



X100 SW UG FW

X100 FW Code User Guide

A03030500

RevA

2026-04-01

Release Status

Release Status	Release Status	Version control	Document Content and Instructions
Development Draft	Exploitated phase	Rev 0.10	

Version History

Edition	Author	Description
Rev 0.10	GravityXR SW	Initial Version
Rev 0.11	GravityXR SW	Add a Display section
Rev 0.2	GravityXR SW	Interface specifications for the Camera Module, Low-Power Module, and Test Case Module

Copyright Notice and Legal Disclaimer

Accessing, browsing, installing, copying or otherwise using this Software or Documentation constitutes your acceptance of the following Terms; if you do not accept these Terms, please immediately cease all access, browsing and use of this Software or Documentation.

All rights reserved © 2026, Wanyiwu (Ningbo) Electronic Technology Co., Ltd.

Copyright Statement

WanYouQuLi (Ningbo) Electronic Technology Co., Ltd. (hereinafter referred to as "WanYouQuLi" or "We") holds the copyright and other intellectual property rights to the Software Package, related firmware, drivers, HAL components, library files, accompanying documentation, source code, patches, and manuals (hereinafter referred to as the "Software Package") provided to the Customer (hereinafter referred to as the "Customer").

Without prior written permission from Universal Gravity, no entity or individual may, in any manner, copy, distribute, use, publicly disclose, publish, distribute, modify, or sell any part of the software package or its related documentation, source code, patches, or manuals—whether in the form of configuration files, images, executable files, dynamic libraries, source code, open-source libraries, or any other form—or engage in any of the following acts:

1. Extracting, decompiling, disassembling, or reverse engineering any part of a software package (unless such activities are permitted under the open-source license applicable to the open-source components);
2. Any modification, adaptation, creation of a derivative work, or other operation of the software package or any part thereof;
3. Publicly disclosing, distributing, or selling modified software packages or their derivative works;
4. Distributing a software package via any medium or method, or providing a software package to any third party.

Subject to compliance with the relevant terms of this Statement, Universal Gravity grants you a non-exclusive, non-transferable license to use this Software. The Client may use the Software solely for the purposes and within the scope agreed upon between the Client and Universal Gravity. Any use of the Software that exceeds the scope defined in this Agreement shall be deemed an infringement, and Universal Gravity reserves the right to take legal action in accordance with applicable law.

disclaimer

This statement does not constitute any express or implied warranty as to the suitability, reliability, functionality, or performance of the Product. Gravity may modify the Software Package or this Statement at any time without prior written notice. To the maximum extent permitted by law, Gravity shall not be liable for any indirect, incidental, special, or consequential damages arising from or in connection with the use or inability to use this Software, including but not limited to data loss or loss of profits. The trademarks mentioned herein belong to their respective owners (where applicable).

This Statement shall be governed by the laws of China. If any clause is deemed null, void or unenforceable, such clause shall be treated as separable and shall not affect the validity or enforceability of the remaining clauses of this Legal Statement.

About This Document

This document is intended as a reference for customers performing modifications or secondary development based on the X100 firmware source code package.

This document is intended for external customers and FAEs to review and use.

Contact Us

Address: 3rd Floor, Building 1, Pudong Software Park, No.498 Guoshoujing Road, Pudong New District, Shanghai.

For product inquiries or more information, please visit:

www.gravityXR.cn. Sales Contact Information: sales@gravityxr.cn

Catalogue

1. preface.....	1
1.1 summary.....	1
1.2 term.....	1
1.3 background.....	1
2. APP	2
2.1 Command Interaction Framework.....	2
2.2 New commands and processing logic.....	3
2.3 Add service.....	4
2.4 New calibration document.....	6
2.4.1 Add calibration document ID.....	7
2.4.2 Calibration Document Checksum/SHA256 Algorithm.....	9
2.4.3 Example function for retrieving the calibration bin.....	10
2.5 Firmware Packaging Format.....	10
2.6 Low-power scenario conversion.....	12
2.6.1 Low-power state definition.....	12
2.6.2 Low-power interface usage.....	14
2.7 Test Framework Usage.....	15
2.7.1 Framework Overview.....	16
2.7.2 Quick Start.....	16
2.7.3 Quick Reference List.....	18
3. Display	19
3.1 DP Edid Configuration.....	19
3.1.1 dp json.....	19
3.1.2 edid Modification Tool.....	19
3.1.3 Timing.....	22

3.2	Panelconfigure.....	27
3.2.1	Panel JSON Directory Structure.....	27
3.2.2	Panel JSON File Documentation.....	27
3.2.3	Panel Power-On.....	27
3.2.4	DSIoperate.....	28
3.3	algorithm WGF.....	28
3.3.1	directory structure.....	28
3.3.2	file specification.....	29
3.3.3	direction for use.....	29
3.3.4	matters need attention.....	30
3.4	Performance Tuning.....	1
3.4.1	Control Command Path.....	1
3.4.2	Screen On/Off Control.....	3
3.4.3	brightness control.....	3
3.4.4	Repositioning Control.....	4
3.4.5	Color Parameter Control.....	5
3.4.6	Color Filter Control.....	6
3.4.7	Sharpening Control.....	1
4.	Camera.....	2
4.1	Camera Sensor Configuration.....	2
4.1.1	Sensor JSON Directory Structure.....	2
4.1.2	Sensor JSON File Description.....	3
4.1.3	Configure logical camera.....	4
4.1.4	Configure physical camera.....	9
4.1.5	Configure sensor module parameters.....	11
4.1.6	Sensor power configuration-related information.....	23
4.2	Sensor DRV Interface.....	26
4.3	Sensor OTP Interface.....	28

