LIBSOPHON 使用手册

发行版本 0.4.13

SOPHGO

2025 年 10 月 27 日

目录

CHAPTER 1

声明

法律声明

注意

您购买的产品、服务或特性等应受算能商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，算能对本文档内容不做任何明示或默示的声明或保证。由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

技术支持

地址
北京市海淀区丰豪东路 9 号院中关村集成电路设计园（ICPARK）1 号楼

邮编
100094

网址
https://www.sophgo.com/

邮箱
sales@sophgo.com

电话
+86-10-57590723 +86-10-57590724

CHAPTER 2

Release note

版本	发布日期	说明
V0.1.0	2022.07.12	第一次发布，包含 bmlib, bm-smi and tpu runtime。
V0.2.0	2022.07.30	增加 bmvid；补充文档。
V0.3.0	2022.08.30	增加 soc mode 支持; 增加 bmcv 支持; 支持 bm1684
V0.4.0	2022.09.15	完善 bm1684 支持，增加 soc mode 交叉编译指南; 支持 SC7 加速卡
V0.4.1	2022.09.21	完善 bm1684 soc mode 支持; fix some opencv bug
V0.4.2	2022.10.15	支持 arm pcie mode
V0.4.3	2022.11.15	支持 sc7 hp75 加速卡；支持 rpm 包安装
V0.4.4	2022.12.15	fix bug
V0.4.5	2023.2.7	支持动态算子加载
V0.4.6	2023.3.13	支持 mix mode
V0.4.7	2023.4.13	fix bug
V0.4.8	2023.5.16	add 64 bit dev mem manager
V0.4.9	2023.8.1	add virtual ethernet driver
V1.0.0	2023.9.11	add bm1688

CHAPTER 3

安装 libsophon

libsophon 在不同的 Linux 发行版上提供不同类型的安装方式。请根据您的系统选择对应的方式，不要在一台机器上混用多种安装方式。以下描述中“0.4.9”仅为示例，视当前实际安装版本会有变化。

如果安装了 V3.0.0 版本及之前版本的 SDK 驱动，请先卸载旧的驱动：

```
进入SDK的安装目录，执行：
sudo ./remove_driver_pcie.sh
或者：
sudo rmmod bmsophon
sudo rm -f /lib/modules/$(uname -r)/kernel/drivers/pci/bmsophon.ko
```

如果使用 Debian/Ubuntu 系统：

安装包由三个文件构成，其中“$arch”为当前机器的硬件架构，使用以下命令可以获取当前服务器的 arch：

```
uname -m
```

通常 x86_64 机器对应的硬件架构为 amd64，arm64 机器对应的硬件架构为 arm64：

- sophon-driver_0.4.9_$arch.deb
- sophon-libsophon_0.4.9_$arch.deb
- sophon-libsophon-dev_0.4.9_$arch.deb

其中：sophon-driver 包含了 PCIe 加速卡驱动；sophon-libsophon 包含了运行时环境（库文件、工具等）；sophon-libsophon-dev 包含了开发环境（头文件等）。如果只是在部署环境上安装，则不需要安装 sophon-libsophon-dev。

可以通过如下步骤安装：

```
安装依赖库，只需要执行一次：
sudo apt install dkms libncurses5
安装libsophon：
sudo dpkg -i sophon-*.deb
在终端执行如下命令，或者登出再登入当前用户后即可使用bm-smi等命令：
source /etc/profile
```

安装位置为：

```
/opt/sophon/
├─── driver-0.4.9
├─── libsophon-0.4.9
|    ├─── bin
|    ├─── data
|    ├─── include
|    └─── lib
└─── libsophon-current -> /opt/sophon/libsophon-0.4.9
```

deb 包安装方式并不允许您安装同一个包的多个不同版本，但您可能用其它方式在/opt/sophon 下放置了若干不同版本。在使用 deb 包安装时，/opt/sophon/libsophon-current 会指向最后安装的那个版本。在卸载后，它会指向余下的最新版本（如果有的话）。

在使用 deb 包安装时，如果出现了下面的提示信息

```
modprobe: FATAL: Module bmsophon is in use.
```

需要您手动停止正在使用驱动的程序后，手动执行下面的命令来安装新的 deb 包里的驱动。

```
sudo modprobe -r bmsophon
sudo modprobe bmsophon
```

卸载方式：

注意：如果安装了 sophon-mw 及 sophon-rpc，因为它们对 libsophon 有依赖关系，请先卸载它们。

```
sudo apt remove sophon-driver sophon-libsophon
或者:
sudo dpkg -r sophon-driver
sudo dpkg -r sophon-libsophon-dev
sudo dpkg -r sophon-libsophon
```

如果卸载时遇到问题，可以尝试如下方法：

```
手工卸载驱动：
dkms status
检查输出结果，通常如下：
bmsophon, 0.4.9, 5.15.0-41-generic, x86_64: installed
记下前两个字段，套用到如下命令中：
sudo dkms remove -m bmsophon -v 0.4.9 --all
然后再次卸载驱动：
```

sudo apt remove sophon-driver
sudo dpkg --purge sophon-driver
彻底清除libsophon：
sudo apt purge sophon-libsophon

如果使用 Centos 系统，当前版本仅支持 x86_64：

安装包由三个文件构成，其中“$arch”为当前机器的硬件架构，使用以下命令可以获取当前服务器的 arch：

```
uname -m
```

x86_64 机器对应的安装包名称为：

- sophon-driver-0.4.9-1.$arch.rpm
- sophon-libsophon-0.4.9-1.$arch.rpm
- sophon-libsophon-dev-0.4.9-1.$arch.rpm

安装前需要通过后面“卸载方式”中的步骤卸载旧版本 libsophon，可以通过如下步骤安装：

安装依赖库，只需要执行一次：
sudo yum install -y epel-release
sudo yum install -y dkms
sudo yum install -y ncurses*
安装libsophon：
sudo rpm -ivh sophon-driver-0.4.9-1.x86_64.rpm
sudo rpm -ivh sophon-libsophon-0.4.9-1.x86_64.rpm
sudo rpm -ivh --force sophon-libsophon-dev-0.4.9-1.x86_64.rpm
在终端执行如下命令，或者登出再登入当前用户后即可使用bm-smi等命令：
source /etc/profile

卸载方式：

```
sudo rpm -e sophon-driver
sudo rpm -e sophon-libsophon-dev
sudo rpm -e sophon-libsophon
```

如果使用其它 Linux 系统：

安装包由一个文件构成，其中“$arch”为当前机器的硬件架构，使用以下命令可以获取当前服务器的 arch：

```
uname -m
```

通常 x86_64 机器对应的硬件架构为 x86_64，arm64 机器对应的硬件架构为 aarch64：

- libsophon_0.4.9_$arch.tar.gz

可以通过如下步骤安装：

注意：如果有旧版本，先参考下面的卸载方式步骤卸载旧版本。

```
tar -xzvf libsophon_0.4.9_$arch.tar.gz
sudo cp -r libsophon_0.4.9_$arch/* /
sudo ln -s /opt/sophon/libsophon-0.4.9 /opt/sophon/libsophon-current
```

接下来请先按照您所使用 Linux 发行版的要求搭建驱动编译环境，然后做如下操作：

```
sudo ln -s /opt/sophon/driver-0.4.9/$bin /lib/firmware/bm1684x_firmware.bin
sudo ln -s /opt/sophon/driver-0.4.9/$bin /lib/firmware/bm1684_ddr_firmware.bin
sudo ln -s /opt/sophon/driver-0.4.9/$bin /lib/firmware/bm1684_tcm_firmware.bin
cd /opt/sophon/driver-0.4.9
```

此处 “$bin” 是 bin 文件全名, 对于 bm1684x 板卡，为 a53lite_pkg.bin, 对于 bm1684 板卡，如 bm1684_ddr.bin_v3.1.3-1b97c53f-221230 和 bm1684_tcm.bin_v3.1.3-1b97c53f-221230。

之后就可以编译驱动了（这里不依赖于 dkms）：

```
sudo make SOC_MODE=0 PLATFORM=asic SYNC_API_INT_MODE=1 \
        TARGET_PROJECT=sg_pcie_device FW_SIMPLE=0 \
        PCIE_MODE_ENABLE_CPU=1
sudo cp ./bmsophon.ko /lib/modules/$(uname -r)/kernel/
sudo depmod
sudo modprobe bmsophon
```

