


status: Confidentialtop secret () secret () internal information () public (✓)		
ISP IPC module framework description and interface specification		
file status: [✓] draft [] under revision [] official release	documen t identif ication :	RK-XX-XX-XXX
	current version :	XYZ
	author:	author
	complet ion date:	2019-01-01
	review:	Reviewer
	Approva l date:	201X-XX-XX
(a product, three)		

Fuzhou Rockchip Microelectronics Co., Ltd. Fuzhou Rockchip Electronics Co., Ltd.	

Contents

1 Framework overview:	2
1.1 Framework diagram:	2
1.2 Framework description:	2
2 Interface specification:	3
2.1 Application layer interface implementation specification:	3
2.2 Interface layer specification:	4
2.3 Protocol layer specification:	4
2.4 uAPI layer specification:	5
3 Source code description:	5
3.1 Isp2-ipc source structure description:	5
3.2 CallFunIpc source structure description:	6

Disclaimer

This document is provided "as is", Fuzhou Rockchip Microelectronics Co., Ltd. ("the company", the same below) is not correct. The accuracy, reliability, completeness, marketability, specific purpose and non-infringement of any statement, information and content of this document provide any express or implied statement or guarantee. This document is only used as a reference for instructions.

Due to product version upgrades or other reasons, this document may be updated or modified from time to time without any notice.

Trademark statement

"Rockchip", "Rockchip" and "Rockchip" are registered trademarks of our company and belong to our company.

All other registered trademarks or trademarks that may be mentioned in this document are owned by their respective owners.

Copyright © 2019 Fuzhou Rockchip Microelectronics Co., Ltd.

goes beyond the scope of fair use. Without the written permission of the company, any unit or individual shall not extract or copy part or all of the content of this document without authorization, and shall not disseminate it in any form.

Fuzhou Rockchip Microelectronics Co., Ltd.

Fuzhou Rockchip Electronics Co., Ltd.

Address: Fuzhou Road, Software Park Zone A copper coil FujianNo www.rock-chips.com

URL 18:

Customer Service Tel: + 86-4007-700 -590

Customer Service Fax: +86-591-83951833

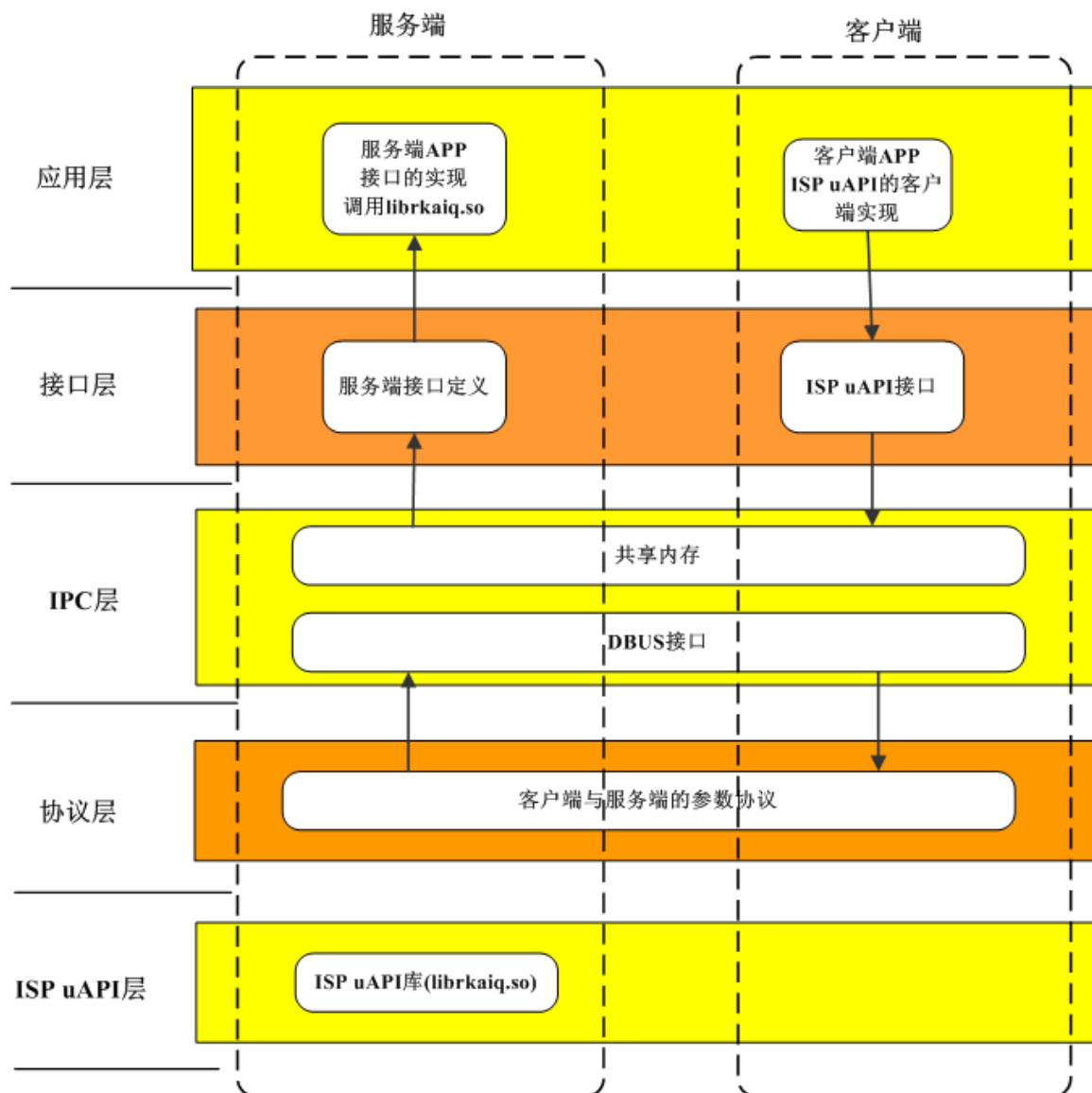
Customer Service Email: fae@rock-chips.com

