipower           Show svi based power telemetry of all rails
                           for all sockets
  --showiobw [SOCKET] [LINK<P0-P3,G0-G3>] Show IO aggregate bandwidth for a given
                           socket and linkname
  --showlclkdpmlvl [SOCKET] [NBIOID<0-3>] Show lclk dpm level for a given nbio in a
                           given socket
  --showsockcclkfreqlimit [SOCKET] Show current clock frequency limit(MHz) for
                           a given socket
  --showxgmibw [LINK<P1,P3,G0-G3>] [BW<AGG_BW,RD_BW,WR_BW>] Show xGMI bandwidth for a given socket,
                           linkname and bwtype
  --showcurrpwefficiencymode [SOCKET] Show current power efficiency mode
  --showcpurailisofreqpolicy [SOCKET] Show current CPU ISO frequency policy
  --showdfcstatectrl [SOCKET] Show current DF C-state status
Set Option<s>:
  --setpowerlimit [SOCKET] [POWER] Set power limit for a given socket (mWatts)

```

```

--setcorebl [CORE] [BOOSTLIMIT]           Set boost limit for a given core (MHz)
--setsockbl [SOCKET] [BOOSTLIMIT]         Set Boost limit for a given Socket (MHz)
--apbdisable [SOCKET] [PSTATE<0-2>]       Set Data Fabric Pstate for a given socket
--apbenable [SOCKET]                       Enable the Data Fabric performance boost
algorithm for a given socket
--setxgmiwidth [MIN<0-2>] [MAX<0-2>]       Set xgmi link width in a multi socket system
(MAX >= MIN)
--setlclkdpmlvl [SOCKET] [NBIOID<0-3>] [MIN<0-3>] [MAX<0-3>] Set lclk dpm level for a given nbio in a
given socket (MAX >= MIN)
--setpcielinkratecontrol [SOCKET] [CTL<0-2>] Set rate control for pcie link for a given
socket
--setdfpstatelvl [SOCKET] [MAX<0-2>] [MIN<0-2>] Set df pstate range for a given socket (MAX
<= MIN)
--setgmi3linkwidth [SOCKET] [MIN<0-2>] [MAX<0-2>] Set gmi3 link width for a given socket (MAX
>= MIN)
--setpowerefficiencymode [SOCKET] [MODE<0-5>] Set power efficiency mode for a given socket
--setxgmipstatelvl [MAX<0,1>] [MIN<0,1>] Set xgmi pstate range
--setcpurailisofreqpolicy [SOCKET] [VAL<0,1>] Set CPU ISO frequency policy
--dfcctrl [SOCKET] [VAL<0,1>]             Enable or disable DF c-state
===== End of E-SMI =====

```

Following are the value ranges and other information needed for passing it to tool

1. `---showxgmibw [SOCKET] [LINKNAME] [BWTYPE]`  
LINKNAME :  
Rolling Stones:P0/P1/P2/P3/G0/G1/G2/G3  
Mi300:G0/G1/G2/G3/G4/G5/G6/G7  
Family 0x1A, model 0x00-0x1F:P1/P3/G0/G1/G2/G3  
BWTYPE : AGG\_BW/RD\_BW/WR\_BW
2. `--setxgmiwidth [MIN] [MAX]`  
MIN : MAX : 0 - 2 with MIN <= MAX
3. `--showlclkdpmlvl [SOCKET] [NBIOID]`  
NBIOID : 0 - 3
4. `--apbdisable [SOCKET] [PSTATE]`  
PSTATE : 0 - 2
5. `--setlclkdpmlvl [SOCKET] [NBIOID] [MIN] [MAX]`  
NBIOID : 0 - 3  
MI300A: MIN : MAX : 0 - 2 with MIN <= MAX  
Other platforms: MIN : MAX : 0 - 3 with MIN <= MAX
6. `--setpcielinkratecontrol [SOCKET] [CTL]`  
CTL : 0 - 2
7. `--setpowerefficiencymode [SOCKET] [MODE]`  
Rolling Stones: MODE : 0 - 3  
Family 0x1A model 0x00-0x1F: MODE : 0 - 5
8. `--setdfpstatelvl [SOCKET] [MAX] [MIN]`  
MIN : MAX : 0 - 2 with MAX <= MIN
9. `--setgmi3linkwidth [SOCKET] [MIN] [MAX]`  
MIN : MAX : 0 - 2 with MIN <= MAX
10. `--testmailbox [SOCKET] [VALUE]`  
VALUE : Any 32 bit value

Below is a sample usage to get different system metrics information

1. `e_smi_library/b$ sudo ./e_smi_tool --showcoreenergy 0`  
===== E-SMI =====  
-----  
| core[000] energy | 646.549 Joules |  
-----  
===== End of E-SMI =====
2. `e_smi_library/b$ sudo ./e_smi_tool --showcoreenergy 12 --showsockpower --setpowerlimit 1 220000`  
`--showsockpower`  
===== E-SMI =====  
-----  
| core[012] energy | 73.467 Joules |  
-----  
-----  
| Sensor Name | Socket 0 | Socket 1 |  
-----  
Power (Watts)	174.051	169.451
PowerLimit (Watts)	400.000	220.000
PowerLimitMax (Watts)	400.000	320.000
-----  
Socket[1] power\_limit set to 220.000 Watts successfully  
-----  
| Sensor Name | Socket 0 | Socket 1 |  
-----  
Power (Watts)	174.085	169.431
PowerLimit (Watts)	400.000	220.000
PowerLimitMax (Watts)	400.000	320.000
-----
3. `e_smi_library/b$ ./e_smi_tool --showxgmibandwidth G2 AGG_BW`  
===== E-SMI =====  
-----  
| Current Aggregate bandwidth of xGMI link G2 | 40 Mbps |  
-----  
===== End of E-SMI =====
4. `e_smi_library/b$ sudo ./e_smi_tool --setdfpstatelvl 0 1 2`

```
[sudo] password for user:
===== E-SMI =====
Data Fabric PState range(max:1 min:2) set successfully
===== End of E-SMI =====
```





## Chapter 2

# Module Index

### 2.1 Modules

Here is a list of all modules:

Initialization and Shutdown . . . . .	17
Energy Monitor (RAPL MSR) . . . . .	18
HSMP System Statistics . . . . .	23
Power Monitor . . . . .	29
Power Control . . . . .	32
Performance (Boost limit) Monitor . . . . .	36
Performance (Boost limit) Control . . . . .	38
ddr_bandwidth Monitor . . . . .	40
Temperature Query . . . . .	41
Dimm statistics . . . . .	42
xGMI bandwidth control . . . . .	45
GMI3 width control . . . . .	46
APB and LCLK level control . . . . .	47
Bandwidth Monitor . . . . .	52
Metrics Table . . . . .	54
Test HSMP mailbox . . . . .	56
Auxiliary functions . . . . .	57



## 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 . . . . .	61
<a href="#">dimm_power</a>	DIMM Power(mW), power update rate(ms) and dimm address . . . . .	61
<a href="#">dimm_thermal</a>	DIMM temperature( °C) and update rate(ms) and dimm address . . . . .	62
<a href="#">dpm_level</a>	Max and min LCLK DPM level on a given NBIO ID. Valid max and min DPM level values are 0 - 1	62
<a href="#">hsmp_driver_version</a>	HSMP Driver major and minor version numbers . . . . .	63
<a href="#">link_id_bw_type</a>	LINK name and Bandwidth type Information.It contains link names i.e valid link names are "↔ P0", "P1", "P2", "P3", "P4", "G0", "G1", "G2", "G3", "G4" "G5", "G6", "G7" Valid bandwidth types 1(Aggregate_BW), 2 (Read BW), 4 (Write BW) . . . . .	63
<a href="#">smu_fw_version</a>	Deconstruct raw uint32_t into SMU firmware major and minor version numbers . . . . .	64
<a href="#">temp_range_refresh_rate</a>	Temperature range and refresh rate metrics of a DIMM . . . . .	64



## 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> . . . . .	67
-----------------------------------	----



## Chapter 5

# Module Documentation

### 5.1 Initialization and Shutdown

This function validates the dependencies that exist and initializes the library.

#### Functions

- [esmi\\_status\\_t esmi\\_init](#) (void)  
*Initialize the library, validates the dependencies.*
- void [esmi\\_exit](#) (void)  
*Clean up any allocation done during init.*

#### 5.1.1 Detailed Description

This function validates the dependencies that exist and initializes the library.

#### 5.1.2 Function Documentation

##### 5.1.2.1 esmi\_init()

```
esmi_status_t esmi_init (  
    void )
```

Initialize the library, validates the dependencies.

Search the available dependency entries and initialize the library accordingly.

Return values

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

## 5.2 Energy Monitor (RAPL MSR)

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.

### 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.*
- [esmi\\_status\\_t esmi\\_rapl\\_units\\_hsmp\\_mailbox\\_get](#) (uint32\_t sock\_ind, uint8\_t \*tu, uint8\_t \*esu)  
*Get the RAPL units through HSMP mailbox.*
- [esmi\\_status\\_t esmi\\_rapl\\_package\\_counter\\_hsmp\\_mailbox\\_get](#) (uint32\_t sock\_ind, uint32\_t \*counter1, uint32\_t \*counter0)  
*Get the socket energy counter values for a given socket through mailbox.*
- [esmi\\_status\\_t esmi\\_rapl\\_core\\_counter\\_hsmp\\_mailbox\\_get](#) (uint32\_t core\_ind, uint32\_t \*counter1, uint32\_t \*counter0)  
*Get the core energy counter values for a given socket through mailbox.*
- [esmi\\_status\\_t esmi\\_core\\_energy\\_hsmp\\_mailbox\\_get](#) (uint32\_t core\_ind, uint64\_t \*penergy)  
*Get the core energy for a given core through HSMP mailbox.*
- [esmi\\_status\\_t esmi\\_package\\_energy\\_hsmp\\_mailbox\\_get](#) (uint32\_t sock\_ind, uint64\_t \*penergy)  
*Get the socket energy for a given socket through mailbox.*

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

## Return values

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

<i>ESMI_SUCCESS</i>	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

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

## Return values

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

## 5.2.2.4 esmi\_rapl\_units\_hsmp\_mailbox\_get()