4.4 Irled Control Interface.....	30
4.5 2A Algorithm Interface.....	34
4.5.1 Statistical Format.....	34
4.5.2 Code Directory Structure.....	37
4.5.3 control flow.....	39
4.5.4 Wrapper Implementation and Parameter Overview.....	39
4.6 SLAM Exposure Center Synchronous Control.....	45
1. CV/DSP (DSP – Zhou Zheng).....	50
1.1 Various algorithm interfaces.....	50
1.2 DSP and R8 data interfaces.....	50
2. SensorecHub	51
2.1 summary.....	51
2.2 System location.....	51
2.3 IPC Interface.....	52
2.3.1 Command Dispatch Table.....	52
2.3.2 Primary Command ID.....	53
2.3.3 Error Code.....	53
2.3.4 CTRL Command Detailed Explanation.....	55
2.4 SensorHub Key APIs.....	55
2.4.1 Core API Overview.....	55
2.4.2 sensor_event_t.....	57
2.4.3 Control Command (sensor_ctl_cmd_t).....	57
2.4.4 5.6 Hardware Registration Structure.....	58
2.4.5 Driver Layer API (Drv_sensor.h).....	59
2.4.6 sensor_report_t.....	59
2.4.7 Typical invocation flow.....	60
2.5 Event Distribution Chain.....	60

2.5.1	Registration Phase.....	60
2.5.2	Event Path.....	61
2.5.3	sensor_flow data stream.....	62
2.6	IPDsubsystem.....	62
2.6.1	hardware.....	62
2.6.2	Calibration Parameters.....	63
2.6.3	Two movement modes.....	63
2.6.4	8.4 Closed-Loop Retrying.....	63
2.7	Sensor Custom Integration.....	64
2.7.1	sensor_custom_dev_t structure.....	65
2.7.2	sensor_hub_add_hw_info.....	65
2.7.3	auto-start.....	66
2.7.4	sanity_test.....	66
2.8	Temperature Control Strategy.....	67
2.8.1	Monitoring Target.....	67
2.8.2	Temperature data source.....	67
2.8.3	Temperature Setting Definition.....	68
1.	Detailed description for each gear level.....	68
	Gear 1: Silent Mode.....	68
	Gear Position 2: Adaptive Range (PID Control).....	69
	Gear 3: Maximum Power Range.....	69
	Gear 4: High-temperature anomaly.....	69
	Gear 5: Ultra-High Temperature.....	70
2.8.4	Ultra-high temperature protection implemented.....	70
2.8.5	PVT Interrupt Level Description.....	71
2.8.6	Hysteresis Control Mechanism.....	71
2.8.7	Hysteresis interval.....	71
2.8.8	Fan Control.....	72
1.	Fan support.....	72
2.	Fan Control Interface.....	72
3.	Fan control example.....	72

4. Fan speed setting.....	73
2.8.9 PIDcontrol algorithm.....	73
1. PIDparameter.....	73
2. PIDcontrol logic.....	74
2.8.10 Temperature Protection Mechanism.....	74
1. Gear 4-High-temperature anomaly.....	74
2. Gear 5 – Ultra-High Temperature.....	74
2.8.11 Event Reporting.....	75
1. Temperature Event Notification.....	75
2. Event Definition.....	75
2.8.12 Configuration Interface.....	75
1. Temperature Control Configuration Parameters.....	75
2. Configuration Parameters Explanation.....	77
3. Drivers	77
3.1 I2C/SPI/UART/GPIO Driver.....	77
3.1.1 UART.....	77
1. data type.....	78
2. API Interface.....	79
3. matters need attention.....	83
3.1.2 GPIO.....	84
1. Pin Numbering.....	84
2. macro definition.....	84
3. API Interface.....	85
4. matters need attention.....	88
3.1.3 IOMUX.....	89
1. Pin Numbering.....	89
2. data type.....	89
3. API Interface.....	91
4. matters need attention.....	93
3.1.4 DMA.....	94

1. summary.....	94
3.1.5 I2C.....	96
1. data type.....	96
2. I2C interface.....	98
3.1.6 SPI.....	103
1. data type.....	103
2. SPI Interface.....	105
3.2 USB drive (@Li Li).....	110
3.3 Flash drive interface, File system interface.....	110
3.4 CSS DVFS Interface.....	112
3.4.1 summary.....	112
3.4.2 direction for use.....	113
3.4.3 Data Types and API.....	113
file system interface.....	115
4. DFTtool.....	119
4.1 Cardiac Emergency Kit (@Wen Yikai @Xu Qunqi @Wang Chunlong).....	119
4.2 SystemView (@Han Dahan).....	119
4.2.1 usage mode.....	119
4.2.2 Implemented features.....	122
4.2.3 Known constraints.....	122
4.2.4 reference material.....	122

1. Preface

1.1 Summary

This article introduces the usage of X100 FW, which is designed for the X100 hardware platform. The main content covers instructions for using and developing the open-source code for the App, Display, Camera, Sensor, and Cv subsystems.