Modification Record

Version Number	Author	Modification Date	Modification Description	Remarks

1 Framework Overview:

1.1 Framework Diagram:



1.2 Framework Description:

✧ Application Layer:

Server:

Implementation The interface definition of the server is mainly based on the uAPI interface definition. Add the `_ipcat` the end of each interface to interfacedirectly call the related interface of librkaiq.so. Since the interface of the server directly calls librkaiq.so , the name of the interface cannot be the same as the uAPI interface. Otherwise, there will be problems with the compilation.

Client:

The client implementation of the uAPI interface. If the client did not use the IPC mode before, if the client wantsback to the IPC mode,

to switch just change the dependent library to libispclient.so. The function interface called can not be changed. Achieve seamless switching.

✧ Interface layer:

Server:

A set of interfaces has been redefined based on the uAPI interface. On the basis of the original uAPI interface, add the "_ipc"

client:

directly adopt the uAPI interface definition.

✧ IPC layer:

✧ Protocol layer:

a structure defined based on uAPI interface parameters.

✧ ISP uAPI layer:

Librkaiq.so's open interface.

2 Interface specification:

2.1 Application layer interface implementation specification:

server side: the server side mainly implements the interface definition under the interface directory. The server calls CALL_SET_AIQ (uAPI interface name) and CALL_GET_AIQ (uAPI interface name) respectively according to the method of set and get attributes.

such as:

```
XCamReturn rk_aiq_uapi_setFocusMode_ipc(void *args){
    CALL_SET_AIQ(rk_aiq_uapi_setFocusMode);
    return 0;
}
XCamReturn rk_aiq_uapi_getFocusMode_ipc(void *args){
    CALL_GET_AIQ(rk_aiq_uapi_getFocusMode);
    return 0;
}
```

Client: The realization of uAPI's .h file interface. The server according to the method of set and get attributes

calls CLIENT_CALL_SET_AIQ (uAPI interface name) and respectively CLIENT_CALL_GET_AIQ (uAPI). If the parameter name is not attr or sys_ctx.

The

parameter name needs to be down-converted, such as:

```
XCamReturn rk_aiq_user_api_ae_setHdrAeRouteAttr(const rk_aiq_sys_ctx_t* ctx, \
    rk_aiq_sys_ctx_t* sys_ctx = ctx; const Uapi_HdrAeRouteAttr_t pHdrAeRouteAttr) {
    Uapi_HdrAeRouteAttr_t* attr = &pHdrAeRouteAttr;
    CLIENT_CALL_SET_AIQ(rk_aiq_user_api_ae_setHdrAeRouteAttr);
}
```

2.2 Interface layer specification:

server:

definition Specification: uAPI interface name+_ipc+(void *args)

void* args: pointer to shared memory structure.

such as:


```

XCamReturn
rk_aiq_user_api_a3dlut_SetAttrib_ipc(void* args);
XCamReturn
rk_aiq_user_api_a3dlut_GetAttrib_ipc(void* args);
XCamReturn
rk_aiq_user_api_a3dlut_Query3dlutInfo_ipc(void* args);

```

Client:

Directly use the uAPI interface for implementation without any modification.
such as:

```

XCamReturn
rk_aiq_user_api_ablc_SetAttrib(const rk_aiq_sys_ctx_t* sys_ctx, \
| | | | | | | | rk_aiq_blc_attr_t *attr) {
| | | | | | | | CLIENT_CALL_SET_AIQ_P(rk_aiq_user_api_ablc_SetAttrib);
| | | | | | | | }
XCamReturn
rk_aiq_user_api_ablc_GetAttrib(const rk_aiq_sys_ctx_t* sys_ctx, \
| | | | | | | | rk_aiq_blc_attr_t *attr) {
| | | | | | | | CLIENT_CALL_GET_AIQ(rk_aiq_user_api_ablc_GetAttrib);
| | | | | | | | }

```

2.3 Specification of the:

The protocol layer protocol layer is mainly defined according to the parameters of the interface. When defining the protocol, no pointers (except `rk_aiq_sys_ctx_t* sys_ctx`) can appear. If there is a return value, you need to add a return value definition.

