X3J3

MU-2520-1-X3J3 Platform

System Software Base API Manual

Rev. 1.0

2021-02

Horizon Robotics, Inc.

https://www.horizon.ai

Revision History

Time	Version	Details
2020.06	V0.5	Document Created
2020.08	V0.5.2	Updated BPU service API Updated Horizon logging system API; Corrected typos
2021-02	V1.0	Release 1.0

Contents

Cautions

None.

1 Overview

Comparing to standard Linux/QNX, there are several unique modules developed: VIO, BPU, diagnose, and migrated android log interface. The majority of this manual will focus on the unique modules mentioned. Other generic interfaces are compatible with Linux POSIX interface.

All header files mentioned in the following contents is located under the path:

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

All library files mentioned in the following contents is located under the path:

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

2 BPU Service API

Header: plat_cnn.h

Library: libcnn_intf.so

2.1 BPU Control API Data Structures

2.1.1 BPU Core Type Definition

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

Item	Description
CORE_TYPE_UNKNOWN	Unknown BPU CORE
CORE_TYPE_4PE	4PE BPU CORE Type
CORE_TYPE_1PE	1PE BPU CORE Type
CORE_TYPE_2PE	2PE BPU CORE Type
CORE_TYPE_ANY	Any BPU CORE
CORE_TYPE_INVALID	Invalid BPU CORE Type

2.1.2 Control Macro Definition

```
#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 Control API

BPU Control API provides BPU clock on/off, power on/off and frequency modulation

2.2.1 hb_bpu_set_clk

【Syntax】

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

【Function】

Turns BPU clock on/off

【Parameters】

- [IN] uint32_t core_index: Valid BPU Core index, value range: [0, BPU Core Num]
  - 0 - bpu0,
  - 1 - bpu1,
- [IN] uint32_t status: Refer to *2.1.2*
  - BPU_CLK_OFF - bpu clock off
  - BPU_CLK_ON - bpu clock on

【Return Value】

- Success: 0
- Fail: Others, please refer to *2.3*

【Compatibility】

System Ver1.1 and above

2.2.2 hb_bpu_set_power

【Syntax】

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

【Function】

Turns BPU power on/off

【Parameters】

- [IN] uint32_t core_index: Valid BPU Core index, value range: [0, BPU Core Num]
  - 0 - bpu0,
  - 1 - bpu1,
- [IN] uint32_t status: Refer to *2.1.2*
  - BPU_POWER_OFF - bpu power off
  - BPU_POWER_ON - bpu power on

【Return Value】

- Success: 0
- Fail: Others

【Compatibility】

System Ver1.1 and above

2.2.3 hb_bpu_set_frq_level

【Syntax】

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

【Function】

Set BPU frequency

【Parameters】

- [IN] uint32_t core_index: Valid BPU Core index, value range: [0, BPU Core Num]
  - 0 - bpu0,
  - 1 - bpu1,
- [IN] int32_t level:
  - <BPU_HIGHEST_FRQ(0)>Maximum frequency
  - <BPU_HIGHEST_FRQ - 1> Second highest frequency
  - <BPU_HIGHEST_FRQ - N> Low Frequency

【Return Value】

- Success: 0
- Fail: Others, please refer to 2.3

【Note】

Input parameter "level" must be smaller than or equal to 0. The smaller the value, the smaller the frequency. If level is smaller than the lowest level, BPU frequency is set to slowest level. Use function bpu_get_total_level() to check how many frequency levels the system supports. For example, if bpu_get_total_level() returns 5, then the smallest frequency level is BPU_HIGHEST_FRQ - (5 -1).

【Compatibility】

System Ver1.1 and above

2.2.4 hb_bpu_get_total_level

【Syntax】

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

【Function】

Get the total number of frequency levels of BPU

【Parameters】

- [IN] unsigned int core_index: Valid BPU Core index, value range: [0, BPU Core Num]
  - 0 - bpu0,
  - 1 - bpu1

【Return Value】

- Success: Total number of frequency levels of BPU
- Fail: Others, please refer to 2.3

【Compatibility】

System Ver1.1 and above

2.2.5 hb_bpu_get_cur_level

【Syntax】

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

【Function】

Get the current frequency level of BPU

【Parameters】

- [IN] uint32_t core_index: Valid BPU Core index, value range: [0, BPU Core Num]
  - 0 - bpu0,
  - 1 - bpu1

【Return Value】

- Success: Current level of BPU (<=0)
- Fail: Others, please refer to 2.3

【Compatibility】

System Ver1.1 and above

2.3 BPU API Return Value

Definition:

```
#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
```

Item	Description
BPU_OK	Operation Success
BPU_NO_CORE	No Corresponding BPU CORE
BPU_INVAL	Invalid Data
BPU_NOMEM	No Memory Error
BPU_TIMEOUT	Timeout Error
BPU_NODATA	No Data Error
BPU_UNKNOW	Unknown Error
BPU_NOGRP	Non-existing Group
BPU_NOTSPT	Not Supported Error

3 Diagnose API

Header: Files in project using diagnostic service need to include diag_lib.h. Kernel modules need to include <soc/hobot/diag.h>

Library: Project need to link libdiag_diag.a

3.1 API Definition

3.1.1 diag_send_event_stat_and_env_data

【Syntax】

```
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)
```

【Function】

send event status and it's environment data to targeted buffer

【Parameters】

- [IN] uint8_t msg_prio: message priority, high, medium, low
- [IN] uint16_t module_id: module id, unique to each module, enumerated in diag.h
- [IN] uint16_t event_id: event id, module defined event id, no restriction, usually starts with 1, enumerate in diag.h
- [IN] uint8_t event_sta: event status: [error, ok], enumerate in diag.h
- [IN] uint8_t env_data_gen_timing: the time when environment data is generated: time error occurred, time recovered from error, the last error time, enumerated in diag.h
- [IN] uint8_t *env_data: pointer to environment data
- [IN] size_t env_len:: length of environment data

【Return Value】

- Success: >= 0
- Fail: -1

【Compatibility】

System Ver1.1 and above

For Application Softwares, "extern" is not needed

3.1.2 diag_send_event_stat

【Syntax】

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

【Function】

send event status to the diag app

【Parameters】

- [IN] uint8_t msg_prio: message priority, high, medium, low
- [IN] uint16_t module_id: module id, unique to each module, enumerated in diag.h
- [IN] uint16_t event_id: event id, module defined event id, no restriction, usually starts with 1, enumerate in diag.h
- [IN] uint8_t event_sta: event status: [error, ok], enumerate in diag.h

【Return Value】

- Success: >= 0
- Fail: -1

【Compatibility】

System Ver1.1 and above

For Application Softwares, "extern" is not needed

3.1.3 diag_register

【Syntax】

```
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))
```

【Function】

Register to diagnose core with diagnose messages. Every event in every module should all be registered.

【Parameters】

- [IN] uint16_t module_id: module id
- [IN] uint16_t event_id: event id
- [IN] size_t envdata_max_size: the maximum size of environment data can be sent for the current event. If environment data requested send is larger than this size, only envdata_max_size worth of environment data will be sent.
- [IN] uint32_t min_snd_ms: Minimum send interval, used in frequent send request. Usually: 50 (Unit: ms)
- [IN] uint32_t max_time_out_snd: Maximum timeout time. If the current send time minus the previous send time is greater than this value, no matter status changed or not, current status will be sent. Used to catch continuous OK/ERROR event.
- [IN] rcvcallback: Call back function. Call this function if diagnose is sending message to the current module. Reserved for now, can be set to NULL.

【Return Value】

- Success: >= 0
- Fail: -1

【Compatibility】

System Ver1.1 and above

Only available in Kernel modules

Note: Message send API is the same in Kernel and User space. The only difference is in actual implementation. Other API are exactly the same in Kernel and User space.

4 Log Service API

Header: logging.h

Library: libalog.so

4.1 Log Service Overview

Horizon Logging system is compatible with the Android Log system for the convenience of User.

4.1.1 Function Implementation

4.1.1.1 Log Output

The Horizon Logging system supports 2 output:

- Console: Print logs to console
- ALOG: Print logs to Android Log buffers

4.1.1.1.1 Console Mode Usage

Include "logging.h" in application source directly.

4.1.1.1.2 ALOG Mode Usage

First, include "logging.h" in application source.

Next, define ALOG_SUPPORT Macro.

Lastly, link libalog.so during compilation.

The second and last step can be done in Makefile, for example:

```
LOG_SUPPORT = -DALOG_SUPPORT
LOG_LIB = -L <libalog_path> -lalog    /* libalog_path is the path to libalog.so */

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

4.1.1.2 Log Level

To summarize, Log Level is divided into 4 severity:

- Debug Level: print all messages;

- Info Level: print Info, Warn and Error messages;
- Warn Level: print Warn and Error messages;
- Error Level: print Error messages only.