1.2 Term

The terms used in this article are as follows:

Term	Explain
XR	Extended Reality (XR) encompasses VR, AR, and MR.
GXR	Gravity XR
HAL	Hardware Abstraction Layer (HAL)

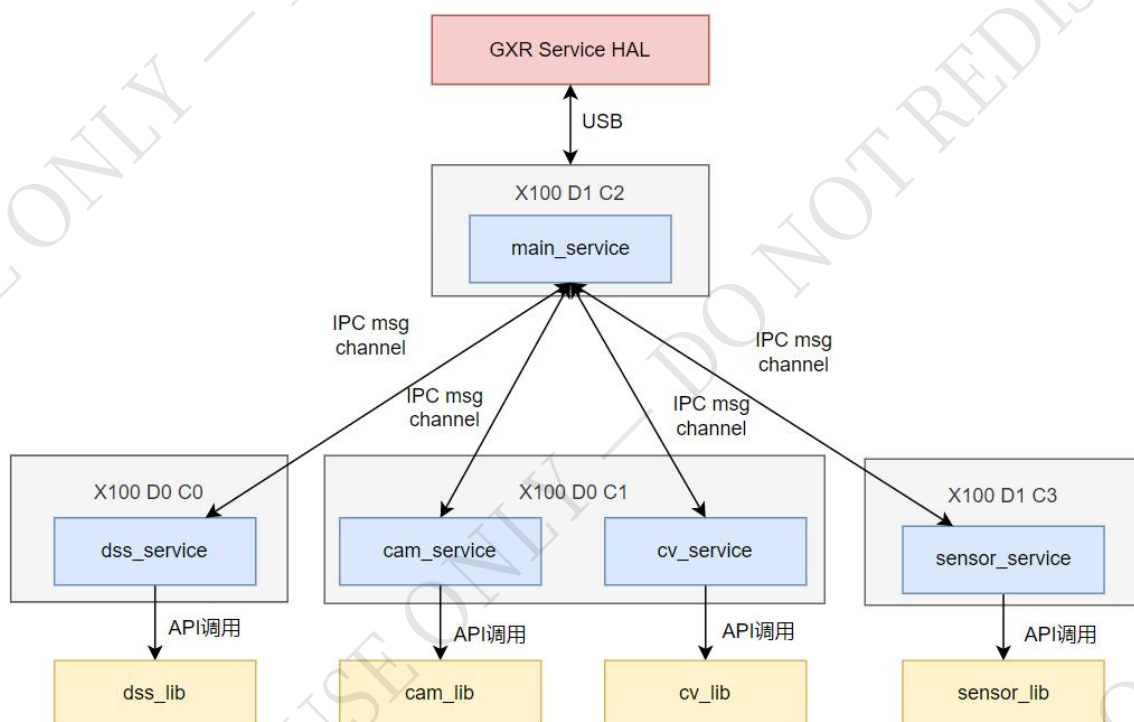
1.3 Background

This article introduces matters related to the secondary development of the X100 project's FW open-source code. The App section covers topics such as adding new commands for interacting with the GXR HAL service, introducing new services in the X100 D1 C2 firmware, and implementing message communication between these new services and the Main service; the Display and Camera subsystems section includes work such as porting and adapting custom-developed algorithms.

2. APP

2.1 Command Interaction Framework

The X100 FW command interaction framework, as well as the command interaction framework between X100FW and the GXR Service HAL, are shown below.



X100 FW Command Interaction Framework

As can be seen from the diagram, the X100 FW comprises five primary services: **main_service**, **dss_service**, **cam_service**, **cv_service**, and **sensor_service**. These services are located in four different clusters within the X100: C0 (D0), C1 (D0), C2 (D1), and C3 (D1). The **main_service** serves as the message processing hub; it communicates with the GXR service HAL over a USB interface and with the other subsystem services via IPC message channels. The four subsystem services do not communicate directly with each other via messages; instead, each subsystem service directly calls the corresponding subsystem's API to perform subsystem control.

2.2 New Commands and Processing Logic

All commands for the X100 are defined in the header file `jxapi`; the code path is `platform/include/jxapi`. The command fields are defined in the file `jx\_api\_ipc\_msg.h`, as shown below:

```
/**
 * @struct jx_ipc_msg_t
 * @brief Structure of IPC message or command, unified message structure define among all subsystem and AP host/PC tools
 */
typedef struct {
    uint32_t length;           /* msg payload length, low 16bit valid for JX firmware */
    uint16_t sub_package_idx;
    uint16_t src_dest_addr;    /* reuse for checksum check for UART transfer only */
    uint8_t flag;              /* bit mask, see define API_MSG_FLAG_xx */
    uint8_t reserved_for_magicnum;
    uint8_t reserved;
    uint8_t frame_type;        /* jx_frame_type_t */
    uint16_t command_id;       /* defined according to global type define */
    uint16_t global_type;      /* jx_ipc_global_type_t */
#ifdef WIN32
    uint8_t data[0];
#endif
} __BYTE_PACKED__ jx_ipc_msg_t;
```