Definition rules:

```

Typedef struct uAPI interface name {
    rk_aiq_sys_ctx_t* sys_ctx;
    parameter 2;
    XCamReturn returnvalue;
} For

```

example:

```

typedef struct rk_aiq_uapi_setMWBCT {
    rk_aiq_sys_ctx_t* sys_ctx;
    rk_aiq_wb_cct_t attr;
    XCamReturn returnvalue;
} rk_aiq_uapi_setMWBCT_t;

typedef struct rk_aiq_uapi_getMWBCT {
    rk_aiq_sys_ctx_t* sys_ctx;
    rk_aiq_wb_cct_t attr;
    XCamReturn returnvalue;
} rk_aiq_uapi_getMWBCT_t;

```

Note: Each interface defines a protocol file.

2.4 Specifications of the:

The parameter structure provided by the uAPI layer cannot be a pointer. If it is a pointer shared memory copy, there will be problems. .

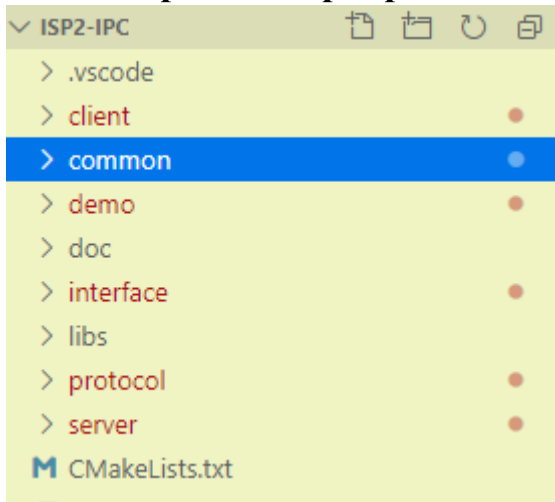
3 Source code description: The

source code is mainly composed of two parts, isp2-ipc and CallFunIpc.

Isp2-ipc is mainly the IPC implementation of rkaiq.

CallFunIpc is mainly an interface package for shared memory and DBUS.

3.1 Description of Isp2-ipc source code structure:



Client: client uAPI interface implementation, compiling will generate libispcient.so

Common: storing common files.

Demo: the demo implementation of the client, the main original libispcient.so

Doc: store related documents

interface: the interface definition of the server.

Libs: Dependent library files, such as librkaiq.so

Protocol: Store protocol files, one protocol file for each interface.

Server: The interface implementation of the server.

Source code address: <ssh://10.10.10.29:29418/linux/external/isp2-ipc>

3.2 CallFunIpc source code structure description:

3.2.1. File description:

demo: provide demo code for client and server.

call_fun_ipc.c: API for client and server call.

dbus.c: API to implement dbus communication, mainly for use by call_fun_ipc.c.

shared_memory.c: Implement inter-process memory sharing API, mainly for use by call_fun_ipc.c.

3.2.2. Process description:

Client: first call `call_fun_ipc_client_init(DBUS_NAME, DBUS_IF, DBUS_PATH, SHARE_PATH, 1)`; during initialization, then `call_fun_ipc_call((char *)__func__, ¶, sizeof(struct Examples_s), 1)`; `call_fun_ipc_client_deinit()`; must be called when the application is closed.

Server: call `call_fun_ipc_server_init(map, sizeof(map) / sizeof(struct FunMap), DBUS_NAME, DBUS_IF, DBUS_PATH, 0)`; when initializing, `call_fun_ipc_server_deinit()`; must be called when the application is closed.

DBUS: There is a dbus configuration file in the demo. The application can modify the dbus name according to its own needs. The name should be consistent with the server and client initialization.

3.2.3. API description:

`void call_fun_ipc_server_init(struct FunMap *map, int num, char *dbus_name, char *dbus_if, char *dbus_path, int needloop);` and `void call_fun_ipc_client_init(char *dbus_name, char *dbus_if, char *dbus_path, char *share_path, int needloop);`, if your application has already run `g_main_loop_run`, `needloop = 0`, if not, set 1;

3.2.4. structure description:

```
struct FunMap{
    char *fun_name; client function name.
    int (*fun)(void *); The function pointer corresponding to the server.
```

};

4. Issues to be solved in the follow-up:

- ✧ the loading interface of the third-party algorithm library needs to be redefined.
- ✧ Some algorithm library files still have pointers and need to be redefined.