```
esmi_status_t esmi_rapl_units_hsmp_mailbox_get (
    uint32_t sock_ind,
    uint8_t * tu,
    uint8_t * esu )
```

Get the RAPL units through HSMP mailbox.

## Parameters

<i>in</i>	<i>sock_ind</i>	a socket index
<i>in, out</i>	<i>tu</i>	Input buffer to return the time units.
<i>in, out</i>	<i>esu</i>	Input buffer to return the energy units. actual energy will be calculated by multiplying the energy counter value with (1/2) <sup>ESU</sup>

## Return values

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

## 5.2.2.5 esmi\_rapl\_package\_counter\_hsmp\_mailbox\_get()

```
esmi_status_t esmi_rapl_package_counter_hsmp_mailbox_get (
    uint32_t sock_ind,
    uint32_t * counter1,
    uint32_t * counter0 )
```

Get the socket energy counter values for a given socket through mailbox.

Updates the `counter0` and `counter1` with lower 32 bit and upper 32 bit of socket energy counter respectively. Please note these units need to be multiplied with energy units to get actual energy consumption.

## Parameters

in	<i>sock_ind</i>	a socket index
in, out	<i>counter0</i>	Input buffer to return the lower 32 bit of socket energy counter.
in, out	<i>counter1</i>	Input buffer to return the upper 32 bit of socket energy counter.

## Return values

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

5.2.2.6 `esmi_rapl_core_counter_hsmp_mailbox_get()`

```
esmi_status_t esmi_rapl_core_counter_hsmp_mailbox_get (
    uint32_t core_ind,
    uint32_t * counter1,
    uint32_t * counter0 )
```

Get the core energy counter values for a given socket through mailbox.

Updates the `counter0` and `counter1` with lower 32 bit and upper 32 bit of core energy counter respectively. Please note these units need to be multiplied with energy units to get actual energy consumption.

## Parameters

in	<i>core_ind</i>	a core index
in, out	<i>counter0</i>	Input buffer to return the lower 32 bit of core energy.
in, out	<i>counter1</i>	Input buffer to return the upper 32 bit of core energy.

## Return values

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

5.2.2.7 `esmi_core_energy_hsmp_mailbox_get()`

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

Get the core energy for a given core through HSMP mailbox.

Given a core index `core_ind`, this function will calculate the energy of that particular cpu by multiplying counter values obtained from `esmi_rapl_core_counter_hsmp_mailbox_get()` with ESU values from `esmi_rapl_units_hsmp_mailbox_get()`(counter value \* 1/2^ESU) and updates the `penergy` in micro Joules.

## Parameters

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

## Return values

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

## 5.2.2.8 esmi\_package\_energy\_hsmpl\_mailbox\_get()

```
esmi_status_t esmi_package_energy_hsmpl_mailbox_get (
    uint32_t sock_ind,
    uint64_t * penergy )
```

Get the socket energy for a given socket through mailbox.

Given a socket index *sock\_ind*, this function will calculate the energy of that particular socket by multiplying counter values obtained from [esmi\\_rapl\\_package\\_counter\\_hsmpl\\_mailbox\\_get\(\)](#) with ESU values from [esmi\\_rapl\\_units\\_hsmpl\\_mailbox\\_get\(\)](#)(counter value \* 1/2^ESU) and returns it in *penergy* in micro joules.

## Parameters

in	<i>sock_ind</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.

## 5.3 HSMP System Statistics

Below functions to get HSMP System Statistics.

### Functions

- `esmi_status_t esmi_hsmp_driver_version_get` (struct `hsmp_driver_version` \*`hsmp_driver_ver`)  
*Get the HSMP Driver version.*
- `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_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.*
- `esmi_status_t esmi_socket_current_active_freq_limit_get` (uint32\_t `sock_ind`, uint16\_t \*`freq`, char \*\*`src_type`)  
*Get the current active frequency limit of the socket.*
- `esmi_status_t esmi_socket_freq_range_get` (uint8\_t `sock_ind`, uint16\_t \*`fmax`, uint16\_t \*`fmin`)  
*Get the Socket frequency range.*
- `esmi_status_t esmi_current_freq_limit_core_get` (uint32\_t `core_id`, uint32\_t \*`freq`)  
*Get the current active frequency limit of the core.*
- `esmi_status_t esmi_cpurail_isofreq_policy_get` (uint8\_t `sock_ind`, bool \*`val`)  
*Get the CpuRailIsoFreqPolicy.*
- `esmi_status_t esmi_dfctrl_setting_get` (uint8\_t `sock_ind`, bool \*`val`)  
*get the DfctrlEnable.*

### 5.3.1 Detailed Description

Below functions to get HSMP System Statistics.

### 5.3.2 Function Documentation

#### 5.3.2.1 esmi\_hsmp\_driver\_version\_get()

```
esmi_status_t esmi_hsmp_driver_version_get (
    struct hsmp_driver_version * hsmp_driver_ver )
```

Get the HSMP Driver version.

This function will return the HSMP Driver version at `hsmp_driver_ver` Supported on all hsmp protocol versions

## Return values

<a href="#"><i>ESMI_SUCCESS</i></a>	is returned upon successful call.
-------------------------------------	-----------------------------------

**5.3.2.2 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` Supported on all hsmv protocol versions

## Parameters

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

## Return values

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

**5.3.2.3 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`. Supported on all hsmv protocol versions

## Parameters

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

## Return values

<a href="#"><i>ESMI_SUCCESS</i></a>	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. Supported on all hsmp protocol versions

##### 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

<a href="#"><i>ESMI_SUCCESS</i></a>	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. Supported on all hsmp protocol versions

##### 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

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

### 5.3.2.6 esmi\_hsmptproto\_ver\_get()

```
esmi_status_t esmi_hsmptproto_ver_get (
    uint32_t * proto_ver )
```

Get the HSMP interface (protocol) version.

This function will get the HSMP interface version at `proto_ver` Supported on all hsmpt protocol versions

#### Parameters

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

#### Return values

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

### 5.3.2.7 esmi\_socket\_current\_active\_freq\_limit\_get()

```
esmi_status_t esmi_socket_current_active_freq_limit_get (
    uint32_t sock_ind,
    uint16_t * freq,
    char ** src_type )
```

Get the current active frequency limit of the socket.

This function will get the socket frequency and source of this limit Supported on all hsmpt protocol versions

#### Parameters

<code>in</code>	<code>sock_ind</code>	A socket index.
<code>in, out</code>	<code>freq</code>	Input buffer to return the frequency(MHz).
<code>in, out</code>	<code>src_type</code>	Input buffer to return the source of this limit

#### Return values

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

### 5.3.2.8 esmi\_socket\_freq\_range\_get()

```
esmi_status_t esmi_socket_freq_range_get (
    uint8_t sock_ind,
```



```
uint16_t * fmax,
uint16_t * fmin )
```

Get the Socket frequency range.

This function returns the socket frequency range, fmax and fmin. Supported only on hsmp protocol version 5 and 7.

#### Parameters

in	<i>sock_ind</i>	Socket index.
in, out	<i>fmax</i>	Input buffer to return the maximum frequency(MHz).
in, out	<i>fmin</i>	Input buffer to return the minimum frequency(MHz).

#### Return values

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

#### 5.3.2.9 esmi\_current\_freq\_limit\_core\_get()

```
esmi_status_t esmi_current_freq_limit_core_get (
    uint32_t core_id,
    uint32_t * freq )
```

Get the current active frequency limit of the core.

This function returns the core frequency limit for the specified core. Supported only on hsmp protocol version 5 and 7.

#### Parameters

in	<i>core↔ _id</i>	Core index.
in, out	<i>freq</i>	Input buffer to return the core frequency limit(MHz)

#### Return values

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

#### 5.3.2.10 esmi\_cpurail\_isofreq\_policy\_get()

```
esmi_status_t esmi_cpurail_isofreq_policy_get (
    uint8_t sock_ind,
    bool * val )
```

Get the CpuRaillsoFreqPolicy.

This function gets the CpuRaillsoFreqPolicy.

#### Parameters

in	<i>sock_ind</i>	a socket index
in	<i>val</i>	Input buffer containing boolean value which indicates whether all cores on both rails have same frequency limit or different frequency limit. All cores on both rails have same freq limit - 1 Each rail has different independent frequency limit - 0

#### Return values

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

### 5.3.2.11 esmi\_dfc\_ctrl\_setting\_get()