最后是一些配置工作：

```
添加库和可执行文件路径:
sudo cp /opt/sophon/libsophon-current/data/libsophon.conf /etc/ld.so.conf.d/
sudo ldconfig
sudo cp /opt/sophon/libsophon-current/data/libsophon-bin-path.sh /etc/profile.d/
在终端执行如下命令，或者登出再登入当前用户后即可使用bm-smi等命令:
source /etc/profile

添加cmake config文件:
sudo mkdir -p /usr/lib/cmake/libsophon
sudo cp /opt/sophon/libsophon-current/data/libsophon-config.cmake /usr/lib/cmake/libsophon/
```

卸载方式：

```
sudo rm -f /etc/ld.so.conf.d/libsophon.conf
sudo ldconfig
sudo rm -f /etc/profile.d/libsophon-bin-path.sh
sudo rm -rf /usr/lib/cmake/libsophon
sudo rmmod bmsophon
sudo rm -f /lib/modules/$(uname -r)/kernel/bmsophon.ko
sudo depmod
sudo rm -f /lib/firmware/bm1684x_firmware.bin
sudo rm -f /lib/firmware/bm1684_ddr_firmware.bin
sudo rm -f /lib/firmware/bm1684_tcm_firmware.bin
sudo rm -f /opt/sophon/libsophon-current
sudo rm -rf /opt/sophon/libsophon-0.4.9
sudo rm -rf /opt/sophon/driver-0.4.9
```


CHAPTER 4

使用 libsophon 开发

4.1 PCIE MODE

在安装完 libsophon 后，推荐您使用 cmake 来将 libsophon 中的库链接到自己的程序中，可以在您程序的 CMakeLists.txt 中添加如下段落：

```
find_package(libsophon REQUIRED)
include_directories(${LIBSOPHON_INCLUDE_DIRS})

add_executable(${YOUR_TARGET_NAME} ${YOUR_SOURCE_FILES})

target_link_libraries(${YOUR_TARGET_NAME} ${the_libbmlib.so})
上面${the_libbmlib.so}即表示要链接libbmlib.so这个库，其它库同理。
```

在您的代码中即可以调用 libbmlib 中的函数：

```
#include <bmlib_runtime.h>
#include <stdio.h>

int main(int argc, char const *argv[])
{
    bm_status_t ret;
    unsigned int card_num = 0;

    ret = bm_get_card_num(&card_num);
    printf("card number: %d/%dn", ret, card_num);

    return 0;
}
```

4.2 SOC MODE

4.2.1 SOC 板卡上编译程序

如果您希望在 SOC 模式的板子上运行的 Linux 环境下进行开发，则需要安装 sophon-soc-libsophon-dev_0.4.8_arm64.deb 工具包，使用以下命令安装。

sudo dpkg -i sophon-soc-libsophon-dev_0.4.8_arm64.deb

安装完成后，您可以参考 PCIE MODE 的开发方法，使用 cmake 将 libsophon 中的库链接到自己的程序中。

4.2.2 x86 交叉编译程序

如果您希望使用 SOPHON SDK 搭建交叉编译环境，您需要用到的是 gcc-aarch64-linux-gnu 工具链以及 sophon-img relesae 包里的 libsophon_soc_0.4.8_aarch64.tar.gz。首先使用如下命令安装 gcc-aarch64-linux-gnu 工具链。

```
sudo apt-get install gcc-aarch64-linux-gnu g++-aarch64-linux-gnu
```

其次解压 sophon-img relesae 包里的 libsophon_soc_0.4.8_aarch64.tar.gz。

1. mkdir -p soc-sdk
2. tar -zxf libsophon_soc_0.4.8_aarch64.tar.gz
3. cp -rf libsophon_soc_0.4.8_aarch64/opt/sophon/libsophon-0.4.8/lib soc-sdk
4. cp -rf libsophon_soc_0.4.8_aarch64/opt/sophon/libsophon-0.4.8/include soc-sdk

libsophon 包含了 bmrt,bmlib 和 bmcv 的库。opencv 和 ffmpeg 相关的库在 sophon-mw-soc_0.4.8_aarch64.tar.gz 中，其目录结构与 libsophon 类似，只需要拷贝 lib 和 include 下所有内容到 soc-sdk 即可。如果需要使用第三方库，可以使用 qemu 在 x86 上构建虚拟环境安装，再将头文件和库文件拷贝到 soc-sdk 目录中，命令如下：

```
构建qemu虚拟环境

sudo apt-get install qemu-user-static
mkdir rootfs
cd rootfs

构建Ubuntu 20.04的rootfs

sudo qemu-debootstrap --arch=arm64 focal .
sudo chroot . qemu-aarch64-static /bin/bash

进入qemu后，安装软件，以gflag为例

apt install software-properties-common
add-apt-repository universe
apt-get update
apt-get install libgflags-dev
```

(续下页)

(接上页)

```
通常情况下安装的so会在/usr/lib/aarch64-linux-gnu/libgflag* 下，只需要拷贝到soc-sdk/lib里即可。
```

下面以如下代码为例，介绍在您的代码中如何使用 SOPHON SDK 制作的 soc-sdk 交叉编译，以调用 libbmlib 中的函数：

```
#include <bmlib_runtime.h>
#include <stdio.h>

int main(int argc, char const *argv[])
{
    bm_status_t ret;
    unsigned int card_num = 0;

    ret = bm_get_card_num(&card_num);
    printf("card number: %d/%dn", ret, card_num);

    return 0;
}
```

首先按照如下步骤创建新的工作目录

```
mkdir -p workspace && pushd workspace
touch CMakeLists.txt
touch get_dev_count.cpp
```

将上面的 c++ 代码导入到 get_dev_count.cpp 中，在 CMakeLists.txt 中添加如下段落：

```
cmake_minimum_required(VERSION 2.8)

set(TARGET_NAME "test_bmlib")

project(${TARGET_NAME} C CXX)

set(CMAKE_C_COMPILER aarch64-linux-gnu-gcc)
set(CMAKE_ASM_COMPILER aarch64-linux-gnu-gcc)
set(CMAKE_CXX_COMPILER aarch64-linux-gnu-g++)

# 该demo链接了bmlib库，所以打开了BM_LIBS
set(BM_LIBS bmlib bmrt)
# 需要链接jpu相关的库可以打开JPU_LIBS
# set(JPU_LIBS bmjpuapi bmjpulite)
# 需要链接opencv相关的库可以打开OPENCV_LIBS
# set(OPENCV_LIBS opencv_imgproc opencv_core opencv_highgui opencv_imgcodecs
      opencv_videoio)
# 引入外部库，可以参看下面导入gflags的方法，打开EXTRA_LIBS
# set(EXTRA_LIBS gflags)

include_directories("${SDK}/include/")
```

(续下页)

(接上页)

```
link_directories("${SDK}/lib/")

set(src get_dev_count.cpp)
get_filename_component(target ${src} NAME_WE)
add_executable(${target} ${src})
target_link_libraries(${target} ${BM_LIBS} pthread dl)
# 未使用OPENCV和FFMPEG等库，所以不需要加下面的链接路径
# target_link_libraries(${target} ${BM_LIBS} ${OPENCV_LIBS} ${FFMPEG_LIBS}
                        ${JPU_LIBS} ${EXTRA_LIBS} pthread dl)
```

接着使用 cmake 来构建程序。

```
mkdir -p build && pushd build
cmake -DSDK=/path_to_sdk/soc-sdk ..
make
```

就可以在 x86 机器上编译出 soc 模式上运行的 aarch64 架构的程序。

上面例子只链接了 bmlib 的库，其它库如 opencv，ffmpeg，其它 lib 同理。

CHAPTER 5

使用 libsophon 工具

libsophon 中带有若干使用工具，在此作介绍。

5.1 bm-smi 使用说明

5.1.1 术语解释

术语	说明
BM1684	算能面向深度学习领域推出的第三代张量处理器
BM1684X	算能面向深度学习领域推出的第四代张量处理器
bm1688	算能面向深度学习领域推出的第五代张量处理器
TPU	芯片内部神经网络处理单元
SOC 模式	一种产品形态，SDK 运行于 A53 AARCH64 平台，TPU 作为平台总线设备
PCIe 模式	一种产品形态，SDK 运行于 X86 平台，BM1684、BM1684X 存在于 PCIe 接口的深度学习计算加速卡上
Drivers	Drivers 是 API 接口访问硬件的通道
Gmem	卡上用于 TPU 加速的 DDR 内存
F	FAULT 故障状态
N/A	此项参数不支持