When adding new commands or implementing new processing logic, focus primarily on the `command\_id` and `global\_type` fields. If you need to add a new set of commands, you must define a new entry in `global\_type`; see below for an example:

```
/* command group definaton */
typedef enum {
    IPC_GLB_NONE           = 0x00,
    IPC_GLB_REPLY          = 0x01,
    IPC_GLB_FW              = 0x02,          /* Firmware management: ready/download/check */
    IPC_GLB_LOG            = 0x03,
    IPC_GLB_SYS            = 0x04,          /* system control, mainly between JX and AP HAL */
    IPC_GLB_PM_CRG         = 0x10,          /* power management, and clock & reset generation, not used temporary */
    IPC_GLB_SOC            = 0x11,

    IPC_GLB_TEST           = 0x30,
    IPC_GLB_DBG            = 0x31,          /* R&D internal use, not open */

    IPC_GLB_SCENARIO       = 0x50,          /* scenario and feature control, cmd original from XR framework */

    IPC_GLB_CAM            = 0x70,
    IPC_GLB_CAM_DBG        = 0x71,          /* internal use for Camera subsys debug and tuning */
    IPC_GLB_DSS            = 0x80,
    IPC_GLB_DSS_DBG        = 0x81,          /* internal use for Display subsys debug and tuning */
    IPC_GLB_CV             = 0x90,
    IPC_GLB_CV_DBG         = 0x91,
    IPC_GLB_SENSOR         = 0xA0,
    IPC_GLB_SENSOR_DBG     = 0xA1,
    /*** INTERNAL USE START ***/
    IPC_GLB_INTERNAL       = 0xF0,
    /*** INTERNAL USE END ***/
    // IPC_GLB_MAX,
} jx_ipc_global_type_t;
```

Then, add the corresponding command definitions to the file `jx\_api\_ipc\_msg.h`; the specific field definitions for each command should be defined in their respective header files—for example, the command definitions related to "display" are defined in `jx\_api\_dss\_msg.h`.

To handle newly added commands, a new processing function must be added to the `mgr.c` file of the corresponding service, and then this function should be included in the processing table. For example, when adding processing commands related to the camera subsystem, a new message processing function must be created in `cam\_mgr.c`, and then a function call to invoke this new function should be added to `s\_cam\_cmd\_cb`.

```
static service_cmd_cb_t s_cam_cmd_cb[] = {
    {IPC_GLB_CAM,          IPC_CAM_OPEN,          cam_service_open},
    {IPC_GLB_CAM,          IPC_CAM_INIT,          cam_service_init},
    {IPC_GLB_CAM,          IPC_CAM_CONTROL_REQ,    cam_service_control},
    {IPC_GLB_CAM,          IPC_CAM_START,          cam_service_start},
    {IPC_GLB_CAM,          IPC_CAM_STOP,          cam_service_stop},
    {IPC_GLB_CAM,          IPC_CAM_DEINIT,         cam_service_deinit},
    {IPC_GLB_CAM,          IPC_CAM_CLOSE,         cam_service_close},

    {IPC_GLB_CAM,          IPC_CAM_CAPABILITY_REQ, cam_service_get_capability},
    {IPC_GLB_SYS,          IPC_SYS_STATE_REQ,      cam_service_get_state},
    {IPC_GLB_SYS,          IPC_SYS_VERSION_REQ,    cam_service_get_version},

    {IPC_GLB_CAM_DBG,      IPC_CAM_DBG_START_REQ, cam_service_debug_start},
    {IPC_GLB_CAM_DBG,      IPC_CAM_DBG_STOP_REQ,  cam_service_debug_stop},
    {IPC_GLB_CAM_DBG,      IPC_CAM_DBG_CONTROL_REQ, cam_service_debug_control},

    {IPC_GLB_LOG,          IPC_LOG_LEVEL_SET,      cmd_log_level_set},
    {IPC_GLB_INTERNAL,     IPC_INTERNAL_HEART_BEAT_REQ, cam_service_heart_beat_req},
};
```

2.3 Add Service

The four R8 clusters within X100 can have additional services (corresponding to one RTT thread) added as needed; specifically, the steps involved in creating and starting an RTT thread—such as its creation and initialization—can be encapsulated into an `service\_probe` interface and invoked within the existing service probe process. For example, if you wish to add an additional service (denoted as `custom\_main\_service`) that runs in parallel with `main\_service` on an X100 D1 C2 node, you can invoke the `custom\_main\_service` probe interface within the `main\_service\_probe` process.

```
jx_void_t main_service_probe(jx_void_t)
{
    jx_err_t ret = 0;

    app_fw_version();

    flow_mgr_cfg_table_init();

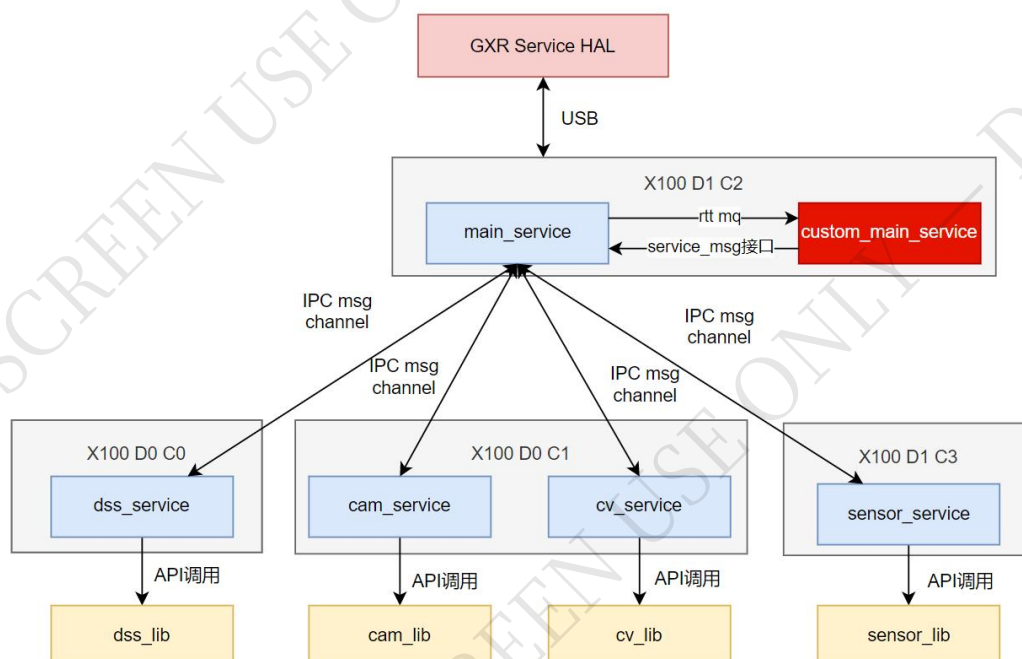
    jx_mailbox_send((jx_u32_t)MAILBOX_INDEX_FOR_FC_CFG_INFO_TABLE, FLOW_CFG_ATTR_TABLE_START_ADDR);
    jx_mailbox_send((jx_u32_t)MAILBOX_INDEX_FOR_APPFW_CFG_BIN_ADDR, APPFW_CFG_BIN_START_ADDR);
    jx_mailbox_send((jx_u32_t)MAILBOX_INDEX_FOR_SOC_CFG_BIN_ADDR, SOC_CFG_BIN_START_ADDR);
    jx_mailbox_send((jx_u32_t)MAILBOX_INDEX_FOR_TOP_CFG_BIN_ADDR, TOP_CFG_BIN_START_ADDR);

    jx_sem_para_t sem_para = {
        .name = "msg_proc_init",
        .value = 0,
        .flag = JX_IPC_FLAG_FIFO,
    };
    s_main_msg_init_done = jx_sem_create(&sem_para);
    if (JX_NULL == s_main_msg_init_done) {
        APP_LOGE("msg proc init done sem create failed!");
        return;
    }

    /* *****FSM: NULL ***** */
    /* 1. init state machine, state:null */
    ret = scenario_fsm_init();
    if (JX_OK != ret) {
        APP_LOGE("fsm init failed!");
        return;
    }
}
```