```
esmi_status_t esmi_dfc_ctrl_setting_get (
    uint8_t sock_ind,
    bool * val )
```

get the DfcEnable.

This function gets DF C-state enabling control. DF C-state is a low power state for IOD.

#### Parameters

in	<i>sock_ind</i>	a socket index
in	<i>val</i>	Input buffer holds a boolean which indicates whether DFC is enabled or disabled. Enable DFC - 1 Disable DFC - 0

#### Return values

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

## 5.4 Power Monitor

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

### 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.*
- `esmi_status_t esmi_pwr_svi_telemetry_all_rails_get` (uint32\_t sock\_ind, uint32\_t \*power)  
*Get the SVI based power telemetry for all rails.*
- `esmi_status_t esmi_pwr_efficiency_mode_get` (uint8\_t sock\_ind, uint8\_t \*mode)  
*Get the current power efficiency mode.*

### 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`. Supported on all hsmp protocol versions

#### Parameters

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

#### Return values

<code>ESMI_SUCCESS</code>	is returned upon successful call.
<code>None-zero</code>	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. Supported on all hsmg protocol versions

##### 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`. Supported on all hsmg protocol versions

##### 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.

#### 5.4.2.4 esmi\_pwr\_svi\_telemetry\_all\_rails\_get()

```
esmi_status_t esmi_pwr_svi_telemetry_all_rails_get (
    uint32_t sock_ind,
    uint32_t * power )
```

Get the SVI based power telemetry for all rails.

This function returns the SVI based power telemetry for all rails. Supported only on hsmp protocol version 5 and 7.

##### Parameters

in	<i>sock_ind</i>	Socket index.
in, out	<i>power</i>	Input buffer to return the power(mW).

##### Return values

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

#### 5.4.2.5 esmi\_pwr\_efficiency\_mode\_get()

```
esmi_status_t esmi_pwr_efficiency_mode_get (
    uint8_t sock_ind,
    uint8_t * mode )
```

Get the current power efficiency mode.

This function returns the current power efficiency mode.

##### Parameters

in	<i>sock_ind</i>	Socket index.
in, out	<i>mode</i>	Input buffer to return the mode. Refer <a href="#">esmi_pwr_efficiency_mode_set</a> for details on the modes

##### Return values

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

## 5.5 Power Control

This function provides a way to control Power Limit.

### 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.*
- [esmi\\_status\\_t esmi\\_pwr\\_efficiency\\_mode\\_set](#) (uint8\_t sock\_ind, uint8\_t mode)  
*Set the power efficiency profile policy.*
- [esmi\\_status\\_t esmi\\_cpurail\\_isofreq\\_policy\\_set](#) (uint8\_t sock\_ind, bool \*val)  
*Set the CpuRailIsoFreqPolicy.*
- [esmi\\_status\\_t esmi\\_dfc\\_enable\\_set](#) (uint8\_t sock\_ind, bool \*val)  
*Set the DfcEnable.*

### 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 socket power reduction occurs if the limit is set below that minimum and also there are independent registers through HSMP and APML whichever is the most constraining between the two is enforced. Supported on all hsmp protocol versions.

#### 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.
------------------------------	-----------------------------------

## Return values

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

## 5.5.2.2 esmi\_pwr\_efficiency\_mode\_set()

```
esmi_status_t esmi_pwr_efficiency_mode_set (
    uint8_t sock_ind,
    uint8_t mode )
```

Set the power efficiency profile policy.

This function will set the power efficiency mode. Supported only on hsmv protocol version 5 and 7.

Power efficiency modes are:

0 = High performance mode: This mode favours core performance. In this mode all df pstates are available and default df pstate and DLWM algorithms are active.

1 = Power efficient mode: This mode limits the boost frequency available to the cores and restricts the DF P-States. This mode also monitors the system load to dynamically adjust performance for maximum power efficiency.

2 = IO performance mode: This mode sets up data fabric to maximize IO performance. This can result in lower core performance to increase the IO throughput.

3 = Balanced Memory Performance Mode: This mode biases the memory subsystem and Infinity Fabric™ performance towards efficiency, by lowering the frequency of the fabric and the width of the xGMI links under light traffic conditions. Core behavior is unaffected. There may be a performance impact under lightly loaded conditions for memory-bound applications compared to the default high performance mode. With higher memory and fabric load, the system becomes similar in performance to the default high performance mode.

4 = Balanced Core Performance Mode: This mode biases toward consistent core performance across varying core utilization levels, by preventing active cores from using the power budget of inactive cores. This mode allows core "boosting" as in the default high performance mode, but does not allow core boost to take advantage of the power budget of inactive cores, resulting in a more efficient operating point for the active cores. The memory subsystem and Infinity Fabric behavior is unaffected. There may be a performance impact under light core utilization conditions compared to the default high performance mode. With high core utilization levels, the performance is similar to the default high performance mode.

5 = Balanced Core and Memory Performance Mode. This mode combines the Balanced Memory Performance and the Balanced Core Performance mode and may result in lower performance under light loads compared to the default high performance mode, but with significant increase in efficiency under light loads. Performance in this mode will be similar to the default high performance mode as the system load increases.

## Parameters

in	<i>sock_ind</i>	A socket index.
in	<i>mode</i>	Power efficiency mode to be set.

## Return values

<i>ESMI_SUCCESS</i>	is returned upon successful call.
---------------------	-----------------------------------

## Return values

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

**5.5.2.3 esmi\_cpurail\_isofreq\_policy\_set()**

```
esmi_status_t esmi_cpurail_isofreq_policy_set (
    uint8_t sock_ind,
    bool * val )
```

Set the CpuRailIsoFreqPolicy.

This function sets the CpuRailIsoFreqPolicy.

If a socket wide limit (e.g. PPT) is setting the core clock frequency, then this setting has no effect. For other limiters specific to CPU power rails (e.g. TDC), this policy allows or disables independent core clocks per rail (VDDCR\_CPU0 or VDDCR\_CPU1).

## Parameters

in	<i>sock_ind</i>	a socket index
in	<i>val</i>	Input buffer to contain a boolean which indicates whether all cores on both rails have same frequency limit or different frequency limit. All cores on both rails have same freq limit - 1 Each rail has different independent frequency limit - 0

## Return values

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

**5.5.2.4 esmi\_dfc\_enable\_set()**

```
esmi_status_t esmi_dfc_enable_set (
    uint8_t sock_ind,
    bool * val )
```

Set the DfcEnable.

This function sets DF C-state enabling control. DF C-state is a low power state for IOD.

## Parameters

in	<i>sock_ind</i>	a socket index
in	<i>val</i>	Input buffer holds a boolean which indicates whether to disable DFC or to enable DFC. Enable DFC - 1 Disable DFC - 0



Return values

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

## 5.6 Performance (Boost limit) Monitor

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

### 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 provides the frequency currently enforced through [esmi\\_core\\_boostlimit\\_set\(\)](#) and [esmi\\_socket\\_boostlimit\\_set\(\)](#) APIs for a particular `cpu_ind`. Supported on all hsmp protocol versions. Please note: there are independent registers through HSMP and APML. This message provides boost limit associated with HSMP only.

#### 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 Supported on all hsmp protocol versions

#### 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

Below functions provide ways to control Boost limit values.

### 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.*

### 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`.

Note: Even though set boost limit provides ability to limit frequency on a core basis, if all the cores of a CCX are not programmed for the same boost limit frequency, then the lower-frequency cores are limited to a frequency resolution that can be as low as 20% of the requested frequency. If the specified boost limit frequency of a core is not supported, then the processor selects the next lower supported frequency. For processor with SMT enabled, writes to different APIC ids that map to the same physical core overwrite the previous write to that core. There are independent registers through HSMP and APML whichever is the most constraining between the two is enforced. Supported on all hsmp protocol versions

#### Parameters

in	<i>cpu_ind</i>	a cpu index is a given core to set the boostlimit
in	<i>boostlimit</i>	a uint32_t that indicates the desired boostlimit value of a given core. The maximum accepted value is 65535MHz(UINT16_MAX).

#### Return values

<a href="#">ESMI_SUCCESS</a>	is returned upon successful call.
<i>None-zero</i>	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`. There are independent registers through HSMP and APML whichever is the most constraining between the two is enforced. Supported on all hsmp protocol versions

#### Parameters

in	<i>socket_idx</i>	a socket index to set boostlimit.
in	<i>boostlimit</i>	a uint32_t that indicates the desired boostlimit value of a particular socket. The maximum accepted value is 65535MHz(UINT16_MAX).

#### Return values

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

## 5.8 ddr\_bandwidth Monitor

This function provides the DDR Bandwidth for a system.

### Functions

- [esmi\\_status\\_t esmi\\_ddr\\_bw\\_get](#) (uint8\_t sock\_ind, 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. Supported only on hsmp protocol version >= 3.*

### 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 (
    uint8_t sock_ind,
    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. Supported only on hsmp protocol version >= 3.

This function will return the DDR Bandwidth metrics `ddr_bw` for DDR connected to a socket.

#### Parameters

in	<i>sock_ind</i>	a socket index provided.
in, out	<i>ddr_bw</i>	Input buffer to return the DDR bandwidth metrics, contains max_bw, utilized_bw and utilized_pct.

#### Return values

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

## 5.9 Temperature Query

This function provides the current temperature value in degree C.

### Functions

- `esmi_status_t esmi_socket_temperature_get (uint32_t sock_ind, uint32_t *ptmon)`  
*Get temperature monitor for a given socket.*

#### 5.9.1 Detailed Description

This function provides the current temperature value in degree C.

#### 5.9.2 Function Documentation

##### 5.9.2.1 `esmi_socket_temperature_get()`