5.1.2 bm-smi 介绍

bm-smi 工具以界面或者文本的形式显示设备状态信息，如设备的温度、风扇转速等信息；也可使能、禁用或者设置设备的某些功能，如 led、ecc 等。
bm-smi 主要功能有：

1) 查看设备参数和运行时状态
   - 查看设备工作模式（PCIe/SOC）
   - 查看物理板卡 ID
   - 查看设备芯片 ID，所在 PCIe 总线 ID
   - 查看设备温度和功耗
   - 查看设备 ECC 使能与否和纠正次数
   - 查看 gmem 总数和利率
   - 查看 tpu 利用率
   - 查看设备工作频率信息
   - 查看运行时，各进程所占 gmem 大小
   - 查看设备风扇状态
2) 修改板卡参数
   - 禁止、使能 ECC
   - 打开、关闭板上 led 指示灯
3) 执行故障设备的 recovery 操作

   下表列举了 bm-smi 可以获取的设备信息以及在 PCIe 和 SOC 模式下的支持情况：

设备信息	**PCIe 模式**	**SOC 模式**
显示时间日期	支持	支持
SDK Version	支持	支持
Driver Version	支持	支持
物理板卡 id 号	支持	支持
tpu 的设备号	支持	支持
板卡名称	支持	支持
板卡状态	支持	支持
板级温度	支持	不支持
芯片温度	支持	不支持
板级功耗	支持	不支持
模块功耗	支持	不支持
模块电压	支持	不支持
供电电流	支持	不支持
DDR ECC 是否使能	支持	不支持
DDR 使能，纠正错误的次数	支持	不支持
板卡序列号	支持	不支持
PCIe 模式下 domain:b:d.f	支持	不支持
PCIe or SOC mode	支持	支持
最小工作频率	支持	支持
最大工作频率	支持	支持
当前工作频率	支持	支持
板卡最大功耗	支持	不支持
模块的工作电流	支持	不支持
gmem 总数和已使用数量	支持	支持
tpu 的瞬时利用率	支持	支持
风扇转速	支持	不支持
每个进程（或者线程）占用的 gmem 的数量	支持	不支持
文本模式	支持	不支持
参数	支持	仅支持 file、lms、loop

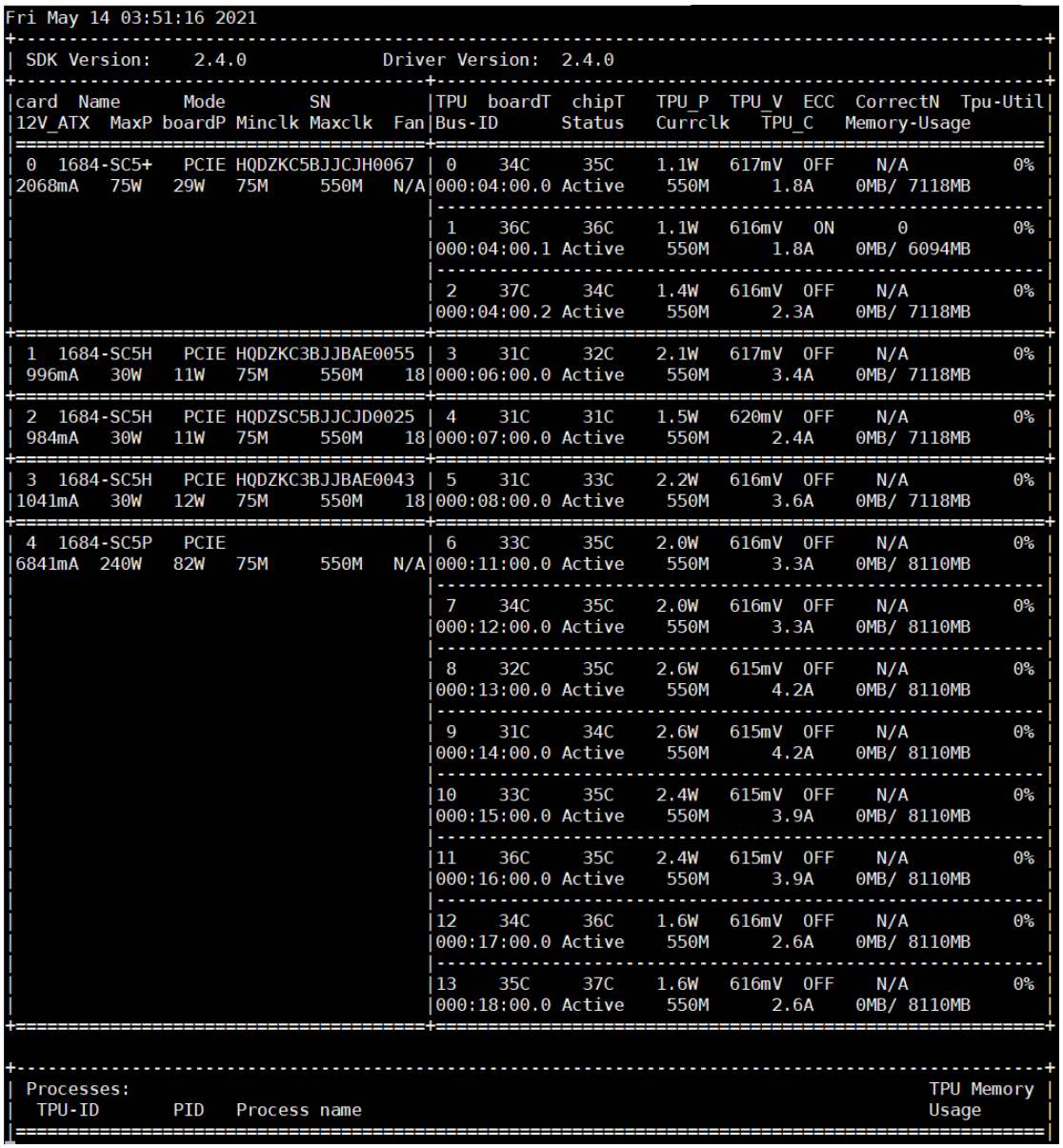


图 1 为 SC5+(三芯)/SC5H/SC5P(八芯) 的显示状态，每张卡之间用 ======== 隔开，最左边显示的板卡级别的属性，右边和中间显示的是单个芯片的状态。

bm-smi 是一个可执行文件，不依赖其他动态库，位于/opt/sophon/libsophon-current/bin 目录下，上图为一个执行 bm-smi 的示意图。

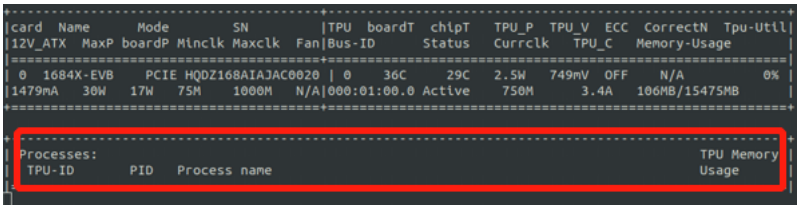
5.1.3 各项参数的含义

下面逐一介绍每个部分代表的含义。

- Fri Aug 7 14:18:57 2020 执行 bm-smi 时的时间日期，这里只是示例，实际执行时可能和这里显示的不同
- SDK Version: 2.3.2：sdk 的版本号，这里只是示例，实际执行时可能和这里显示的不同
- Driver Version: 2.3.2：驱动的版本号，这里只是示例，实际执行时可能和这里显示的不同
- Card：物理板卡 id 号
- Name：板卡名称
- Mode: PCIe or SOC mode
- SN：板卡序列号（共 17 位）
- TPU：tpu 的设备号
- BoardT: 板级温度
- chipT: 芯片温度
- TPU_P:TPU 模块功耗
- TPU_V:TPU 模块电压
- ECC: DDR ECC 是否使能
- CorrectNum: 若 DDR 使能，纠正错误的次数
- Tpu-Util:tpu 的瞬时利用率
- 12V_ATX：板级 12V 供电电流
- MaxP: 板卡最大功耗
- boardP: 板级功耗
- Minclk:tpu 最小工作频率
- Maxclk:tpu 最大工作频率
- Fan: 风扇转速，显示 N/A 表示本卡无风扇，显示 F 表示有风扇故障
- Bus-ID:PCIe 模式下 domain:b:d.f
- Status: 板卡状态，Active 为活动状态，Fault 为故障状态
- Curclk:tpu 当前工作频率，显示的值的颜色根据当前工作频率而不同，550M（bm1684）或 1000M（bm1684x）显示白色，75M 显示红色，其他频率显示黄色；红色和黄色用于提示用户当前工作频率不是最大工作频率。显示不同颜色只在 2.1.0 版本及以上版本才有。
- TPU_C: tpu 模块的工作电流