To Address the Requirement for Communication Between `custom\_main\_service` and Existing Services, Given the Cross-Cluster Nature of the Setup.

Message exchange requires using the IPC msg channel (as shown in the command interaction framework diagram); since this component has not yet been made open-source, it must be forwarded via the `main\_service`, as illustrated in the figure below.



As shown in the figure, to send a message to `main\_service` via the newly added `custom\_main\_service`, you can call the `service\_msg` interface:

```
jx_void_t service_msg_process_cmd(jx_u8_t *buf, jx_u32_t size);
```

This interface is defined in the header file `service\_msg.h`; it is important to note that this interface currently only supports sending messages to `main\_service`; other services are not yet supported.

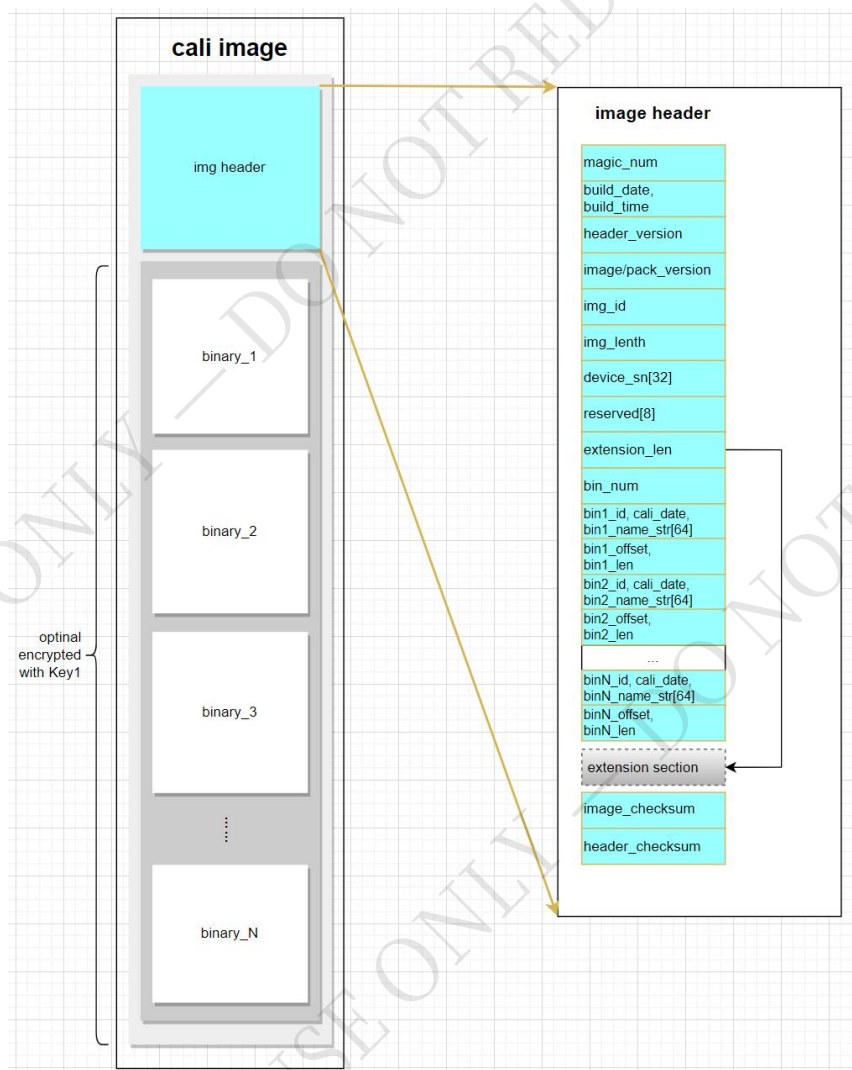
The `main\_service` can send a message to `custom\_main\_service` via the `rtt` interface of `custom\_main\_service`.