```
esmi_status_t esmi_socket_temperature_get (
    uint32_t sock_ind,
    uint32_t * ptmon )
```

Get temperature monitor for a given socket.

This function will return the socket's current temperature in milli degree celsius `ptmon` for a particular `sock_ind`. Supported only on hsmv protocol version-4

##### Parameters

in	<i>sock_ind</i>	a socket index provided.
in, out	<i>ptmon</i>	pointer to a uint32_t that indicates the possible tmon value.

##### Return values

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

## 5.10 Dimm statistics

This function provides the dimm temperature, power and update rates.

### Functions

- [esmi\\_status\\_t esmi\\_dimm\\_temp\\_range\\_and\\_refresh\\_rate\\_get](#) (uint8\_t sock\_ind, uint8\_t dimm\_addr, struct [temp\\_range\\_refresh\\_rate](#) \*rate)  
*Get dimm temperature range and refresh rate.*
- [esmi\\_status\\_t esmi\\_dimm\\_power\\_consumption\\_get](#) (uint8\_t sock\_ind, uint8\_t dimm\_addr, struct [dimm\\_power](#) \*dimm\_pow)  
*Get dimm power consumption and update rate.*
- [esmi\\_status\\_t esmi\\_dimm\\_thermal\\_sensor\\_get](#) (uint8\_t sock\_ind, uint8\_t dimm\_addr, struct [dimm\\_thermal](#) \*dimm\_temp)  
*Get dimm thermal sensor.*

### 5.10.1 Detailed Description

This function provides the dimm temperature, power and update rates.

### 5.10.2 Function Documentation

#### 5.10.2.1 esmi\_dimm\_temp\_range\_and\_refresh\_rate\_get()

```
esmi_status_t esmi_dimm_temp_range_and_refresh_rate_get (
    uint8_t sock_ind,
    uint8_t dimm_addr,
    struct temp_range_refresh_rate * rate )
```

Get dimm temperature range and refresh rate.

This function returns the per DIMM temperature range and refresh rate from the MR4 register. Supported only on hsmp protocol version 5 and 7.

#### Parameters

in	<i>sock_ind</i>	Socket index through which the DIMM can be accessed
in	<i>dimm_addr</i>	DIMM identifier, follow "HSMP DIMM Addr encoding".
in, out	<i>rate</i>	Input buffer of type struct <a href="#">temp_range_refresh_rate</a> with refresh rate and temp range.

#### Return values

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



### 5.10.2.2 esmi\_dimm\_power\_consumption\_get()

```
esmi_status_t esmi_dimm_power_consumption_get (
    uint8_t sock_ind,
    uint8_t dimm_addr,
    struct dimm_power * dimm_pow )
```

Get dimm power consumption and update rate.

This function returns the DIMM power and update rate Supported only on hsmp protocol version 5 and 7.

#### Parameters

in	<i>sock_ind</i>	Socket index through which the DIMM can be accessed.
in	<i>dimm_addr</i>	DIMM identifier, follow "HSMP DIMM Address encoding".
in, out	<i>dimm_pow</i>	Input buffer of type struct <a href="#">dimm_power</a> containing power(mW), update rate(ms) and dimm address.

#### Return values

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

### 5.10.2.3 esmi\_dimm\_thermal\_sensor\_get()

```
esmi_status_t esmi_dimm_thermal_sensor_get (
    uint8_t sock_ind,
    uint8_t dimm_addr,
    struct dimm_thermal * dimm_temp )
```

Get dimm thermal sensor.

This function will return the DIMM thermal sensor(2 sensors per DIMM) and update rate Supported only on hsmp protocol version 5 and 7.

#### Parameters

in	<i>sock_ind</i>	Socket index through which the DIMM can be accessed.
in	<i>dimm_addr</i>	DIMM identifier, follow "HSMP DIMM Address encoding".
in, out	<i>dimm_temp</i>	Input buffer of type struct <a href="#">dimm_thermal</a> which contains temperature(°C), update rate(ms) and dimm address Update rate value can vary from 0 to 511ms. Update rate of "0" means last update was < 1ms and 511ms means update was >= 511ms.

## Return values

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

## 5.11 xGMI bandwidth control

This function provides a way to control width of the xgmi links connected in multisocket systems.

### Functions

- [esmi\\_status\\_t esmi\\_xgmi\\_width\\_set](#) (uint8\_t min, uint8\_t max)  
*Set xgmi width for a multi socket system. values range from 0 to 2.*

#### 5.11.1 Detailed Description

This function provides a way to control width of the xgmi links connected in multisocket systems.

#### 5.11.2 Function Documentation

##### 5.11.2.1 esmi\_xgmi\_width\_set()

```
esmi_status_t esmi_xgmi_width_set (
    uint8_t min,
    uint8_t max )
```

Set xgmi width for a multi socket system. values range from 0 to 2.

0 => 4 lanes on family 19h model 10h and 2 lanes on other models.

1 => 8 lanes.

2 => 16 lanes.

Supported on all hsmv protocol versions.

This function will set the xgmi width `min` and `max` for all the sockets in the system

##### Parameters

in	<i>min</i>	minimum xgmi link width, varies from 0 to 2 with min <= max.
in	<i>max</i>	maximum xgmi link width, varies from 0 to 2.

##### Return values

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

## 5.12 GMI3 width control

This function provides a way to control global memory interconnect link width.

### Functions

- [esmi\\_status\\_t esmi\\_gmi3\\_link\\_width\\_range\\_set](#) (uint8\_t sock\_ind, uint8\_t min\_link\_width, uint8\_t max\_link\_width)  
*Set gmi3 width.*

### 5.12.1 Detailed Description

This function provides a way to control global memory interconnect link width.

### 5.12.2 Function Documentation

#### 5.12.2.1 esmi\_gmi3\_link\_width\_range\_set()

```
esmi_status_t esmi_gmi3_link_width_range_set (
    uint8_t sock_ind,
    uint8_t min_link_width,
    uint8_t max_link_width )
```

Set gmi3 width.

This function will set the global memory interconnect width. Values can be 0, 1 or 2.

0 => Quarter width

1 => Half width

2 => Full width

Supported only on hsmp protocol version 5 and 7.

#### Parameters

in	<i>sock_ind</i>	Socket index.
in	<i>min_link_width</i>	Minimum link width to be set.
in	<i>max_link_width</i>	Maximum link width to be set.

#### Return values

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

## 5.13 APB and LCLK level control

This functions provides a way to control APB and lclk values.

### Functions

- `esmi_status_t esmi_apb_enable` (uint32\_t sock\_ind)  
*Enable automatic P-state selection.*
- `esmi_status_t esmi_apb_disable` (uint32\_t sock\_ind, uint8\_t pstate)  
*Set data fabric P-state to user specified value.*
- `esmi_status_t esmi_socket_lclk_dpm_level_set` (uint32\_t sock\_ind, uint8\_t nbio\_id, uint8\_t min, uint8\_t max)  
*Set lclk dpm level.*
- `esmi_status_t esmi_socket_lclk_dpm_level_get` (uint8\_t sock\_ind, uint8\_t nbio\_id, struct `dpm_level` \*nbio)  
*Get lclk dpm level.*
- `esmi_status_t esmi_pcie_link_rate_set` (uint8\_t sock\_ind, uint8\_t rate\_ctrl, uint8\_t \*prev\_mode)  
*Set pcie link rate.*
- `esmi_status_t esmi_df_pstate_range_set` (uint8\_t sock\_ind, uint8\_t max\_pstate, uint8\_t min\_pstate)  
*Set data fabric pstate range.*
- `esmi_status_t esmi_xgmi_pstate_range_set` (uint8\_t max\_pstate, uint8\_t min\_pstate)  
*Set xgmi pstate range.*

### 5.13.1 Detailed Description

This functions provides a way to control APB and lclk values.

### 5.13.2 Function Documentation

#### 5.13.2.1 `esmi_apb_enable()`

```
esmi_status_t esmi_apb_enable (
    uint32_t sock_ind )
```

Enable automatic P-state selection.

Given a socket index `sock_ind`, this function will enable performance boost algorithm. By default, an algorithm adjusts DF P-States automatically in order to optimize performance. However, this default may be changed to a fixed DF P-State through a CBS option at boottime. APBDisable may also be used to disable this algorithm and force a fixed DF P-State. Supported on all hsmp protocol versions

NOTE: While the socket is in PC6 or if PROCHOT\_L is asserted, the lowest DF P-State (highest value) is enforced regardless of the APBEnable/APBDisable state.

#### Parameters

in	<code>sock_ind</code>	a socket index
----	-----------------------	----------------

## Return values

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

**5.13.2.2 esmi\_apb\_disable()**

```
esmi_status_t esmi_apb_disable (
    uint32_t sock_ind,
    uint8_t pstate )
```

Set data fabric P-state to user specified value.

This function will set the desired P-state at `pstate`. Acceptable values for the P-state are 0(highest) - 2 (lowest) If the PC6 or PROCHOT\_L is asserted, then the lowest DF pstate is enforced regardless of the APBenable/APBdiable states. Supported on all hsmmp protocol versions.

## Parameters

in	<i>sock_ind</i>	a socket index
in	<i>pstate</i>	a uint8_t that indicates the desired P-state to set.

## Return values

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

**5.13.2.3 esmi\_socket\_lclk\_dpm\_level\_set()**

