地平线
Horizon Robotics

X3J3

MU-2520-1-X3J3平台系统软件基础接口手册

V0.1.0
2021-02

免责声明

本文档信息仅用于帮助系统和软件使用人员使用地平线产品。本文档信息未以明示或暗示方式授权他人基于本文档信息设计或制造任何集成电路。

本文档中的信息如有更改，恕不另行通知。尽管本文档已尽可能确保内容的准确性，本文档中的所有声明、信息和建议均不构成任何明示或暗示的保证、陈述或担保。

本文档中的所有信息均按“原样”提供。地平线不就其产品在任何特定用途的适销性、适用性以及不侵犯任何第三方知识产权方面做出任何明示或暗示保证、陈述或担保。地平线不承担产品使用所引起的任何责任，包括但不限于直接或间接损失赔偿。

买方和正在基于地平线产品进行开发的其他方（以下统称为“用户”）理解并同意，用户在设计产品应用时应承担独立分析、评估和判断的责任。用户应对其应用（以及用于其应用的所有地平线产品）的安全性承担全部责任，并保证符合所有适用法规、法律和其它规定的要求。地平线产品简介和产品规格中提供的“典型”参数在不同应用下可能会不同，实际性能也可能随时间而变化。所有工作参数，包括“典型”参数，都必须由用户自己针对每项用户应用进行验证。

用户同意如因用户未经授权使用地平线产品或因不遵守本说明中的条款，造成任何索赔、损害、成本、损失和（或）责任，用户将为地平线及其代表提供全额赔偿。

修订记录

时间	版本	修订说明
2019/12	V0.5	建立 X3J3 接口文档
2020/04	V0.5.1	修订部分错误
2020/08	V0.5.2	更新 Log 系统 API 说明及介绍
2021/02	V1.0	1.0 版本发布

目录

免责声明 ..... B
修订记录 I
目录 II
注意事项 III
1 概述 1
2 BPU SERVICE API 2
2.1 BPU 控制 API 数据描述 2
2.2 BPU 控制 API 2
2.3 BPU API 返回值 5
3 诊断 API 7
3.1 API 接口 7
4 LOG SERVICE API 10
4.1 LOG SERVICE 简介 10
4.2 应用程序 API 接口 11
4.3 ALOG 缓存区 11
4.4 LOGCAT 获取 LOG 日志方法 12
4.5 代码示例 12
5 系统通知 API 使用说明 14
5.1 SYS_NOTIFY_REGISTER 14
5.2 SYS_NOTIFY_UNREGISTER 14

注意事项

暂无

1 概述

相对于标准的 Linux/QNX，新开发部分主要有 VIO，BPU，显示以及诊断部分，同时移植安卓 log 接口。其他功能（网络，文件系统，BSP 等），沿用 linux POSIX 接口，在此不做赘述。图像及显示系统由单独文档介绍，本文档主要介绍 BPU，诊断，log 等系统基础接口。

文档内所有头文件位于下方路径文件夹内：

```
AppSDK\appuser\include\
```

文档内所有库文件位于下方路径文件夹内：

```
AppSDK\appuser\lib\
```

2 BPU Service API

头文件：plat_cnn.h

链接库：libcnn_intf.so

2.1 BPU 控制 API 数据描述

2.1.1 BPU Core 类型定义

```
enum cnn_core_type {
    CORE_TYPE_UNKNOWN,
    CORE_TYPE_4PE,
    CORE_TYPE_1PE,
    CORE_TYPE_2PE,
    CORE_TYPE_ANY,
    CORE_TYPE_INVALID,
};
```

数据项	描述
CORE_TYPE_UNKNOWN	未知 BPU CORE
CORE_TYPE_4PE	4PE 类型 BPU CORE
CORE_TYPE_1PE	1PE 类型 BPU CORE
CORE_TYPE_2PE	2PE 类型 BPU CORE
CORE_TYPE_ANY	不区分类型 BPU CORE
CORE_TYPE_INVALID	无效类型 BPU CORE