Message Sending Implemented Via a Message Queue.

2.4 New Calibration Document

All calibration files on the device are aggregated using the Device Tool, then converted into an image and written to the Flash memory; this image is then provided to the respective modules for use during scene initialization.

The format of the calibrated image is as follows: the calibrated image is created by stitching together all individual calibration bins and appending an image header to its header section:



2.4.1 Add Calibration Document ID

If You Need to Add a User-Defined Calibration File, First Add a New User-Defined Calibration Bin File ID to the ``jx_bin_id_t`` Enum Type.


```
161 /* JX_IMG_CALI_BIN, header define using v2 version */
162 JX_BIN_OM_CALI = (JX_IMG_CALI_BIN | 0x01), /* not used */
163 JX_BIN_DSS1_CALI = (JX_IMG_CALI_BIN | 0x02), /* not used */
164 JX_BIN_DSS2_CALI = (JX_IMG_CALI_BIN | 0x03), /* not used */
165 JX_BIN_CAM_CALI = (JX_IMG_CALI_BIN | 0x04), /* not used */
166 JX_BIN_SENSORHUB_CALI = (JX_IMG_CALI_BIN | 0x05), /* not used */
167 JX_BIN_CV_CALI = (JX_IMG_CALI_BIN | 0x06), /* not used */
168
169 JX_BIN_RST_CALI_CYCLE = (JX_IMG_CALI_BIN | 0x10), /* 周期 */
170 JX_BIN_RST_CALI_AOB = (JX_IMG_CALI_BIN | 0x11), /* 倾斜角 */
171 JX_BIN_RST_CALI_PRODUCT = (JX_IMG_CALI_BIN | 0x12), /* 模组生产工序 */
172
173 JX_BIN_DISP_CALI_PRC_L = (JX_IMG_CALI_BIN | 0x20), /* PRC */
174 JX_BIN_DISP_CALI_PRC_R = (JX_IMG_CALI_BIN | 0x21),
175 JX_BIN_DISP_CALI_WPC_L = (JX_IMG_CALI_BIN | 0x22), /* WPC */
176 JX_BIN_DISP_CALI_WPC_R = (JX_IMG_CALI_BIN | 0x23),
177 JX_BIN_DISP_CALI_CLUT_L = (JX_IMG_CALI_BIN | 0x24), /* CLUT */
178 JX_BIN_DISP_CALI_CLUT_R = (JX_IMG_CALI_BIN | 0x25),
179 JX_BIN_DISP_CALI_PUC_L = (JX_IMG_CALI_BIN | 0x26), /* PUC */
180 JX_BIN_DISP_CALI_PUC_R = (JX_IMG_CALI_BIN | 0x27),
181 JX_BIN_DISP_CALI_2DBL_L = (JX_IMG_CALI_BIN | 0x28), /* 2DBL */
182 JX_BIN_DISP_CALI_2DBL_R = (JX_IMG_CALI_BIN | 0x29),
183
```

The content and format of the calibration bin can be customized by the user; unless otherwise specified, the naming convention for calibration files is as follows: {descriptstr}_{bin_id}_{device_sn}_{cali_date_time}_{checksum_SHA256}, where each component is separated by an underscore (_); spaces or other separators are not supported.

- descriptstr: Description of the calibration file content; the descript string for each calibration file is fixed; any modifications must be synchronized to the tool side.
- bin_id: A uniformly defined 32-bit binary ID used for distinguishing between different firmware versions; refer to the jx_bin_id_t enum type for details on how this ID is defined.
- device_sn: The ASCII value of the device SN number (maximum 32 characters). This value must match the folder name; otherwise, the packaging script will raise an error and remove that file.
- cali_date: Definition of the calibration date; a 32-bit integer.
- Checksum/SHA256: The checksum value for this binary file, computed using SHA 256 (with the result encoded in Base64); the top 4 characters of this SHA256 value are extracted and converted into ASCII numerical values. The packaging script will verify this; if the checksum computed for the actual binary file does not match the checksum specified in the file name, an error will be raised, and the file will be removed.
- bin/json: The suffix for calibration files must be either `bin` or `json`; other formats are not supported.

After generating the calibration file, use the Device Tool provided by GravityXR to package and generate the calibration image (refer to the tool's user manual). The calibration image is bound to the device's SN number; during the firmware boot phase, an SN number

comparison is performed. Only if this comparison succeeds is the file considered a valid calibration file (this is the recommended approach; however, customers may choose not to bind the SN number to the calibration image and thus skip this mandatory check).

2.4.2 Calibration Document Checksum/SHA256 Algorithm

The checksum/SHA256 calculation method for the file name is as follows; using a Python script as an example:

```
# 检查文件名中的sha256值是否正确
sha256_result = calculate_file_sha256_base64(file['filepath'])
if sha256_result is None:
    file['caculate_sha256'] = 0
else:
    file['caculate_sha256'] = sha256_result[:4]
```

```
def calculate_file_sha256_base64(file_path):
    """
    Generate a checksum following the same process as the calibration service.

    Args:
        file_path (str): Path to the input file (text or binary)

    Returns:
        tuple: (base64_data, checksum, decoded_content)
    """
    # Step 1: Read file as binary to preserve exact content
    # This prevents line ending normalization and encoding issues
    with open(file_path, 'rb') as f:
        file_content = f.read()

    # Step 2: Encode to base64 (this is what gets sent to the API)
    base64_data = base64.b64encode(file_content).decode('utf-8')

    # Step 3: Generate checksum exactly like the service does
    # This is the exact same line as in routes.py line 175
    checksum = hashlib.sha256(base64_data.encode('utf-8')).hexdigest()

    # Step 4: Verify we can decode back to original content
    # decoded_content = base64.b64decode(base64_data)

    return checksum
```