```
esmi_status_t esmi_socket_lclk_dpm_level_set (
    uint32_t sock_ind,
    uint8_t nbio_id,
    uint8_t min,
    uint8_t max )
```

Set lclk dpm level.

This function will set the lclk dpm level / nbio pstate for the specified `nbio_id` in a specified socket `sock_ind` with provided values `min` and `max`. Supported on hsmmp protocol version  $\geq 2$

## Parameters

in	<i>sock_ind</i>	socket index.
in	<i>nbio_id</i>	northbridge number varies from 0 to 3.
in	<i>min</i>	pstate minimum value, varies from 0(lowest) to 3(highest) with $\text{min} \leq \text{max}$
in	<i>max</i>	pstate maximum value, varies from 0 to 3.

## Return values

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

5.13.2.4 `esmi_socket_lclk_dpm_level_get()`

```
esmi_status_t esmi_socket_lclk_dpm_level_get (
    uint8_t sock_ind,
    uint8_t nbio_id,
    struct dpm_level * nbio )
```

Get lclk dpm level.

This function will get the lclk dpm level. DPM level is an encoding to represent PCIe link frequency. DPM levels can be set from APML also. This API gives current levels which may have been set from either APML or HSMP.

Supported in hsmp protocol version 5 and 7.

## Parameters

in	<i>sock_ind</i>	Socket index
in	<i>nbio_id</i>	NBIO id(0-3)
in, out	<i>nbio</i>	Input buffer of struct <a href="#"><code>dpm_level</code></a> type to hold min and max dpm levels

## Return values

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

5.13.2.5 `esmi_pcie_link_rate_set()`

```
esmi_status_t esmi_pcie_link_rate_set (
    uint8_t sock_ind,
    uint8_t rate_ctrl,
    uint8_t * prev_mode )
```

Set pcie link rate.

This function will set the pcie link rate to gen4/5 or auto detection based on bandwidth utilisation. Values are: 0 => auto detect bandwidth utilisation and set link rate

1 => Limit at gen4 rate

2 => Limit at gen5 rate

Supported only on hsmp protocol version 5 and 7.

## Parameters

in	<i>sock_ind</i>	Socket index.
in	<i>rate_ctrl</i>	Control value to be set.
in, out	<i>prev_mode</i>	Input buffer to hold the previous mode.

## Return values

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

5.13.2.6 **esmi\_df\_pstate\_range\_set()**

```
esmi_status_t esmi_df_pstate_range_set (
    uint8_t sock_ind,
    uint8_t max_pstate,
    uint8_t min_pstate )
```

Set data fabric pstate range.

This function will set the max and min pstates for the data fabric. Acceptable values for the P-state are 0(highest) - 2 (lowest) with max <= min. DF pstate range can be set from both HSMP and APML, the most recent of the two is enforced. Supported only on hsmp protocol version 5 and 7.

## Parameters

in	<i>sock_ind</i>	a socket index.
in	<i>max_pstate</i>	Maximum pstate value to be set.
in	<i>min_pstate</i>	Minimum pstate value to be set.

## Return values

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

5.13.2.7 **esmi\_xgmi\_pstate\_range\_set()**

```
esmi_status_t esmi_xgmi_pstate_range_set (
    uint8_t max_pstate,
    uint8_t min_pstate )
```

Set xgmi pstate range.

This function will set the max and min xgmi pstate. Acceptable values for the P-state are 0(high performance) and 1(low performance) with max\_state <= min\_state. XGMI pstate range can be set from both HSMP and APML, the most recent of the two is enforced.



## Parameters

in	<i>max_pstate</i>	Maximum pstate value to be set.
in	<i>min_pstate</i>	Minimum pstate value to be set.

## Return values

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

## 5.14 Bandwidth Monitor

This function provides the IO and xGMI bandiwdth.

### Functions

- `esmi_status_t esmi_current_io_bandwidth_get` (uint8\_t sock\_ind, struct `link_id_bw_type` link, uint32\_t \*io\_↵  
bw)  
*Get IO bandwidth on IO link.*
- `esmi_status_t esmi_current_xgmi_bw_get` (uint8\_t sock\_ind, struct `link_id_bw_type` link, uint32\_t \*xgmi\_bw)  
*Get xGMI bandwidth.*

### 5.14.1 Detailed Description

This function provides the IO and xGMI bandiwdth.

### 5.14.2 Function Documentation

#### 5.14.2.1 esmi\_current\_io\_bandwidth\_get()

```
esmi_status_t esmi_current_io_bandwidth_get (
    uint8_t sock_ind,
    struct link_id_bw_type link,
    uint32_t * io_bw )
```

Get IO bandwidth on IO link.

This function returns the IO Aggregate bandwidth for the given link id. Supported only on hsmg protocol version 5 and 7.

#### Parameters

in	<i>sock_ind</i>	Socket index.
in	<i>link</i>	structure containing link_id(Link encoding values of given link) and bwtype info.
in, out	<i>io_bw</i>	Input buffer for bandwidth data in Mbps.

#### Return values

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

### 5.14.2.2 esmi\_current\_xgmi\_bw\_get()

```
esmi_status_t esmi_current_xgmi_bw_get (
    uint8_t sock_ind,
    struct link_id_bw_type link,
    uint32_t * xgmi_bw )
```

Get xGMI bandwidth.

This function will read xGMI bandwidth in Mbps for the specified link and bandwidth type in a multi socket system. Supported only on hsmp protocol version 5 and 7.

#### Parameters

in	<i>sock_ind</i>	Socket index.
in	<i>link</i>	structure containing link_id(Link encoding values of given link) and bwtype info.
in, out	<i>xgmi_bw</i>	Input buffer for bandwidth data in Mbps.

#### Return values

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

## 5.15 Metrics Table

The following functions are assigned for CPU relative functionality, which is expected to be compatible with Epyc products.

### Functions

- `esmi_status_t esmi_metrics_table_version_get` (uint32\_t \*metrics\_version)  
*Get metrics table version.*
- `esmi_status_t esmi_metrics_table_get` (uint8\_t sock\_ind, struct hsmp\_metric\_table \*metrics\_table)  
*Get metrics table.*
- `esmi_status_t esmi_dram_address_metrics_table_get` (uint8\_t sock\_ind, uint64\_t \*dram\_addr)  
*Get the DRAM address for the metrics table.*

### 5.15.1 Detailed Description

The following functions are assigned for CPU relative functionality, which is expected to be compatible with Epyc products.

### 5.15.2 Function Documentation

#### 5.15.2.1 esmi\_metrics\_table\_version\_get()

```
esmi_status_t esmi_metrics_table_version_get (
    uint32_t * metrics_version )
```

Get metrics table version.

Get the version number[31:0] of metrics table

#### Parameters

<code>in, out</code>	<code>metrics_version</code>	input buffer to return the metrics table version.
----------------------	------------------------------	---

#### Return values

<code>ESMI_SUCCESS</code>	is returned upon successful call.
<code>Non-zero</code>	is returned upon failure.

#### 5.15.2.2 esmi\_metrics\_table\_get()

```
esmi_status_t esmi_metrics_table_get (
```

```
uint8_t sock_ind,
struct hsmc_metric_table * metrics_table )
```

Get metrics table.

Read the metrics table

#### Parameters

in	<i>sock_ind</i>	Socket index.
in, out	<i>metrics_table</i>	input buffer to return the metrics table.

#### Return values

<a href="#"><i>ESMI_SUCCESS</i></a>	is returned upon successful call.
<i>Non-zero</i>	is returned upon failure.

### 5.15.2.3 esmi\_dram\_address\_metrics\_table\_get()

```
esmi_status_t esmi_dram_address_metrics_table_get (
    uint8_t sock_ind,
    uint64_t * dram_addr )
```

Get the DRAM address for the metrics table.

Get DRAM address for Metric table transfer

#### Parameters

in	<i>sock_ind</i>	Socket index.
in, out	<i>dram_addr</i>	64-bit DRAM address

#### Return values

<a href="#"><i>ESMI_SUCCESS</i></a>	is returned upon successful call.
<i>Non-zero</i>	is returned upon failure.

## 5.16 Test HSMP mailbox

This is used to check if the HSMP interface is functioning correctly. Increments the input argument value by 1.

### Functions

- [esmi\\_status\\_t esmi\\_test\\_hsmp\\_mailbox](#) (uint8\_t sock\_ind, uint32\_t \*data)  
*Test HSMP mailbox interface.*

#### 5.16.1 Detailed Description

This is used to check if the HSMP interface is functioning correctly. Increments the input argument value by 1.

#### 5.16.2 Function Documentation

##### 5.16.2.1 esmi\_test\_hsmp\_mailbox()