Depending on the output mode, Log Level Macron is defined as below:

```
/* 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
```

Change environment variable to change Log Level, for example:

```
export LOGLEVEL=loglevel_value
```

When the defined Log Level is not within range, the default Log Level 11 is selected.

4.1.1.3 Module Name Definition

Define SUBSYS_NAME macro to include module name in log messages. For example, define module name by Makefile:

```
DSUBSYS_NAME=camera
```

4.2 Application API

Definition:

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

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

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

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

4.3 ALOG Buffers

The amount of log is enormous, especially those from communication modules. Thus we are directing logs to difference buffers. Currently we have four buffers:

1) Radio: store logs from communication module
2) System: store logs from system module
3) Event: store logs from event module
4) Main: Other logs that does not belong to previous 3 and those from Java level

Buffers are created for system, usual application does not need to concern buffers. All logs from application are stored in main buffer. Default log output (without specifying buffers) is reading logs from System and Main buffer.

4.4 Using logcat to acquire logs

Logcat is a command line tool used to acquire application logs. Log type denotes the type of log, can be passed to logcat to print messages of that type only.

Syntax:

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

On PC: Use adb interface

```
adb logcat
```

In Shell: logcat e.g.

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

Options	Description
-b < buffer>	Load a log buffer for read. E.g. event or radio. Default value: main
-c	Clear all logs in the buffer and exit. (use -g to check buffer after clear)
-d	Print logs in buffer to screen and exit
-f < filename>	Store logs to file specified by <filename>, default to: stdout
-g	Print size of log buffer and exit
-n < count>	Set the maximum number of logs to <count>, default: 4, must be used with -r
-r < kbytes>	Print log every <kbytes> default: 16, must be used with: -f

-S	Set filter
-v < format>	Set message format for log output

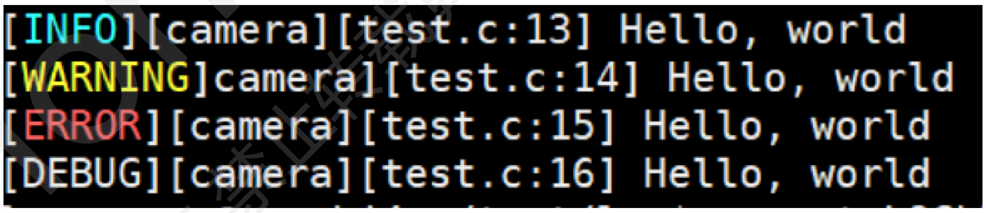
4.5 Sample Code

```
#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;
}
```

After compiling the test program with the corresponding Makefile, transfer the program to XJ3 Board. Execute the following command:

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

Output below should be printed:

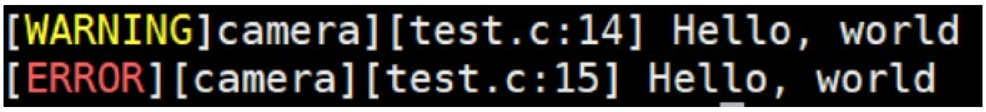


```
[INFO][camera][test.c:13] Hello, world
[WARNING]camera][test.c:14] Hello, world
[ERROR][camera][test.c:15] Hello, world
[DEBUG][camera][test.c:16] Hello, world
```

Execute the following:

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

Output below should be printed:



```
[WARNING]camera][test.c:14] Hello, world
[ERROR][camera][test.c:15] Hello, world
```

5 System Notification API

Header: sys_notify.h

Library: libsys_notify.so

5.1 sys_notify_register

【Syntax】

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

【Parameters】

- [IN] int module_id: module id
- [IN] int type_id: event id
- [IN] notify_cb cb: call back function to execute when event happens

【Return Value】

- Success: 0
- Fail: < 0

【Description】

Register event to be watched

Callback function Syntax: void (*notify_cb)(void *data, int len)

【Compatibility】

System Ver1.1 and above

5.2 sys_notify_unregister

【Syntax】

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

【Parameters】

- [IN] int module_id: module id
- [IN] int type: event type

【Return Value】

None

【Description】

Unregister the event

Module id Defined:

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

For event types, each module has its own definition. Please refer to modules for details.

【Compatibility】

System Ver1.1 and above