2.1.2 控制宏定义

```
#define BPU_POWER_OFF       0
#define BPU_POWER_ON        1
#define BPU_CLK_OFF         0
#define BPU_CLK_ON          1
#define BPU_HIGHEST_FRQ      0
```

2.2 BPU 控制 API

BPU 控制 API 提供了 BPU 开/关时钟，power on/off 以及调频功能。

2.2.1 hb_bpu_set_clk

【函数声明】

```
int32_t hb_bpu_set_clk(uint32_t core_index, uint32_t status)
```

【功能描述】

bpu 的 clock 的 on/off

【参数描述】

- [IN] uint32_t core_index：有效的 BPU Core 序号，取值范围：0 到 BPU Core Num
  - 0 - bpu0,
  - 1 - bpu1，
- [IN] uint32_t status：参考 2.1.2
  - BPU_CLK_OFF – bpu 关时钟
  - BPU_CLK_ON – bpu 开时钟

【返回值】

- 成功：0
- 失败：其它值

【兼容性】

系统版本 1.1 及以上

2.2.2 hb_bpu_set_power

【函数声明】

```
int32_t hb_bpu_set_power(uint32_t core_index, uint32_t status)
```

【功能描述】

BPU 的 power 的 on/off

【参数描述】

- [IN] uint32_t core_index：有效的 BPU Core 序号，取值范围：0 到 BPU Core Num
  - 0 - bpu0,
  - 1 - bpu1，
- [IN] uint32_t status：参考 2.1.2
  - BPU_POWER_OFF – bpu 掉电
  - BPU_POWER_ON – bpu 上电

【返回值】

- 成功：0
- 失败：其它值

【兼容性】

系统版本 1.1 及以上

2.2.3 hb_bpu_set_frq_level

【函数声明】

```
int32_t hb_bpu_set_frq_level(uint32_t core_index, int32_t level)
```

【功能描述】

设置对应 BPU Core 当前的频率级别

【参数描述】

- [IN] uint32_t core_index：有效的 BPU Core 序号，取值范围：0 到 BPU Core Num
  - 0 - bpu0,
  - 1 - bpu1,
- [IN] int32_t level：
  - BPU_HIGHEST_FRQ (0) 最高频率
  - BPU_HIGHEST_FRQ - 1 次高频率
  - BPU_HIGHEST_FRQ - N 低频率

【返回值】

- 成功：0
- 失败：其它值，参考 2.3

【注意事项】

参数“level”只能设置小于等于 0 的值，level 值越小，频率值越低，如果设置的 level 小于最低级别，则将会被设置为最低频率。可以通过下面的函数 bpu_get_total_level()获取系统提供了最多几档 level，如 bpu_get_total_level()返回值为 5，那最低频率就是 BPU_HIGHEST_FRQ - (5 -1)。

【兼容性】

系统版本 1.1 及以上

2.2.4 hb_bpu_get_total_level

【函数声明】

```
int32_t hb_bpu_get_total_level(uint32_t core_index)
```

【功能描述】

获取相应 BPU Core 支持的频率级别

【参数描述】

- [IN] uint32_t core_index：有效的 BPU Core 序号，取值范围：0 到 BPU Core Num
  - 0 - bpu0,
  - 1 - bpu1,

【返回值】

- 成功：返回当前 BPU Core 支持的频率级别
- 失败：其它值，参考 *2.3*

【兼容性】

系统版本 1.1 及以上

2.2.5 hb_bpu_get_cur_level

【函数声明】

```
int32_t hb_bpu_get_cur_level(uint32_t core_index)
```

【功能描述】

获取相应 BPU Core 当前的频率级别

【参数描述】

- [IN] uint32_t core_index：有效的 BPU Core 序号，取值范围：0 到 BPU Core Num
  - 0 - bpu0,
  - 1 - bpu1,