```
esmi_status_t esmi_test_hsmp_mailbox (
    uint8_t sock_ind,
    uint32_t * data )
```

Test HSMP mailbox interface.

[31:0] = input value

##### Parameters

in	<i>sock_ind</i>	Socket index.
in, out	<i>data</i>	input buffer to send input value and to get the output value

##### Return values

<a href="#">ESMI_SUCCESS</a>	is returned upon successful call.
<i>Non-zero</i>	is returned upon failure.

## 5.17 Auxiliary functions

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.

### 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.*

### 5.17.1 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.

### 5.17.2 Function Documentation

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

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

Get the CPU family.

#### Parameters

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

#### Return values

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

### 5.17.2.2 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.

### 5.17.2.3 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.

### 5.17.2.4 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

<code>in, out</code>	<code>cpus</code>	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.

5.17.2.5 `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

<code>in, out</code>	<code>sockets</code>	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.

5.17.2.6 `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

<code>in</code>	<code>socket_idx</code>	a socket index provided.
<code>in, out</code>	<code>pcore_ind</code>	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.

### 5.17.2.7 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 dimm\_power Struct Reference

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

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

## Data Fields

- uint16\_t [power](#): 15  
*Dimm power consumption[31:17](15 bits data)*
- uint16\_t [update\\_rate](#): 9  
*Time since last update[16:8](9 bit data)*
- uint8\_t [dimm\\_addr](#)  
*Dimm address[7:0](8 bit data)*

### 6.2.1 Detailed Description

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

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

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

## 6.3 dimm\_thermal Struct Reference

DIMM temperature(°C) and update rate(ms) and dimm address.

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

## Data Fields

- uint16\_t [sensor](#): 11  
*Dimm thermal sensor[31:21](11 bit data)*
- uint16\_t [update\\_rate](#): 9  
*Time since last update[16:8](9 bit data)*
- uint8\_t [dimm\\_addr](#)  
*Dimm address[7:0](8 bit data)*
- float [temp](#)  
*temperature in degree celcius*

### 6.3.1 Detailed Description

DIMM temperature(°C) and update rate(ms) and dimm address.

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

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

## 6.4 dpm\_level Struct Reference

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

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

## Data Fields

- `uint8_t max_dpm_level`  
*Max LCLK DPM level[15:8](8 bit data)*
- `uint8_t min_dpm_level`  
*Min LCLK DPM level[7:0](8 bit data)*

### 6.4.1 Detailed Description

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

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

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

## 6.5 hsmpt\_driver\_version Struct Reference

HSMP Driver major and minor version numbers.

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

## Data Fields

- `uint32_t major`  
*Major version number.*
- `uint32_t minor`  
*Minor version number.*

### 6.5.1 Detailed Description

HSMP Driver major and minor version numbers.

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

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

## 6.6 link\_id\_bw\_type Struct Reference

LINK name and Bandwidth type Information. It contains link names i.e valid link names are "P0", "P1", "P2", "P3", "P4", "G0", "G1", "G2", "G3", "G4", "G5", "G6", "G7". Valid bandwidth types 1 (Aggregate\_BW), 2 (Read BW), 4 (Write BW).

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

## Data Fields

- [io\\_bw\\_encoding bw\\_type](#)  
*Bandwidth Type Information [1, 2, 4].*
- `char * link\_name`  
*Link name [P0, P1, G0, G1 etc].*

### 6.6.1 Detailed Description

LINK name and Bandwidth type Information. It contains link names i.e valid link names are "P0", "P1", "P2", "P3", "P4", "G0", "G1", "G2", "G3", "G4", "G5", "G6", "G7" Valid bandwidth types 1 (Aggregate\_BW), 2 (Read BW), 4 (Write BW).

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

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

## 6.7 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*

### 6.7.1 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](#)

## 6.8 temp\_range\_refresh\_rate Struct Reference

temperature range and refresh rate metrics of a DIMM

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

## Data Fields

- uint8\_t [range](#): 3  
*temp range[2:0](3 bit data)*
- uint8\_t [ref\\_rate](#): 1  
*DDR refresh rate mode[3](1 bit data)*

### 6.8.1 Detailed Description

temperature range and refresh rate metrics of a DIMM

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