2.4.3 Example Function for Retrieving the Calibrated Bin

The function for retrieving calibrated bin data based on the calibration bin ID from the calibrated image can be implemented using the following code. This function returns the starting address and length of the specified calibrated bin in the Flash memory, which are then passed to the designated algorithm.

```
void appfw_get_als_cali(sensor_custom_dev_t *custom)
{
    jx_bin_load_info_t *info = service_get_config_bin(bin_id: (jx_bin_id_t)JX_IMG_CALI_BIN);
    if (JX_NULL == info) {
        APP_LOGE("get cali bin info failed!");
        return;
    }
    jx_cali_img_hdr_t *cali_hdr = (jx_cali_img_hdr_t *)info->load_base_addr;
    jx_cali_img_hdr_bin_info_t *bin_info = (jx_cali_img_hdr_bin_info_t *) (info->load_base_addr + sizeof(jx_cali_img_hdr_t));
    jx_u32_t img_hdr_len = sizeof(jx_cali_img_hdr_t) + (cali_hdr->bin_num * sizeof(jx_cali_img_hdr_bin_info_t)) +
        cali_hdr->extension_len + sizeof(jx_img_checksum_t);
    als_cali_bin_t *als_cali = NULL;
    for (jx_u32_t i = 0; i < cali_hdr->bin_num; i++) {
        if (bin_info->bin_id == JX_BIN_SENHUB_CALI_ALS) {
            als_cali = (als_cali_bin_t *) (info->load_base_addr + img_hdr_len + bin_info->bin_offset);
            APP_LOGD("found als cali bin, offset=%u, len=%u", bin_info->bin_offset, bin_info->bin_len);
            break;
        }
        bin_info++;
    }

    if (JX_NULL == als_cali) {
        APP_LOGE("als cali bin not found!");
        return;
    }

    APP_LOGD("als cali header: type=%3s, fmt_version=%3s, data_version=%3s, date=%8s",
        als_cali->header.module_type, als_cali->header.fmt_version,
        als_cali->header.data_version, als_cali->header.date);
    APP_LOGD("als cali header: vendor=%16s, sn=%20s, operator=%16s",
        als_cali->header.als_vendor, als_cali->header.optical_sn,
        als_cali->header.operator_name);
}
```

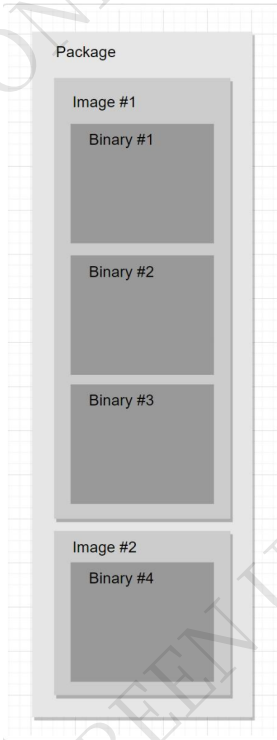
2.5 Firmware Packaging Format

The X100 firmware is released as a single package, which integrates the following components together:

1. bootloader
2. R8 & M3 elf
3. DSP elf
4. Logo Image
5. Custom configuration bin
6. AP HAL – Required differentiated configuration

The Firmware Package Format, as shown in the figure below, consists of three layers: package, image, and binary. The `package` layer represents the overall distribution package released by the JX chip; the `image` layer is primarily intended for the AP side – all AP-side operations are performed at the `image` level, and the AP does not need to concern itself with individual binaries within the `image`; the `image` is partitioned primarily based on functionality and download phase; the `binary` layer is primarily intended for the X100 firmware side – both the CPU and firmware operations on the X100 side are performed at the `image` level.

Using the following diagram as an example, this Package layer consists of two Image layers: Image #1 and Image #2; among them, the Image #1 layer is composed of three Binary layers: Binary #1, Binary #2, and Binary #3; while the Image #2 layer consists of only one Binary layer: Binary #4.



The `package Header` Is Defined as Follows:

Item	Size	Description
magic_num	u32 'P"A"C"K'	
build_date	The ASCII code for a u32 date.	For example: 2023-10-15--> 0x20231015
build_time	The ASCII code for a 24-hour time format, with leading zeros padded to the high-order bits. For example: 13:24:38 → 0x00132438. In the current internal version, when using the script-based packaging tool (package).	

Item	Size	Description
		This field is defined as the lowest 8 bits of the commit ID, making it convenient to locate specific Git commits during the development process.
header_version	u32	"00.00.00.01" --> 0x00000001
`package_version`	u32	"00.00.00.01" --> 0x00000001`. You could consider using up to 8 bits to distinguish between different customer projects, and the remaining 24 bits to define the version number.
package_len	u32	The length of the data within the package (excluding the package header length).
img_nums	u32	Number of image sub-elements within the package hierarchy
img_offset	u32	The offset of the image header within each image subdirectory in the package
img_attr_h	u32	Image subdirectory attribute field high-order bits
img_attr_l	u32	The status of attribute fields in the image subdirectory
package_checksum	u32	Checksum of data in the package
header_checksum	u32	Checksum of all data in the package header

The packaging method for all packages is defined by the JSON configuration files located in the `pkg/pack\_config` folder; customers may also extend the `image` and `binary` components according to their specific requirements, or add their own data types.