- Ion-Usage:gmem 总数和已使用数量。板卡上的 memory 有可能分布在不同的地址空间，我们分配的内存都是地址连续的内存，而且由于每次分配的大小不一样，会导致内存的碎片化，所以有可能出现利用率达不到 100% 的情况。
- Vpp-Usage:Vpp heap 总数和已使用数量。
- Npu-Usage:Npu heap 总数和已使用数量。

下面显示的是每个设备上每个进程（或者线程）通过 bmlib handle 申请的 gmem 的数量。



注意事项：

1、因为我们的板卡是支持多任务多用户同时使用的，理论上可以有无限个进程创建无限个 handle 申请 global memory，可以使用上下方向键以及翻页键去查看所有的 process 占用 gmem 的信息，通过标记保存成文件，也是包含所有 process 信息的。

2、process 占用的 gmem 信息，每一行显示的是这个 process 创建的一个 bmlib handle 对应的 gmem，如果这个 process 创建了多个 handle，那么每个 handle 占用的 gmem 信息是单独一行显示的。

5.1.4 具体使用方法和参数

bm-smi 支持的参数有：

- dev (which dev is selected to query, 0xff is for all.) type: int32，default: 255

  用于选择查询或者修改哪个设备的参数，默认所有设备。

  该功能 SOC 模式不支持。

- ecc (ECC on DDR is on or off.)

  ```
  type: string default: ""
  ```

  用来配置 DDR ECC 的使能和关闭，示例如下

  ```
  bm-smi --dev=0x0 --ecc=on

  bm-smi --dev=0x0 --ecc=off
  ```

  执行这个命令时，不要让任何进程使用这个设备，设置完毕后，重启主机生效。

  执行这个命令时，请不要和其他参数一起使用，例如：

  ```
  bm-smi --dev=0x0 --ecc=on --file=~/a.txt 这条命令中的--file=~/a.txt会被忽略，这条命令只会执行ecc相关的动作。
  ```

如果不指定 dev 参数，默认对所有设备做操作。

该功能 SOC 模式不支持。

- file (target file to save smi log.)

```
type: string default: ""
```

可以将设备的状态重定向到文本文档中，使用方法如下：

```
bm-smi --dev=0x0 --file=./bm-smi.log
```

该功能 SOC 模式支持。

- led (pcie card LED status: on/off/blink)

```
type: string default: "on"
```

用来配置板卡 LED 的亮和灭，示例如下

```
bm-smi --dev=0x0 --led=on

bm-smi --dev=0x0 --led=off
```

注意：此功能在 SC5+ 和 SC5P 上支持 on/off/blink，在 SC5H 上支持 on/off，其它板卡类型不支持。SC5+ 板卡只有第一个芯片才能控制 LED 灯的状态，SC5P 拥有 8 个 led，每个设备都对应一个 led，每个 led 都支持单独设置状态。

该功能 SOC 模式不支持。

- lms (sample interval in loop mode.) type: int32 default: 500

用来设置运行 bm-smi 时查询设备状态的时间间隔，默认是 500ms 查询一次，这个参数的最小值是 300ms。该功能 SOC 模式支持。

- loop (true is for loop mode, false is for only once mode.) type: bool，default: true

用来设置运行 bm-smi 时是单次模式还是周期模式，默认周期模式。单次模式下查询一次设备状态后 bm-smi 就退出了；周期模式下按照 lms 为周期反复查询设备状态。示例如下：

```
bm-smi --loop

bm-smi --noloop
```

该功能 SOC 模式支持。

- recovery，使用方式为：发现某个设备 x 功能出现故障，用户将所有业务从这个卡上移走，达到没有任何上层业务和应用使用这个板卡的状态，执行

```
bm-smi --dev=0x(0/1/2/3⋯..) --recovery
```

三芯卡 SC5+ 和八芯卡 SC5P 只支持整卡 recovery，recovery 卡上的任意设备，就会把整个卡 recovery，所以 recovery 的时候需要把整个卡上的任务停掉。

注意：不要在板卡正常工作时执行这个操作，某些服务器不支持这个功能，执行这个功能会导致服务器重启。目前已知不支持的有 dell R740、dell R940、浪潮 5468 和曙光 X785-G30。

该功能 SOC 模式不支持。

- opmode 和 opval，使用方式为：选择 bm-smi 执行的模式以及模式值，兼容前面的标记，例如：

```
bm-smi --opmode=display与bm-smi效果一样

bm-smi --opmode=ecc --opval=on 与bm-smi --ecc=on效果一样。其他标记以此类推。
```

目前 opmode 共有：display(显示)、ecc(使能)、led（指示灯）、recovery 四种操作模式，后续新功能都将以这种方式使用，为了照顾旧版本用户操作习惯，旧版本的使用方法在新版依旧可以使用。(注：目前只有 opmode 为 ecc 和 led 时要搭配使用 opval 去赋值)

2.5.0 display mode 添加了对 heap 和 vpu 内存监控显示，使用方法为

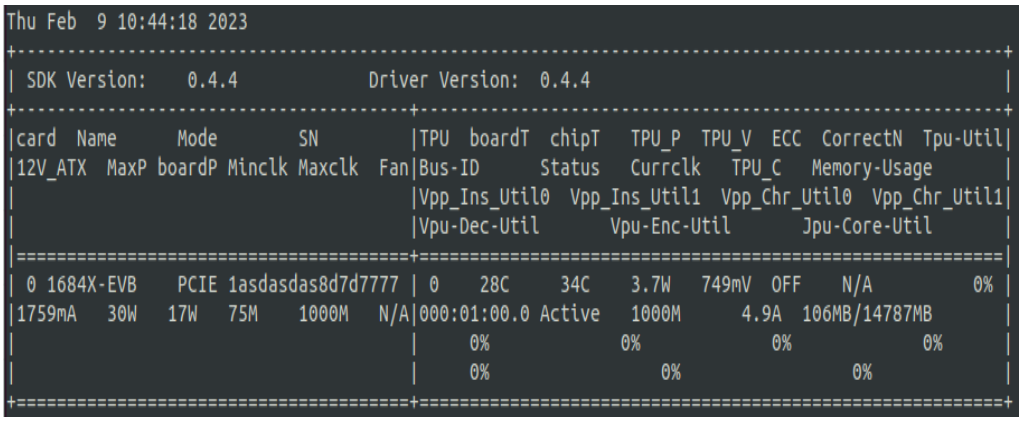
```
bm-smi --opmode=display_memory_detail
```

```
Mon Sep 13 10:47:45 2021
+-----------------------------------------------------------------------------------------------+
| SDK Version:    2.5.0            Driver Version:  2.5.0                                       |
+-----------------------------------+-----------------------------------------------------------+
|card  Name      Mode        SN     |TPU  boardT  chipT   TPU_P  TPU_V  ECC  CorrectN  Tpu-Util|
|12V_ATX  MaxP boardP Minclk Maxclk  Fan|Bus-ID      Status   Currclk   TPU_C   Memory-Usage      |
|                                   |     Heap0         Heap1        Heap2     Vpu-Memory-Usage|
|===================================+===========================================================|
| 0  1684-SC5+   PCIE HQDZKC5AIABAH0001 | 0    53C     52C    1.8W   615mV   ON      0          0% |
|2516mA   75W   34W   75M     550M   N/A|000:01:00.0 Active    550M       2.9A  170MB/ 9646MB      |
|                                   |   0MB/3503MB    0MB/2559MB    0MB/32MB    170MB/ 3552MB  |
|                                   |-----------------------------------------------------------|
|                                   | 1    53C     52C    1.8W   615mV  OFF    N/A          0% |
|                                   |000:01:00.1 Active    550M       3.0A  170MB/11182MB      |
|                                   |   0MB/4015MB    0MB/3071MB    0MB/32MB    170MB/ 4064MB  |
|                                   |-----------------------------------------------------------|
|                                   | 2    52C     52C    2.0W   614mV  OFF    N/A          0% |
|                                   |000:01:00.2 Active    550M       3.2A  170MB/11182MB      |
|                                   |   0MB/4015MB    0MB/3071MB    0MB/32MB    170MB/ 4064MB  |
+===================================+===========================================================+

+-----------------------------------------------------------------------------------------------+
| Processes:                                                                      TPU Memory |
|  TPU-ID       PID   Process name                                                Usage      |
|===============================================================================================|
```