【返回值】

- 成功：返回当前 BPU 频率等级（<=0）
- 失败：其他值，参考 *2.3*

【兼容性】

系统版本 1.1 及以上

2.3 BPU API 返回值

控制相关返回值定义：

```
#define BPU_OK         0
#define BPU_NO_CORE    -1
#define BPU_INVAL      -2
#define BPU_NOMEM      -3
```

```
#define BPU_TIMEOUT   -4
#define BPU_NODATA    -5
#define BPU_UNKNOW    -6
#define BPU_NOGRP     -7
#define BPU_NOTSPT    -8
```

数据项	描述
BPU_OK	操作成功
BPU_NO_CORE	无对应的 BPU CORE
BPU_INVAL	无效的数据
BPU_NOMEM	内存错误
BPU_TIMEOUT	超时错误
BPU_NODATA	无数据错误
BPU_UNKNOW	未知错误
BPU_NOGRP	不存在的 group
BPU_NOTSPT	不支持

3 诊断 API

头文件：所在应用工程，需要使用诊断 API 的源文件中需包含 diag_lib.h 头文件。所在 kernel 模块需要包含<soc/hobot/diag.h>

链接库：所在应用工程需链接 libdiag_lib.a 静态库。

3.1 API 接口

3.1.1 diag_send_event_stat_and_env_data

【函数声明】

```
extern int diag_send_event_stat_and_env_data(
uint8_t msg_prio,
uint16_t module_id,
uint16_t event_id,
uint8_t event_sta,
uint8_t env_data_gen_timing,
uint8_t *env_data,
size_t env_len)
```

【功能描述】

将当前事件的状态及环境变量发送到指定缓存

【参数描述】

- [IN] uint8_t msg_prio：消息优先级，高，中，低。
- [IN] uint16_t module_id：模块 id，每个模块有一个单独的唯一 id，在 diag.h 中用枚举定义。
- [IN] uint16_t event_id：事件 id，每个模块自定义的事件 id，不做限制，一般是从 1 开始，在 diag.h 中用枚举定义。
- [IN] uint8_t event_sta：事件状态，error，ok，在 diag.h 中用枚举定义。
- [IN] uint8_t env_data_gen_timing：环境数据产生的时刻，事件错误时，事件从错误恢复时，最后一次错误时刻，在 diag.h 中用枚举定义。
- [IN] uint8_t env_data：环境数据指针。
- [IN] size_t env_len：环境数据长度。

【返回值】

- 成功：>= 0
- 失败：-1

【兼容性】

系统版本 1.1 及以上
应用工程无需 extern 字段。

3.1.2 diag_send_event_stat

【函数声明】

```
extern int diag_send_event_stat(
uint8_t msg_prio,
uint16_t module_id,
uint16_t event_id,
uint8_t event_sta)
```

【功能描述】

将当前事件的状态发送到指定缓存

【参数描述】

- [IN] uint8_t msg_prio：消息优先级，高，中，低。
- [IN] uint16_t module_id：模块 id，每个模块有一个单独的唯一 id，在 diag.h 中用枚举定义。
- [IN] uint16_t event_id：事件 id，每个模块自定义的事件 id，不做限制，一般是从 1 开始，在 diag.h 中用枚举定义。
- [IN] uint8_t event_sta：事件状态，error，ok，在 diag.h 中用枚举定义。

【返回值】

- 成功：>= 0
- 失败：-1

【兼容性】

系统版本 1.1 及以上
应用工程无需 extern 字段。

3.1.3 diag_register

【函数声明】

```
extern int diag_register(
uint16_t module_id,
uint16_t event_id,
size_t envdata_max_size,
uint32_t min_snd_ms,
uint32_t max_time_out_snd,
void (*rcvcallback)(void *p, size_t len))
```