```
#include <stdbool.h>
#include <asm/amd_hsmpt.h>
```

#### Data Structures

- struct [hsmpt\\_driver\\_version](#)  
*HSMP Driver major and minor version numbers.*
- struct [smu\\_fw\\_version](#)  
*Deconstruct raw uint32\_t into SMU firmware major and minor version numbers.*
- struct [ddr\\_bw\\_metrics](#)  
*DDR bandwidth metrics.*
- struct [temp\\_range\\_refresh\\_rate](#)  
*temperature range and refresh rate metrics of a DIMM*
- struct [dimm\\_power](#)  
*DIMM Power(mW), power update rate(ms) and dimm address.*
- struct [dimm\\_thermal](#)  
*DIMM temperature( °C) and update rate(ms) and dimm address.*
- struct [link\\_id\\_bw\\_type](#)  
*LINK name and Bandwidth type Information.It contains link names i.e valid link names are "P0", "P1", "P2", "P3", "P4", "G0", "G1", "G2", "G3", "G4", "G5", "G6", "G7" Valid bandwidth types 1(Aggregate\_BW), 2 (Read BW), 4 (Write BW).*
- struct [dpm\\_level](#)  
*max and min LCLK DPM level on a given NBIO ID. Valid max and min DPM level values are 0 - 1.*

#### Macros

- #define [ENERGY\\_DEV\\_NAME](#) "amd\_energy"  
*Supported Energy driver name.*
- #define [HSMP\\_CHAR\\_DEVFILE\\_NAME](#) "/dev/hsmpt"  
*HSMP device path.*
- #define [HSMP\\_METRICTABLE\\_PATH](#) "/sys/devices/platform/amd\_hsmpt"  
*HSMP MetricTable sysfs path.*
- #define [ARRAY\\_SIZE](#)(arr) (sizeof(arr) / sizeof((arr)[0]))  
*macro to calculate size*
- #define [BIT](#)(N) (1 << N)  
*macro for mask*

## Enumerations

- enum `io_bw_encoding` { `AGG_BW` = BIT(0), `RD_BW` = BIT(1), `WR_BW` = BIT(2) }  
*xGMI Bandwidth Encoding types*
- enum `esmi_status_t` {  
`ESMI_SUCCESS` = 0, `ESMI_INITIALIZED` = 0, `ESMI_NO_ENERGY_DRV`, `ESMI_NO_MSR_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`,  
`ESMI_HSMP_TIMEOUT`, `ESMI_NO_HSMP_MSG_SUP`, `ESMI_PRE_REQ_NOT_SAT`, `ESMI_SMU_BUSY` }  
*Error codes returned by E-SMI functions.*

## Functions

- `esmi_status_t esmi_init` (void)  
*Initialize the library, validates the dependencies.*
- void `esmi_exit` (void)  
*Clean up any allocation done 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_rapl_units_hsmp_mailbox_get` (uint32\_t sock\_ind, uint8\_t \*tu, uint8\_t \*esu)  
*Get the RAPL units through HSMP mailbox.*
- `esmi_status_t esmi_rapl_package_counter_hsmp_mailbox_get` (uint32\_t sock\_ind, uint32\_t \*counter1, uint32\_t \*counter0)  
*Get the socket energy counter values for a given socket through mailbox.*
- `esmi_status_t esmi_rapl_core_counter_hsmp_mailbox_get` (uint32\_t core\_ind, uint32\_t \*counter1, uint32\_t \*counter0)  
*Get the core energy counter values for a given socket through mailbox.*
- `esmi_status_t esmi_core_energy_hsmp_mailbox_get` (uint32\_t core\_ind, uint64\_t \*penergy)  
*Get the core energy for a given core through HSMP mailbox.*
- `esmi_status_t esmi_package_energy_hsmp_mailbox_get` (uint32\_t sock\_ind, uint64\_t \*penergy)  
*Get the socket energy for a given socket through mailbox.*
- `esmi_status_t esmi_hsmp_driver_version_get` (struct `hsmp_driver_version` \*hsmp\_driver\_ver)  
*Get the HSMP Driver version.*
- `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_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.*
- `esmi_status_t esmi_socket_current_active_freq_limit_get` (uint32\_t sock\_ind, uint16\_t \*freq, char \*\*src\_type)  
*Get the socket current active frequency limit and source type.*

- Get the current active frequency limit of the socket.*

  - [esmi\\_status\\_t esmi\\_socket\\_freq\\_range\\_get](#) (uint8\_t sock\_ind, uint16\_t \*fmax, uint16\_t \*fmin)
- Get the Socket frequency range.*

  - [esmi\\_status\\_t esmi\\_current\\_freq\\_limit\\_core\\_get](#) (uint32\_t core\_id, uint32\_t \*freq)
- Get the current active frequency limit of the core.*

  - [esmi\\_status\\_t esmi\\_cpurail\\_isofreq\\_policy\\_get](#) (uint8\_t sock\_ind, bool \*val)
- Get the CpuRailIsoFreqPolicy.*

  - [esmi\\_status\\_t esmi\\_dfc\\_ctrl\\_setting\\_get](#) (uint8\_t sock\_ind, bool \*val)
- get the DfcEnable.*

  - [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\\_pwr\\_svi\\_telemetry\\_all\\_rails\\_get](#) (uint32\_t sock\_ind, uint32\_t \*power)
- Get the SVI based power telemetry for all rails.*

  - [esmi\\_status\\_t esmi\\_pwr\\_efficiency\\_mode\\_get](#) (uint8\_t sock\_ind, uint8\_t \*mode)
- Get the current power efficiency mode.*

  - [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\\_pwr\\_efficiency\\_mode\\_set](#) (uint8\_t sock\_ind, uint8\_t mode)
- Set the power efficiency profile policy.*

  - [esmi\\_status\\_t esmi\\_cpurail\\_isofreq\\_policy\\_set](#) (uint8\_t sock\_ind, bool \*val)
- Set the CpuRailIsoFreqPolicy.*

  - [esmi\\_status\\_t esmi\\_dfc\\_enable\\_set](#) (uint8\_t sock\_ind, bool \*val)
- Set the DfcEnable.*

  - [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\\_ddr\\_bw\\_get](#) (uint8\_t sock\_ind, 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. Supported only on hsmp protocol version >= 3.*

  - [esmi\\_status\\_t esmi\\_socket\\_temperature\\_get](#) (uint32\_t sock\_ind, uint32\_t \*ptmon)
- Get temperature monitor for a given socket.*

  - [esmi\\_status\\_t esmi\\_dimm\\_temp\\_range\\_and\\_refresh\\_rate\\_get](#) (uint8\_t sock\_ind, uint8\_t dimm\_addr, struct [temp\\_range\\_refresh\\_rate](#) \*rate)
- Get dimm temperature range and refresh rate.*

  - [esmi\\_status\\_t esmi\\_dimm\\_power\\_consumption\\_get](#) (uint8\_t sock\_ind, uint8\_t dimm\_addr, struct [dimm\\_power](#) \*dimm\_pow)
- Get dimm power consumption and update rate.*

  - [esmi\\_status\\_t esmi\\_dimm\\_thermal\\_sensor\\_get](#) (uint8\_t sock\_ind, uint8\_t dimm\_addr, struct [dimm\\_thermal](#) \*dimm\_temp)
- Get dimm thermal sensor.*

  - [esmi\\_status\\_t esmi\\_xgmi\\_width\\_set](#) (uint8\_t min, uint8\_t max)

- Set xgmi width for a multi socket system. values range from 0 to 2.*

  - `esmi_status_t esmi_gmi3_link_width_range_set` (uint8\_t sock\_ind, uint8\_t min\_link\_width, uint8\_t max\_↵ link\_width)

*Set gmi3 width.*
- `esmi_status_t esmi_apb_enable` (uint32\_t sock\_ind)

*Enable automatic P-state selection.*
- `esmi_status_t esmi_apb_disable` (uint32\_t sock\_ind, uint8\_t pstate)

*Set data fabric P-state to user specified value.*
- `esmi_status_t esmi_socket_lclk_dpm_level_set` (uint32\_t sock\_ind, uint8\_t nbio\_id, uint8\_t min, uint8\_t max)

*Set lclk dpm level.*
- `esmi_status_t esmi_socket_lclk_dpm_level_get` (uint8\_t sock\_ind, uint8\_t nbio\_id, struct `dpm_level` \*nbio)

*Get lclk dpm level.*
- `esmi_status_t esmi_pcie_link_rate_set` (uint8\_t sock\_ind, uint8\_t rate\_ctrl, uint8\_t \*prev\_mode)

*Set pcie link rate.*
- `esmi_status_t esmi_df_pstate_range_set` (uint8\_t sock\_ind, uint8\_t max\_pstate, uint8\_t min\_pstate)

*Set data fabric pstate range.*
- `esmi_status_t esmi_xgmi_pstate_range_set` (uint8\_t max\_pstate, uint8\_t min\_pstate)

*Set xgmi pstate range.*
- `esmi_status_t esmi_current_io_bandwidth_get` (uint8\_t sock\_ind, struct `link_id_bw_type` link, uint32\_t \*io\_↵ bw)

*Get IO bandwidth on IO link.*
- `esmi_status_t esmi_current_xgmi_bw_get` (uint8\_t sock\_ind, struct `link_id_bw_type` link, uint32\_t \*xgmi\_bw)

*Get xGMI bandwidth.*
- `esmi_status_t esmi_metrics_table_version_get` (uint32\_t \*metrics\_version)

*Get metrics table version.*
- `esmi_status_t esmi_metrics_table_get` (uint8\_t sock\_ind, struct `hsmp_metric_table` \*metrics\_table)

*Get metrics table.*
- `esmi_status_t esmi_dram_address_metrics_table_get` (uint8\_t sock\_ind, uint64\_t \*dram\_addr)

*Get the DRAM address for the metrics table.*
- `esmi_status_t esmi_test_hsmp_mailbox` (uint8\_t sock\_ind, uint32\_t \*data)

*Test HSMP mailbox interface.*
- `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 io\_bw\_encoding

enum `io_bw_encoding`

xGMI Bandwidth Encoding types

Enumerator

AGG_BW	Aggregate Bandwidth.
RD_BW	Read Bandwidth.
WR_BW	Write Bandwidth.

### 7.1.2.2 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_MSR_DRV	MSR driver not found.
ESMI_NO_HSMP_DRV	HSMP driver not found.
ESMI_NO_HSMP_SUP	HSMP 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 be 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.

## Enumerator

ESMI_HSMP_TIMEOUT	HSMP message is timedout.
ESMI_NO_HSMP_MSG_SUP	HSMP message/feature not supported.
ESMI_PRE_REQ_NOT_SAT	Prerequisite to execute the command not satisfied.
ESMI_SMU_BUSY	SMU is busy.

# Index

- AGG\_BW
  - e\_smi.h, [71](#)
- APB and LCLK level control, [47](#)
  - esmi\_apb\_disable, [48](#)
  - esmi\_apb\_enable, [47](#)
  - esmi\_df\_pstate\_range\_set, [50](#)
  - esmi\_pcie\_link\_rate\_set, [49](#)
  - esmi\_socket\_lclk\_dpm\_level\_get, [49](#)
  - esmi\_socket\_lclk\_dpm\_level\_set, [48](#)
  - esmi\_xgmi\_pstate\_range\_set, [50](#)
- Auxiliary functions, [57](#)
  - esmi\_cpu\_family\_get, [57](#)