同时还添加了对 vpu 和 jpu 的利用率显示，使用方法为

```
bm-smi --opmode=display_util_detail
```



5.1.5 文本模式介绍

bm-smi 输出的是一个简单的图形界面，描述了板卡的状态，为了满足部分用户对文本信息的需求（便于用脚本 parse 部分参数），支持了文本模式（SOC 模式不支持文本模式），使用方法如下：

```
bm-smi --start_dev=0 --last_dev=2 --text_format

1684-SC5+ PCIE chip0: 0 000:01:00.0 Active 56C 55C 2W 615mV OFF N/A 0% 75M 550M 550M
3.3A 0MB 7086MB
1684-SC5+ PCIE chip1: 1 000:01:00.1 Active 56C 55C 2W 613mV OFF N/A 0% 75M 550M 550M
4.1A 0MB 7086MB
1684-SC5+ PCIE chip2: 2 000:01:00.2 Active 54C 53C 1W 615mV OFF N/A 0% 75M 550M 550M
2.6A 0MB 7086MB
```

上述命令的输出一行文本信息，分为三个区域：

第一个区域：

```
1684-SC5+ PCIE chip0: 0 000:01:00.0 Active 56C 55C 2W 615mV OFF N/A 0% 75M 550M 550M
3.3A 0MB 7086MB
```

三芯卡上的第 0 个 chip 的状态，1684-SC5+ PCIE chip0:
后面的信息依次对应 bm-smi 中的：TPU Bus-ID Status boardT chipT TPU_P TPU_V ECC CorrectN Tpu-Util Minclk Maxclk Curclk TPU_C Memory-Usage

第二个区域：

```
1684-SC5+ PCIE chip1: 1 000:01:00.1 Active 56C 55C 2W 613mV OFF N/A 0% 75M 550M 550M
4.2A 0MB 7086MB
```

三芯卡上的第 1 个 chip 的状态，1684-SC5+ PCIE chip1:

后面的信息依次对应 bm-smi 中的：TPU Bus-ID Status boardT chipT TPU_P TPU_V ECC CorrectN Tpu-Util Minclk Maxclk Curclk TPU_C Memory-Usage

第三个区域：

```
1684-SC5+ PCIE chip2: 2 000:01:00.2 Active 54C 53C 1W 615mV OFF N/A 0% 75M 550M 550M 2.6A 0MB 7086MB
```

三芯卡上的第 2 个 chip 的状态，1684-SC5+ PCIE chip2:
后面的信息依次对应 bm-smi 中的：TPU Bus-ID Status boardT chipT TPU_P TPU_V ECC CorrectN Tpu-Util Minclk Maxclk Curclk TPU_C Memory-Usage

```
注意事项：

1、--start_dev=0 --last_dev=2 表示bm-smi中显示的某张卡的第0个和最后1个chip对应的设备号；

2、--start_dev --last_dev --text_format要一起使用。
```

5.1.6 bm-smi 的 help 信息:

5.1.6.1 PCIe 模式 bm-smi 的 help 信息

```
bm-smi --help

bm-smi: command line brew

usage: bm-smi [--ecc=on/off] [--file=/xx/yy.txt] [--dev=0/1...][--start_dev=x] [--last_dev=y] [--text_format] [--lms=500] [--recovery] [-loop] [--led=on/off/blink]

ecc:

set ecc status, default is off

file:

the target file to save smi log, default is empty.

dev:

which device to be selected to query, default is all.

start_dev:

the first device to be selected to query, must chip0 of one card, default is invalid.
```

(续下页)

(接上页)

```
last_dev:

the last device to be selected to query, default is invalid.

lms:

how many ms of the sample interval, default is 500.

loop:

if -loop (default): smi sample device every lms ms.

if -noloop: smi sample device only once.

recovery:

recovery dev from fault to active status.

text_format:

if true only display attr value from start_dev to last_dev.

led:

pcie card LED status: on/off/blink.

New usage: bm-smi [--opmode=display/ecc/led/recovery][--opval=on/off/...] [--file=/xx/yy.txt]
[--dev=0/1...] [--start_dev=x] [--last_dev=y][--text_format][--lms=500] [-loop]

opmode(default null):

choose different mode,example:display, ecc, led, recovery

display: means open bm-smi window and check info, use like ./bm-smi

ecc: means enable or disable ecc, collocation opval=on/off

led: means modify led status, collocation opval=on/blink/off

recovery: means recovery dev from fault to active status.

opval(default null):

set mode value, use with opmode!

off: for led/ecc

on: for led/ecc

blink: for led
```

(续下页)

(接上页)

```
other flags have same usage, Both usage can be used!


  No modules matched: use -help
```

bm-smi 在 PCIe 模式支持上面 help 列出的所有参数。

5.1.6.2 SOC 模式 bm-smi 的 help 信息

```
bm-smi --help

bm-smi: command line brew

usage: bm-smi [--opmode=display] [--file=/xx/yy.txt] [--lms=500] [-loop]

opmode:

SOC mode only use display for bm-smi.

file:

the target file to save smi log, default is empty.

lms:

how many ms of the sample interval, default is 500.

loop:

if -loop (default): smi sample device every lms ms.

if -noloop: smi sample device only once.


  No modules matched: use -help
```

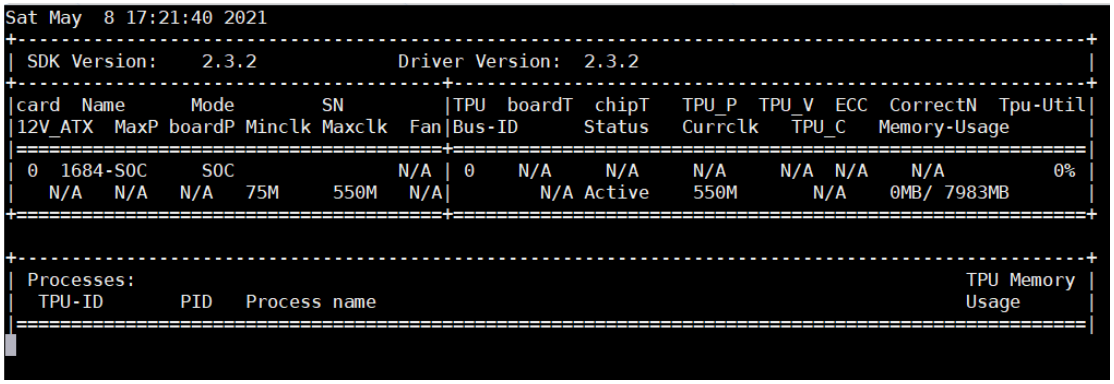
SOC 模式只支持 opmode=display、file、lms 和 loop 参数，其他参数无效。

5.1.7 bm-smi 用于 SOC 模式

PCIe 模式 bm-smi 支持上述所有功能，SOC 模式 bm-smi 界面显示支持功能如图 2 所示，N/A 表示该功能不支持；参数只支持 opmode=display、file、lms 和 loop。

SOC 模式 bm-smi 使用方法：登录 soc 后，直接运行 bm-smi 即可，

```
bm-smi or bm-smi --opmode=display
```



5.2 proc 文件系统使用说明

5.2.1 proc 文件系统介绍

proc 文件系统接口在/proc 节点下创建设备信息节点，用户通过 cat 或者编程的方式读取相关节点获取设备温度、版本等信息。

bm-smi 侧重于以界面的形式直观显示设备信息，proc 文件系统接口侧重于为用户提供编程获取设备信息的接口。下表列举了 proc 文件系统可以获取的设备信息以及在 PCIe 和 SOC 模式下的支持情况：