【功能描述】

这个函数用来向诊断核心注册一些诊断消息，每个模块中的每个事件都应该注册一下：

【参数描述】

- [IN] uint16_t module_id：模块 id
- [IN] uint16_t event_id：事件 id
- [IN] size_t envdata_max_size：这个模块的这个事件，将来如果有环境数据要发送时，能够发送的最大字节数，如果后期实际发送的超过这个，则发送最大字节数。
- [IN] uint32_t min_snd_ms：最小发送间隔，多用于频繁的发送中，一般设置为 50，也即 50ms。
- [IN] uint32_t max_time_out_snd：最大超时时间，当下一次发送时刻减去上次发送时刻的值大于这个数值，则无论状态是否发生跳变，本次都要发送。主要用于连续的一直 OK，一直 ERROR 的事件时。
- [IN] rcvcallback：回调函数，如果诊断进程向这个模块的这个事件发送消息时，这个函数会被调用。目前为保留，可以设置为 NULL。

【返回值】

- 成功：>= 0
- 失败：-1

【兼容性】

系统版本 1.1 及以上
仅内核模块可用。

注意：在消息发送 API 用法上，内核层和应用层完全一样，只是在内部实现机制上不一样。内核态其他 API 函数和应用层一样。

4 Log Service API

头文件：logging.h

链接库：libalog.so

4.1 Log Service 简介

地平线日志系统封装了 Android Log 系统，以方便用户开发使用。

4.1.1 功能说明

4.1.1.1 Log 输出方式

Log 输出支持两种方式：

- Console：将 Log 打印到串口输出
- ALOG：使用 Android Log 模式输出

4.1.1.1.1 Console 模式使用方法

直接在代码中包含头文件

4.1.1.1.2 ALOG 模式使用方法

首先，在代码内包含头文件

其次，打开 ALOG_SUPPORT 宏

最后，链接 libalog.so 库。

第二，第三步可以在 Makefile 中指定，例如：

```
LOG_SUPPORT = -DALOG_SUPPORT
LOG_LIB = -L <libalog_path> -lalog    /* libalog_path 为 libalog.so 库文件的存放路径 */

CFLAGS += $(LOG_SUPPORT)
LDFLAGS += $(LOG_LIB)
```

4.1.1.2 Log Level

概括而言，Log Level 分为 4 级：

- Debug Level，输出所有信息；
- Info Level，输出 Info，Warn，Error 信息；
- Warn Level，输出 Warn， Error 信息；
- Error Level，输出 Error 信息；

根据输出模式的不同，Log Level 值宏定义如下：

```
/* output log by console */
#define CONSOLE_DEBUG_LEVEL     14
#define CONSOLE_INFO_LEVEL      13
#define CONSOLE_WARNING_LEVEL   12
#define CONSOLE_ERROR_LEVEL     11
/* output log by ALOG */
#define ALOG_DEBUG_LEVEL        4
#define ALOG_INFO_LEVEL         3
#define ALOG_WARNING_LEVEL      2
#define ALOG_ERROR_LEVEL        1
```

修改坏境变量来定义 Log Level，例如：

```
export LOGLEVEL=loglevel_value
```

当 Log Level 的值不在 1-4，11-14 之间时，默认选择 Log Level 11

4.1.1.3 模块名定义

各个模块在输出 Log 时，可以通过定义 SUBSYS_NAME 宏来打印自身模块名。通过 Makefile 传入：

```
DSUBSYS_NAME=camera
```

4.2 应用程序 API 接口

接口定义如下：

```
/* debug level */
pr_debug(fmt, ...);

/* info level */
pr_info(fmt, ...);

/* warn level */
pr_warn(fmt, ...);

/* error level */
pr_error(fmt. ...);
```

4.3 ALOG 缓存区

log 输出量巨大，特别是通信系统的 log。因此，把 log 输出到不同的缓冲区中，目前定义了四个 log 缓冲区：

1) Radio：输出通信系统的 log
2) System：输出系统组件的 log

3) Event：输出 event 模块的 log

4) Main：所有 java 层的 log，以及不属于上面 3 层的 log

缓冲区主要给系统组件使用，一般的应用不需要关心，应用的 log 都输出到 main 缓冲区中。默认 log 输出（不指定缓冲区的情况下）是输出 System 和 Main 缓冲区的 log。