  - esmi\_cpu\_model\_get, [58](#)
  - esmi\_first\_online\_core\_on\_socket, [59](#)
  - esmi\_get\_err\_msg, [60](#)
  - esmi\_number\_of\_cpus\_get, [58](#)
  - esmi\_number\_of\_sockets\_get, [59](#)
  - esmi\_threads\_per\_core\_get, [58](#)
- Bandwidth Monitor, [52](#)
  - esmi\_current\_io\_bandwidth\_get, [52](#)
  - esmi\_current\_xgmi\_bw\_get, [52](#)
- ddr\_bandwidth Monitor, [40](#)
  - esmi\_ddr\_bw\_get, [40](#)
- ddr\_bw\_metrics, [61](#)
- Dimm statistics, [42](#)
  - esmi\_dimm\_power\_consumption\_get, [43](#)
  - esmi\_dimm\_temp\_range\_and\_refresh\_rate\_get, [42](#)
  - esmi\_dimm\_thermal\_sensor\_get, [43](#)
- dimm\_power, [61](#)
- dimm\_thermal, [62](#)
- dpm\_level, [62](#)
- e\_smi.h, [67](#)
  - AGG\_BW, [71](#)
  - ESMI\_ARG\_PTR\_NULL, [71](#)
  - ESMI\_DEV\_BUSY, [71](#)
  - ESMI\_FILE\_ERROR, [71](#)
  - ESMI\_FILE\_NOT\_FOUND, [71](#)
  - ESMI\_HSMP\_TIMEOUT, [72](#)
  - ESMI\_INITIALIZED, [71](#)
  - ESMI\_INTERRUPTED, [71](#)
  - ESMI\_INVALID\_INPUT, [71](#)
  - ESMI\_IO\_ERROR, [71](#)
  - ESMI\_NO\_DRV, [71](#)
  - ESMI\_NO\_ENERGY\_DRV, [71](#)
  - ESMI\_NO\_HSMP\_DRV, [71](#)
  - ESMI\_NO\_HSMP\_MSG\_SUP, [72](#)
  - ESMI\_NO\_HSMP\_SUP, [71](#)
  - ESMI\_NO\_MEMORY, [71](#)
  - ESMI\_NO\_MSR\_DRV, [71](#)
  - ESMI\_NOT\_INITIALIZED, [71](#)
  - ESMI\_NOT\_SUPPORTED, [71](#)
  - ESMI\_PERMISSION, [71](#)
  - ESMI\_PRE\_REQ\_NOT\_SAT, [72](#)
  - ESMI\_SMU\_BUSY, [72](#)
  - esmi\_status\_t, [71](#)
  - ESMI\_SUCCESS, [71](#)
  - ESMI\_UNEXPECTED\_SIZE, [71](#)
  - ESMI\_UNKNOWN\_ERROR, [71](#)
  - io\_bw\_encoding, [71](#)
  - RD\_BW, [71](#)
  - WR\_BW, [71](#)
- Energy Monitor (RAPL MSR), [18](#)
  - esmi\_all\_energies\_get, [19](#)
  - esmi\_core\_energy\_get, [18](#)
  - esmi\_core\_energy\_hsmp\_mailbox\_get, [21](#)
  - esmi\_package\_energy\_hsmp\_mailbox\_get, [22](#)
  - esmi\_rapl\_core\_counter\_hsmp\_mailbox\_get, [21](#)
  - esmi\_rapl\_package\_counter\_hsmp\_mailbox\_get, [20](#)
  - esmi\_rapl\_units\_hsmp\_mailbox\_get, [20](#)
  - esmi\_socket\_energy\_get, [19](#)
- esmi\_all\_energies\_get
  - Energy Monitor (RAPL MSR), [19](#)
- esmi\_apb\_disable
  - APB and LCLK level control, [48](#)
- esmi\_apb\_enable
  - APB and LCLK level control, [47](#)
- ESMI\_ARG\_PTR\_NULL
  - e\_smi.h, [71](#)
- esmi\_cclk\_limit\_get
  - HSMP System Statistics, [25](#)
- esmi\_core\_boostlimit\_get
  - Performance (Boost limit) Monitor, [36](#)
- esmi\_core\_boostlimit\_set
  - Performance (Boost limit) Control, [38](#)
- esmi\_core\_energy\_get
  - Energy Monitor (RAPL MSR), [18](#)
- esmi\_core\_energy\_hsmp\_mailbox\_get
  - Energy Monitor (RAPL MSR), [21](#)
- esmi\_cpu\_family\_get
  - Auxiliary functions, [57](#)
- esmi\_cpu\_model\_get
  - Auxiliary functions, [58](#)
- esmi\_cpurail\_isofreq\_policy\_get
  - HSMP System Statistics, [27](#)

- esmi\_cpurail\_isofreq\_policy\_set  
Power Control, 34
- esmi\_current\_freq\_limit\_core\_get  
HSMP System Statistics, 27
- esmi\_current\_io\_bandwidth\_get  
Bandwidth Monitor, 52
- esmi\_current\_xgmi\_bw\_get  
Bandwidth Monitor, 52
- esmi\_ddr\_bw\_get  
ddr\_bandwidth Monitor, 40
- ESMI\_DEV\_BUSY  
e\_smi.h, 71
- esmi\_df\_pstate\_range\_set  
APB and LCLK level control, 50
- esmi\_dfc\_ctrl\_setting\_get  
HSMP System Statistics, 28
- esmi\_dfc\_enable\_set  
Power Control, 34
- esmi\_dimm\_power\_consumption\_get  
Dimm statistics, 43
- esmi\_dimm\_temp\_range\_and\_refresh\_rate\_get  
Dimm statistics, 42
- esmi\_dimm\_thermal\_sensor\_get  
Dimm statistics, 43
- esmi\_dram\_address\_metrics\_table\_get  
Metrics Table, 55
- esmi\_fclk\_mclk\_get  
HSMP System Statistics, 24
- ESMI\_FILE\_ERROR  
e\_smi.h, 71
- ESMI\_FILE\_NOT\_FOUND  
e\_smi.h, 71
- esmi\_first\_online\_core\_on\_socket  
Auxiliary functions, 59
- esmi\_get\_err\_msg  
Auxiliary functions, 60
- esmi\_gmi3\_link\_width\_range\_set  
GMI3 width control, 46
- esmi\_hsmp\_driver\_version\_get  
HSMP System Statistics, 23
- esmi\_hsmp\_proto\_ver\_get  
HSMP System Statistics, 25
- ESMI\_HSMP\_TIMEOUT  
e\_smi.h, 72
- esmi\_init  
Initialization and Shutdown, 17
- ESMI\_INITIALIZED  
e\_smi.h, 71
- ESMI\_INTERRUPTED  
e\_smi.h, 71
- ESMI\_INVALID\_INPUT  
e\_smi.h, 71
- ESMI\_IO\_ERROR  
e\_smi.h, 71
- esmi\_metrics\_table\_get  
Metrics Table, 54
- esmi\_metrics\_table\_version\_get  
Metrics Table, 54
- ESMI\_NO\_DRV  
e\_smi.h, 71
- ESMI\_NO\_ENERGY\_DRV  
e\_smi.h, 71
- ESMI\_NO\_HSMP\_DRV  
e\_smi.h, 71
- ESMI\_NO\_HSMP\_MSG\_SUP  
e\_smi.h, 72
- ESMI\_NO\_HSMP\_SUP  
e\_smi.h, 71
- ESMI\_NO\_MEMORY  
e\_smi.h, 71
- ESMI\_NO\_MSR\_DRV  
e\_smi.h, 71
- ESMI\_NOT\_INITIALIZED  
e\_smi.h, 71
- ESMI\_NOT\_SUPPORTED  
e\_smi.h, 71
- esmi\_number\_of\_cpus\_get  
Auxiliary functions, 58
- esmi\_number\_of\_sockets\_get  
Auxiliary functions, 59
- esmi\_package\_energy\_hsmp\_mailbox\_get  
Energy Monitor (RAPL MSR), 22
- esmi\_pcie\_link\_rate\_set  
APB and LCLK level control, 49
- ESMI\_PERMISSION  
e\_smi.h, 71
- ESMI\_PRE\_REQ\_NOT\_SAT  
e\_smi.h, 72
- esmi\_prochot\_status\_get  
HSMP System Statistics, 24
- esmi\_pwr\_efficiency\_mode\_get  
Power Monitor, 31
- esmi\_pwr\_efficiency\_mode\_set  
Power Control, 33
- esmi\_pwr\_svi\_telemetry\_all\_rails\_get  
Power Monitor, 30
- esmi\_rapl\_core\_counter\_hsmp\_mailbox\_get  
Energy Monitor (RAPL MSR), 21
- esmi\_rapl\_package\_counter\_hsmp\_mailbox\_get  
Energy Monitor (RAPL MSR), 20
- esmi\_rapl\_units\_hsmp\_mailbox\_get  
Energy Monitor (RAPL MSR), 20
- ESMI\_SMU\_BUSY  
e\_smi.h, 72
- esmi\_smu\_fw\_version\_get  
HSMP System Statistics, 24
- esmi\_socket\_boostlimit\_set  
Performance (Boost limit) Control, 39
- esmi\_socket\_c0\_residency\_get  
Performance (Boost limit) Monitor, 36
- esmi\_socket\_current\_active\_freq\_limit\_get  
HSMP System Statistics, 26
- esmi\_socket\_energy\_get  
Energy Monitor (RAPL MSR), 19
- esmi\_socket\_freq\_range\_get  
HSMP System Statistics, 26



- esmi\_socket\_lclk\_dpm\_level\_get
    - APB and LCLK level control, [49](#)
  - esmi\_socket\_lclk\_dpm\_level\_set
    - APB and LCLK level control, [48](#)
  - esmi\_socket\_power\_cap\_get
    - Power Monitor, [30](#)
  - esmi\_socket\_power\_cap\_max\_get
    - Power Monitor, [30](#)
  - esmi\_socket\_power\_cap\_set
    - Power Control, [32](#)
  - esmi\_socket\_power\_get
    - Power Monitor, [29](#)
  - esmi\_socket\_temperature\_get
    - Temperature Query, [41](#)
  - esmi\_status\_t
    - e\_smi.h, [71](#)
  - ESMI\_SUCCESS
    - e\_smi.h, [71](#)
  - esmi\_test\_hsmp\_mailbox
    - Test HSMP mailbox, [56](#)
  - esmi\_threads\_per\_core\_get
    - Auxiliary functions, [58](#)
  - ESMI\_UNEXPECTED\_SIZE
    - e\_smi.h, [71](#)
  - ESMI\_UNKNOWN\_ERROR
    - e\_smi.h, [71](#)
  - esmi\_xgmi\_pstate\_range\_set
    - APB and LCLK level control, [50](#)
  - esmi\_xgmi\_width\_set
    - xGMI bandwidth control, [45](#)
- GMI3 width control, [46](#)
- esmi\_gmi3\_link\_width\_range\_set, [46](#)
- HSMP System Statistics, [23](#)
- esmi\_cclk\_limit\_get, [25](#)
  - esmi\_cpurail\_isofreq\_policy\_get, [27](#)
  - esmi\_current\_freq\_limit\_core\_get, [27](#)
  - esmi\_dfc\_ctrl\_setting\_get, [28](#)
  - esmi\_fclk\_mclk\_get, [24](#)
  - esmi\_hsmp\_driver\_version\_get, [23](#)
  - esmi\_hsmp\_proto\_ver\_get, [25](#)
  - esmi\_prochot\_status\_get, [24](#)
  - esmi\_smu\_fw\_version\_get, [24](#)
  - esmi\_socket\_current\_active\_freq\_limit\_get, [26](#)
  - esmi\_socket\_freq\_range\_get, [26](#)
- hsmp\_driver\_version, [63](#)
- Initialization and Shutdown, [17](#)
- esmi\_init, [17](#)
- io\_bw\_encoding
- e\_smi.h, [71](#)
- link\_id\_bw\_type, [63](#)
- Metrics Table, [54](#)
- esmi\_dram\_address\_metrics\_table\_get, [55](#)
  - esmi\_metrics\_table\_get, [54](#)
  - esmi\_metrics\_table\_version\_get, [54](#)
- Performance (Boost limit) Control, [38](#)
- esmi\_core\_boostlimit\_set, [38](#)
  - esmi\_socket\_boostlimit\_set, [39](#)
- Performance (Boost limit) Monitor, [36](#)
- esmi\_core\_boostlimit\_get, [36](#)
  - esmi\_socket\_c0\_residency\_get, [36](#)
- Power Control, [32](#)
- esmi\_cpurail\_isofreq\_policy\_set, [34](#)
  - esmi\_dfc\_enable\_set, [34](#)
  - esmi\_pwr\_efficiency\_mode\_set, [33](#)
  - esmi\_socket\_power\_cap\_set, [32](#)
- Power Monitor, [29](#)
- esmi\_pwr\_efficiency\_mode\_get, [31](#)
  - esmi\_pwr\_svi\_telemetry\_all\_rails\_get, [30](#)
  - esmi\_socket\_power\_cap\_get, [30](#)
  - esmi\_socket\_power\_cap\_max\_get, [30](#)
  - esmi\_socket\_power\_get, [29](#)
- RD\_BW
- e\_smi.h, [71](#)
- smu\_fw\_version, [64](#)
- temp\_range\_refresh\_rate, [64](#)
- Temperature Query, [41](#)
- esmi\_socket\_temperature\_get, [41](#)
- Test HSMP mailbox, [56](#)
- esmi\_test\_hsmp\_mailbox, [56](#)
- WR\_BW
- e\_smi.h, [71](#)
- xGMI bandwidth control, [45](#)
- esmi\_xgmi\_width\_set, [45](#)