设备信息	PCIe 模式	SOC 模式
card_num	支持	不支持
chip_num	支持	不支持
chip_num_on_card	支持	不支持
board_power	支持	不支持
board_temp	支持	不支持
chipid	支持	不支持
chip_temp	支持	不支持
dbdf	支持	不支持
dynfreq	支持	不支持
ecc	支持	不支持
maxboardp	支持	不支持
mode	支持	不支持
pcie_cap_speed	支持	不支持
pcie_cap_width	支持	不支持
pcie_link_speed	支持	不支持
pcie_link_width	支持	不支持
pcie_region	支持	不支持
tpuid	支持	不支持
tpu_maxclk	支持	不支持
tpu_minclk	支持	不支持
tpu_freq	支持	不支持

续下页

表 5.1 – 接上页

tpu_power	支持	不支持
firmware_info	支持	不支持
sn	支持	不支持
boot_loader_version	支持	不支持
board_type	支持	不支持
driver_version	支持	不支持
board_version	支持	不支持
mcu_version	支持	不支持
versions	支持	不支持
cdma_in_time	支持	不支持
cdma_in_counter	支持	不支持
cdma_out_time	支持	不支持
cdma_out_counter	支持	不支持
tpu_process_time	支持	不支持
completed_api_counter	支持	不支持
send_api_counter	支持	不支持
tpu_volt	支持	不支持
tpu_cur	支持	不支持
fan_speed	支持	不支持
media	支持	不支持
a53_enable	支持	不支持
arm9_cache	支持	不支持
bmcpu_status	支持	不支持
bom_version	支持	不支持
boot_mode	支持	不支持
clk	支持	不支持
ddr_capacity	支持	不支持
dumpreg	支持	不支持
heap	支持	不支持
location	支持	不支持
pcb_version	支持	不支持
pmu_infos	支持	不支持
status	支持	不支持
vddc_power	支持	不支持
vddphy_power	支持	不支持
jpu	不支持	支持
vpu	不支持	支持

驱动安装时系统根据板卡数量依次创建/proc/bmsophon/card0…n 目录，其中 card0 目录对应存放第 0 张板卡的信息，card1 目录对应存放第 1 张板卡信息，依次类推。

在板卡目录下根据当前板卡上设备数量依次创建/proc/bmsophon/cardn/bmsophon0…x 目录，其中 bmsophonx 中的 x 对应板卡 n 下设备 id 为 x 的设备信息，例如机器上只插了一张 SC5+，安装驱动后对应生成/proc/bmsophon/card0，card0 目录下会生成 bmsophon0/1/2 目录，分别存放设备 0/1/2 的信息。

SOC 模式只有 JPU 和 VPU 支持 proc 文件系统接口，节点分别是/proc/jpuinfo 和/proc/vpuinfo。

5.2.2 各项参数的含义

SOC 模式只有/proc/jpuinfo 和/proc/vpuinfo; PCIe 模式 proc 文件系统中目录和文件节点安排如下：

```
bitmain@weiqiao-MS-7B46:~/work/bm168x$ ls /proc/bmsophon/ -l

total 0

-r--r--r--.
1 root root 0 5月 6 23:06 card_num

-r--r--r--.
1 root root 0 5月 6 23:06 chip_num

-r--r--r--.
1 root root 0 5月 6 23:06 driver_version

dr-xr-xr-x 2 root root 0 5月 6 13:46 card0 //文件夹下面有板卡0的信息，如下：

bitmain@weiqiao-MS-7B46:~/work/bm168x$ ls /proc/bmsophon/card0/ -l

total 0

-r--r--r--.
1 root root 0 5月 6 23:06 board_power

-r--r--r--.
1 root root 0 5月 6 23:06 board_temp

-r--r--r--.
1 root root 0 5月 6 23:06 board_type

-r--r--r--.
1 root root 0 5月 6 23:06 board_version

-r--r--r--.
1 root root 0 5月 6 23:06 bom_version

-r--r--r--.
1 root root 0 5月 6 23:06 chipid

-r--r--r--.
1 root root 0 5月 6 23:06 chip_num_on_card

-rw-r--r--.
1 root root 0 5月 6 23:06 fan_speed
```

(续下页)

(接上页)

```
-r--r--r--.
1 root root 0 5月 6 23:06 maxboardp

-r--r--r--.
1 root root 0 5月 6 23:06 mode

-r--r--r--.
1 root root 0 5月 6 23:06 pcb_version

-r--r--r--.
1 root root 0 5月 6 23:06 sn

-r--r--r--.
1 root root 0 5月 6 23:06 tpu_maxclk

-r--r--r--.
1 root root 0 5月 6 23:06 tpu_minclk

-r--r--r--.
1 root root 0 5月 6 23:06 versions

dr-xr-xr-x.
2 root root 0 5月 6 23:06 bmsophon0//文件夹下有设备0的信息，如下：

bitmain@weiqiao-MS-7B46:~/work/bm168x$ ls /proc/bmsophon/card0/bmsophon0 -l

total 0

-r--r--r--.
1 root root 0 5月 6 23:11 a53_enable

-r--r--r--.
1 root root 0 5月 6 23:11 arm9_cache

-r--r--r--.
1 root root 0 5月 6 23:11 bmcpu_status

-r--r--r--.
1 root root 0 5月 6 23:11 boot_loader_version

-r--r--r--.
1 root root 0 5月 6 23:11 boot_mode

-r--r--r--.
1 root root 0 5月 6 23:11 cdma_in_counter

-r--r--r--.
1 root root 0 5月 6 23:11 cdma_in_time

-r--r--r--.
1 root root 0 5月 6 23:11 cdma_out_counter
```

(续下页)

(接上页)

```
-r--r--r--.
1 root root 0 5月  6 23:11 cdma_out_time

-r--r--r--.
1 root root 0 5月  6 23:11 chip_temp

-r--r--r--.
1 root root 0 5月  6 23:11 clk

-r--r--r--.
1 root root 0 5月  6 23:11 completed_api_counter

-r--r--r--.
1 root root 0 5月  6 23:11 dbdf

-r--r--r--.
1 root root 0 5月  6 23:11 ddr_capacity

-rw-r--r--.
1 root root 0 5月  6 23:11 dumpreg

-rw-r--r--.
1 root root 0 5月  6 23:11 dynfreq

-r--r--r--.
1 root root 0 5月  6 23:11 ecc

-r--r--r--.
1 root root 0 5月  6 23:11 heap

-rw-r--r--.
1 root root 0 5月  6 23:11 jpu

-r--r--r--.
1 root root 0 5月  6 23:11 location

-r--r--r--.
1 root root 0 5月  6 23:11 mcu_version

-rw-r--r--.
1 root root 0 5月  6 23:11 media

-r--r--r--.
1 root root 0 5月  6 23:11 pcie_cap_speed

-r--r--r--.
1 root root 0 5月  6 23:11 pcie_cap_width

-r--r--r--.
1 root root 0 5月  6 23:11 pcie_link_speed
```

(续下页)

(接上页)

```
-r--r--r--.
1 root root 0 5月 6 23:11 pcie_link_width

-r--r--r--.
1 root root 0 5月 6 23:11 pcie_region

-r--r--r--.
1 root root 0 5月 6 23:11 pmu_infos

-r--r--r--.
1 root root 0 5月 6 23:11 sent_api_counter

-r--r--r--.
1 root root 0 5月 6 23:11 status

-r--r--r--.
1 root root 0 5月 6 23:11 tpu_cur

-rw-r--r--.
1 root root 0 5月 6 23:06 tpu_freq

-r--r--r--.
1 root root 0 5月 6 23:11 tpuid

-r--r--r--.
1 root root 0 5月 6 23:11 tpu_power

-r--r--r--.
1 root root 0 5月 6 23:11 firmware_info

-r--r--r--.
1 root root 0 5月 6 23:11 tpu_process_time

-rw-r--r--.
1 root root 0 5月 6 23:11 tpu_volt

-rw-r--r--.
1 root root 0 5月 6 23:11 vddc_power

-rw-r--r--.
1 root root 0 5月 6 23:11 vddphy_power
```