4.4 logcat 获取 log 日志方法

Logcat 是一个命令行工具，可以用于得到程序的 log 信息。Log 类是一个日志类，可以在代码中使用 logcat 打印出消息。

命令格式：

```
[ adb ] logcat [ <option> ] … [ <filter-spec> ]
```

PC 端使用 adb：

```
adb logcat
```

Shell 模式下使用：logcat，例如：

```
logcat -f test.txt
```

参数	描述
-b <buffer>	加载一个可使用的日志缓冲区供查看，比如 event 和 radio。默认值是 main
-c	清除缓冲区中的全部日志并退出（清除完后可以使用-g 查看缓冲区）
-d	将缓冲区的 log 转存到屏幕中然后退出
-f <filename>	将 log 输出到指定的文件中<文件名>.默认为标准输出（stdout）
-g	打印日志缓冲区的大小并退出
-n <count>	设置日志的最大数目<count>，默认值是 4，需要和-r 选项一起使用
-r <kbytes>	每<kbytes>时输出日志，默认值是 16，需要和-f 选项一起使用
-s	设置过滤器
-v <format>	设置输出格式的日志消息。默认是短暂的格式。支持的格式列表

4.5 代码示例

```
#include <stdio.h>
#include <stdlib.h>
#include <logging.h>
```

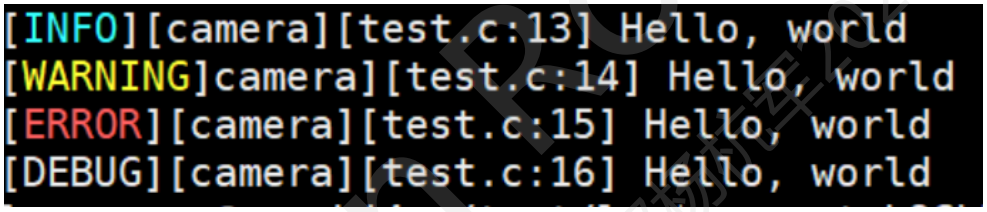
```
int main(void)
{
    /* Test Logging */
    pr_info("Hello, world\n");
    pr_warn(("Hello, world\n");
    pr_err("Hello, world\n");
    pr_debug("Hello, world\n");

    return 0;
}
```

使用相应 Makefile 编译后，将程序拷贝到板端，并在板端执行如下命令：

```
chmod +x ./test
export LOGLEVEL=14
./test
```

下方打印应出现：



执行：

```
export LOGLEVEL=12
./test
```

下方打印应出现：

5 系统通知 API 使用说明

头文件：sys_notify.h

链接库：libsys_notify.so

5.1 sys_notify_register

【函数声明】

```
int sys_notify_register(int module_id, int type_id, notify_cb cb)
```

【参数描述】

- [IN] int module_id：模块 id
- [IN] int type_id：事件 id
- [IN] notify_cb cb：事件发生时执行的回调函数

【返回值】

- 成功：0
- 失败：<0

【功能描述】

注册通知事件监听
回调函数：void (*notify_cb)(void *data，int len)

【兼容性】

系统版本 1.1 及以上

5.2 sys_notify_unregister

【函数声明】

```
void sys_notify_unregister(int module_id, int type)
```

【参数描述】

- [IN] int module_id：模块 id
- [IN] int type：事件类型

【返回值】

无

【功能描述】

注销通知事件监听

目前定义的 module id 如下

```
enum module_id {
    ModuleDiagr = 1,
    Module_I2C,
    Module_VIO,
    Module_BPU,
    Module_SOUND,
    Module_BIF,
    Module_ETH,
    ModuleIdMax = 1000,
};
```

对于通知事件类型，各模块有不同定义，需查看相关模块定义

【兼容性】

系统版本 1.1 及以上