注：如果 PCIe 模式使用 SC5P，则 mcu_version 会创建在/proc/bmsophon/card/板卡目录下。

如果使用其他类型板卡，则 mcu_version 会创建在/proc/bmsophon/card/bmsophon/设备目录下。

5.2.3 各项参数的含义和使用方法

5.2.3.1 PCIe 模式各个设备的详细信息

- card_num

  读写属性：只读；

  含义：系统板卡数量

- chip_num

  读写属性：只读；

  含义：系统设备数量

- chip_num_on_card

  读写属性：只读；

  含义：对应板卡上设备数量

- board_power

  读写属性：只读；

  含义：板级功耗

- board_temp

  读写属性：只读；

  含义：板级温度

- chipid

  读写属性：只读；

  含义：芯片 id（0x1684x/0x1684/0x1682）

- chip_temp

  读写属性：只读

  含义：芯片温度

- dbdf

  读写属性：只读

  含义：domain:bus:dev.function

- dynfreq

  读写属性：读写

  含义：使能或者禁止动态 tpu 调频功能；0/1 有效，其他值无效

- ecc

读写属性：只读

含义：打开或者关闭 ECC 功能

- maxboardp

  读写属性：只读

  含义：最大板级功耗

- mode

  读写属性：只读

  含义：工作模式，PCIe/SOC

- pcie_cap_speed

  读写属性：只读

  含义：设备支持的 PCIe 最大速度

- pcie_cap_width

  读写属性：只读

  含义：设备支持的 PCIe 接口最大 lane 的宽度

- pcie_link_speed

  读写属性：只读

  含义：设备的 PCIe 接口速度

- pcie_link_width

  读写属性：只读

  含义：设备的 PCIe 接口 lane 宽度

- pcie_region

  读写属性：只读

  含义：设备 PCIe bar 的大小

- tpuid

  读写属性：只读

  含义：tpu 的 ID（0/1/2/3……）

- tpu_maxclk

  读写属性：只读

  含义：tpu 的最大工作频率

- tpu_minclk

读写属性：只读

含义：tpu 的最小工作频率

- tpu_freq

  读写属性：读写

  含义：tpu 的工作频率，可通过写入参数来改变频率，写入前应向 dynfreq 写入 0 来关闭动态 tpu 调频, 示例如下：

```
sudo -s
echo 0 > /proc/bmsophon/card0/bmsophon0/dynfreq
echo 750 > /proc/bmsophon/card0/bmsophon0/tpu_freq
```

- tpu_power

  读写属性：只读

  含义：tpu 的瞬时功率

- firmware_info

  读写属性：只读

  含义：firmware 的版本信息，包括 commit id 和编译时间

- sn

  读写属性：只读

  含义：板卡产品编号

- boot_loader_version

  读写属性：只读

  含义：spi flash 中的 bootloader 版本号

- board_type

  读写属性：只读

  含义：板卡类型

- driver_version

  读写属性：只读

  含义：驱动的版本号

- board_version

  读写属性：只读

  含义：板卡硬件的版本号

- mcu_version

读写属性：只读

含义：mcu 软件版本号

- versions

  读写属性：只读

  含义：板卡软硬件版本的集合

- cdma_in_time

  读写属性：只读

  含义：cdma 从 host 搬数据到板卡消耗的总时间

- cdma_in_counter

  读写属性：只读

  含义：cdma 从 host 搬数据到板卡的总次数

- cdma_out_time

  读写属性：只读

  含义：cdma 从板卡搬数据到 host 消耗的总时间

- cdma_out_counter

  读写属性：只读

  含义：cdma 从板卡搬数据到 host 的总次数

- tpu_process_time

  读写属性：只读

  含义：tpu 处理过程中消耗的时间

- completed_api_counter

  读写属性：只读

  含义：已完成 api 的次数

- send_api_counter

  读写属性：只读

  含义：已发送 api 的次数

- tpu_volt

  读写属性：读写

  含义：tpu 的电压，可通过写入参数来改变电压

- tpu_cur

读写属性：只读

含义：tpu 电流

- fan_speed

  读写属性：只读

  含义：duty 风扇调速 pwm 占空比，fan_speed 风扇实际转速

- media

  读写属性：只读

  total_mem_size :vpu 和 jpu 使用内存总大小

  used_mem_size :vpu 和 jpu 正在使用的内存

  free_mem_size : 空闲内存

  id :vpu core 的编号

  link_num : 编/解码路数

- a53_enable

  读写属性：只读

  含义：a53 使能状态

- arm9_cache

  读写属性：只读

  含义：arm9 的 cache 的使能状态

- bmcpu_status

  读写属性：只读

  含义：bmcpu 的状态

- bom_version

  读写属性：只读

  含义：bom 的版本号

- boot_mode

  读写属性：只读

  含义：启动方式

- clk

  读写属性：只读

  含义：各个模块的时钟

- ddr_capacity

读写属性：只读

含义：ddr 的容量

- dumpreg

  读写属性：读写

  含义：转存寄存器，输入 1 转存到 tpu 寄存器，输入 2 转存到 gdma 寄存器

- heap

  读写属性：只读

  含义：显示各个 heap 的大小

- location

  读写属性：只读

  含义：显示当前位于哪个设备之上

- pcb_version

  读写属性：只读

  含义：pcb 的版本号

- pmu_infos

  读写属性：只读

  含义：更详细的电流电压信息

- status

  读写属性：只读

  含义：板卡状态

- vddc_power

  读写属性：只读

  含义：vddc 功率

- vddphy_power

  读写属性：只读

  含义：vddphy 功率

5.2.3.2 SOC 模式各个设备的详细信息

SOC 模式只有 JPU 和 VPU 支持 proc 接口，对应的 proc 节点为/proc/jpuinfo 和/proc/vpuinfo。

- jpuinfo

  读写属性：只读

  JPU loadbalance：记录 JPU0-JPU1(1684x),JPU0-JPU3(1684) 编码/解码次数，JPU* 为芯片内部的 JPEG 编解码器, 取值范围：0~ 2147483647

- vpuinfo

  读写属性：只读

  id: vpu core 的编号，取值范围: 0~2(1684x), 0-4(1684)

  link_num: 编/解码路数，取值范围：0~32

5.3 Tpu 驱动 sysfs 文件系统使用说明

5.3.1 Tpu 驱动 sysfs 文件系统介绍

sysfs 文件系统接口用来获取 TPU 的利用率等信息。下表列举了 Tpu 驱动 sysfs 文件系统可以获取的设备信息以及在 PCIe 和 SOC 模式下的支持情况：

设备信息	PCIe 模式	SOC 模式
npu_usage	支持	支持
npu_usage_enable	支持	支持
npu_usage_interval	支持	支持

驱动安装后，会为每个设备创建一些属性，用于查看和修改一些参数。这些属性所在的位置：

PCIe 模式：/sys/class/bm-sophon/bm-sophon0/device

SOC 模式：/sys/class/bm-tpu/bm-tpu0/device

5.3.2 各项参数的含义

下面逐一介绍每个部分代表的含义。

- npu_usage，Tpu（npu）在一段时间内（窗口宽度）处于工作状态的百分比。
- npu_usage_enable，是否使能统计 npu 利用率，默认使能。
- npu_usage_interval，统计 npu 利用率的时间窗口宽度，单位 ms，默认 500ms。取值范围 [200,2000]。

5.3.3 Tpu 驱动 sysfs 文件系统接口的具体使用方法

使用例子如下：

更改时间窗口宽度（只能在超级用户下）：

```
root@bitmain:/sys/class/bm-sophon/bm-sophon0/device# cat npu_usage_interval
"interval": 600
root@bitmain:/sys/class/bm-sophon/bm-sophon0/device# echo 500 > npu_usage_interval
root@bitmain:/sys/class/bm-sophon/bm-sophon0/device# cat npu_usage_interval
"interval": 500
```

使能关闭对 npu 利用率的统计：

```
root@bitmain:/sys/class/bm-sophon/bm-sophon0/device# cat npu_usage_enable
"enable": 1
root@bitmain:/sys/class/bm-sophon/bm-sophon0/device# echo 0 > npu_usage_enable
root@bitmain:/sys/class/bm-sophon/bm-sophon0/device# cat npu_usage_enable
"enable": 0
root@bitmain:/sys/class/bm-sophon/bm-sophon0/device# cat npu_usage
Please, set [Usage enable] to 1
root@bitmain:/sys/class/bm-sophon/bm-sophon0/device# echo 1 > npu_usage_enable
root@bitmain:/sys/class/bm-sophon/bm-sophon0/device# cat npu_usage_enable
"enable": 1
root@bitmain:/sys/class/bm-sophon/bm-sophon0/device# cat npu_usage
"usage": 0, "avusage": 0
root@bitmain:/sys/class/bm-sophon/bm-sophon0/device#
```

查看 npu 利用率：

```
root@bitmain:/sys/class/bm-sophon/bm-sophon0/device# cat npu_usage
"usage": 0, "avusage": 0
```

Usage 表示过去一个时间窗口内的 npu 利用率。

Avusage 表示自安装驱动以来 npu 的利用率。

CHAPTER 6

使用 Docker 搭建测试环境

可使用 Docker 搭建 libsophon 的测试环境, 这里介绍环境的搭建流程和使用方法。

进行此部分操作前请参照安装 libsophon 中内容在 host 端安装 sophon-driver!

6.1 Docker 测试环境搭建

6.1.1 安装 Docker

参考 官网教程 进行安装:

```
# 卸载 docker
sudo apt-get remove docker docker.io containerd runc

# 安装依赖
sudo apt-get update
sudo apt-get install \
        ca-certificates \
        curl \
        gnupg \
        lsb-release

# 获取密钥
sudo mkdir -p /etc/apt/keyrings
curl -fsSL \
    https://download.docker.com/linux/ubuntu/gpg | \
    gpg --dearmor -o docker.gpg && \
    sudo mv -f docker.gpg /etc/apt/keyrings/
```

(续下页)

(接上页)

```
# 添加 docker 软件包
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \
  https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# 安装 docker
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

6.1.2 构建测试镜像及容器

6.1.2.1 构建镜像

使用 dockerfile 构建测试所需的 Docker 镜像, dockerfile 位于 libsophon 文件夹下。构建镜像前请检查系统时间是否准确，错误的系统时间会导致构建过程中更新 package list 时证书验证失败。

```
# 替换以下 path/to/libsophon 为实际 libsophon 路径
cd path/to/libsophon
sudo docker build --platform linux/arm64/v8 -t image_name:image_version -f libsophon_
↪dockerfile .
```

以上 image_name 和 image_version 可自定义。

运行完以上命令后, 可运行 docker images 命令核对构建结果, 命令输出应如下:

```
$ sudo docker images
REPOSITORY   TAG            IMAGE ID    CREATED        SIZE
image_name   image_version  image_id    create_time    1.74GB
```

以上内容中 image_name 与 image_version 应与构建时设置内容一致, image_id 与 create_time 应与实际情况相符合。

6.1.2.2 创建容器

在 Docker 容器中进行测试需要依赖 libsophon 动态库, 依赖可分以下两种方法建立:

1. 在 host 端安装 sophon-libsophon 和 sophon-libsophon-dev, 然后通过文件映射将动态库映射进 Docker 容器中;

```
# sophon-libsophon 和 sophon-libsophon-dev 安装在 host 端
sudo docker run \
  -td \
  --privileged=true \
  -v /opt/sophon:/opt/sophon \
  -v /etc/profile.d:/etc/profile.d \
```

(续下页)

(接上页)

```
    -v /etc/ld.so.conf.d:/etc/ld.so.conf.d \
    --name container_name image_name:image_version bash
```

2. 直接在 Docker 容器中安装 sophon-libsophon 和 sophon-libsophon-dev。

```
# 在 Docker 容器中安装 sophon-libsophon 和 sophon-libsophon-dev
sudo docker run \
    -td \
    --privileged=true \
    --name container_name image_name:image_version bash
```

通过以上两种方法建立动态依赖前请首先前请参照安装 libsophon 中内容在 host 端安装 sophon-driver, 安装 sophon-libsophon 和 sophon-libsophon-dev 参考安装 libsophon 中对应内容。

以上 image_name 和 image_version 对应创建镜像时的内容, container_name 可自定义。

6.2 测试环境生效

为确保已经构建的 Docker 环境能正常使用 libsophon, 进入 Docker 容器运行以下命令:

```
# 进入 Docker 容器
sudo docker exec -it container_name bash

# 在 Docker 容器中运行此命令以确保 libsophon 动态库能被找到
ldconfig

# 在 Docker 容器中运行此命令以确保 libsophon 工具可使用
for f in /etc/profile.d/*sophon*; do source $f; done
```

运行以上命令后可运行 bm-smi 命令以检查是否可正常使用 libsophon, 命令输出应与bm-smi 使用说明 中对应内容相符。

CHAPTER 7

带外管理接口说明

7.1 SMBUS 协议接口定义

7.1.1 命令组成

先写 1 byte 的 CMD 到 i2c slave, 再读取 n byte 数据。以下为读取 CHIP0 芯片温度的例子:

```
#先往 slave 地址为 0x60 的设备写数据 0x0
i2c write 0x60 (slave addr) 0x0 (cmd)
#再从 slave 地址为 0x60 的设备读取 1 byte 数据
i2c read 0x60 (slave addr) 0x1 (n byte)
```

7.2 SC5 系列板卡带外管理接口

7.2.1 说明

服务器厂商 BMC 控制

- SC5 系列多芯卡 Slave 地址 CHIP0 为 0x60, CHIP1 为 0x61, CHIP2 为 0x62。
- 返回 int 类型数据时候按照高字节在前顺序发送 (例如 int 类型数为 0x16841e30, 返回顺序为 0x16, 0x84, 0x1e, 0x30)。

7.2.2 SC5+ MCU 接口命令

含义	地址	属性	说明
芯片温度	0x00	RO	unsigned byte, 单位: 摄氏度
单板温度	0x01	RO	unsigned byte, 单位: 摄氏度
单板功耗	0x02	RO	unsigned byte, 单位: 瓦
风扇速度占比	0x03	RO	unsigned byte, 0xff 表示无风扇
Vendor ID	0x10	RO	unsigned int;[31:16]:Device ID 0x1684;[15:0]:Vendor ID 0x1e30;
硬件版本	0x14	RO	unsigned byte
固件版本	0x18	RO	unsigned int;[7:0] 小版本号;[15:8] 主版本号;[31:16]chip 版本号
板卡种类	0x1c	RO	unsigned byte(代表板卡种类,sc5+ 是 7)
Sub Vendor ID	0x20	RO	unsigned int;[15:0]:sub Vendor ID 0x0;[31:16]:sub system ID 0x0

备注:

- 单板功耗只能从 0x60 的 0x02 地址读取
- unsigned int 表示该地址有效数据 4 个字节大小
- unsigned byte 表示该地址有效数据 1 个字节大小

7.3 SC7 系列板卡带外管理接口

7.3.1 说明

服务器厂商 BMC 控制

- SC7 系列多芯卡 Slave 地址为 0x60, CHIP1 为 0x61, CHIP2 为 0x62, 依此类推, CHIP7 为 0x67。
- 返回 int 类型数据时候按照高字节在前顺序发送 (例如 int 类型数为 0x16861f1c, 返回顺序为 0x16, 0x86, 0x1f, 0x1c)。

7.3.2 SC7PRO MCU 接口命令

含义	地址	属性	说明
芯片温度	0x00	RO	unsigned byte, 单位: 摄氏度
单板温度	0x01	RO	unsigned byte, 单位: 摄氏度
单板功耗	0x02	RO	unsigned byte, 单位: 瓦
风扇速度占比	0x03	RO	unsigned byte, 0xff 表示无风扇
Vendor ID	0x10	RO	unsigned int;[31:16]:Device ID 0x1686;[15:0]:Vendor ID 0x1f1c;
硬件版本	0x14	RO	unsigned byte
固件版本	0x18	RO	unsigned int;[7:0] 小版本号;[15:8] 主版本号;[31:16]chip 版本号
板卡种类	0x1c	RO	unsigned byte(代表板卡种类,sc7pro 是 0x21)
Sub Vendor ID	0x20	RO	unsigned int;[15:0]:sub Vendor ID 0x0;[31:16]:sub system ID 0x0
SN ID	0x24	RO	产品序列号
MCU Version	0x36	RO	unsigned byte; MCU 版本号:0

备注:

- 单板功耗只能从 0x60 的 0x02 地址读取
- unsigned int 表示该地址有效数据 4 个字节大小
- unsigned byte 表示该地址有效数据 1 个字节大小