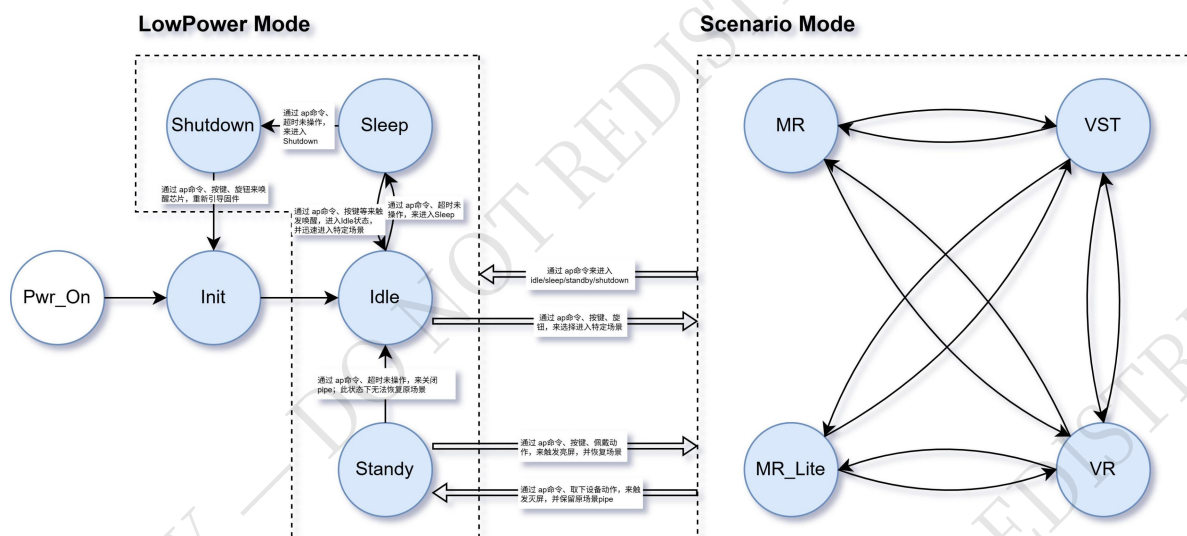
2.6 Low-Power Scenario Conversion

2.6.1 Low-Power State Definition

To illustrate the low-power scenario conversion scheme and its relationship to real-world operating scenarios, the operating modes and states of the X100 software are defined as follows:

- Scenario Mode: Normal scenario operating conditions, such as MR/VR/MR_Lite/VST;
- LowPower Mode: A non-operational state; defines Standby/Idle/Sleep/Shutdown states based on the X100 chip and software architecture.

The transition diagram between various states is shown below:



All states depicted in the above diagram are uniformly defined and managed by the software state machine; the definitions of each state are as follows:

1. **Pwr.On: Power-on Trigger**
2. **Init: System initialization phase; does not belong to the low-power mode.**
 - (1) Firmware retrieves product configuration information.
 - (2) Initialize global system resources
 - (3) Subsystem Self-Check
 - (4) Subsystem handshake – Synchronization of state information
3. **Idle :**
 - (1) Subsystem initialization is complete; the CPU enters the idle state, typically the WFI state.
 - (2) All pipelines are in reset state; backlight/display is turned off.
 - (3) Enter the working mode in Scene Mode via a command or button press.
 - (4) If the device has not entered Scene Mode, it will automatically enter the Sleep low-power state (*optional*) upon reaching the timeout condition (*adjustable time*).
4. **Standby :**
 - (1) Turn off display/背光; set CPU to idle mode; retain all pipeline configurations to support rapid wake-up.
 - (2) Disable interactive-related algorithms and DSPs, such as HT/ET, etc.
 - (3) Wake-up source (Optional: *Command/Key/Wearable*)– Returns to the original scene mode state after waking up.
 - (4) If the timeout (*adjustable*) elapses without the device being awakened, all pipelines will be reset and the system will enter the Idle low-power state.

5. Sleep :

- (1) During idle mode, the idle computing units are powered down to further reduce power consumption, while the USB connection remains active.
- (2) Wake source (Optional: *Command/Key Press/Wearable*) Upon waking, the device automatically enters the Idle state.
- (3) If the device does not wake up within the *adjustable timeout period*, it will automatically enter the Shutdown state (*optional*).

6. Shutdown :

- (1) In sleep mode, all computing units are powered off, most memory is powered off, and the USB connection is disconnected; (*Optional*) Wake-up functionality is retained, with selectable wake-up sources.
- (2) Wake-up source (Optional: *Key press/ Wearable device*) Upon waking, the software automatically enters the Init state to re-initialize the system.

2.6.2 Low-Power Interface Usage

The low-power handover procedure is implemented by the main service and is proprietary; it provides an external API interface for invoking the `jx\_api\_ipc\_mgs` command, as defined below:

■ Dormancy Request/Response Interface:

```
/**
 * @struct jx_sys_sleep_req_t
 * @brief payload of MSG_ID: IPC_SYS_SLEEP_REQ, MSG_Direction: AP Host -> JX APPFW
 */
typedef struct {
    uint32_t mode; /* jx_sleep_mode_t */
} __BYTE_PACKED__ jx_sys_sleep_req_t;

/**
 * @struct jx_sys_sleep_resp_t
 * @brief payload of MSG_ID: IPC_SYS_SLEEP_RESP, MSG_Direction: JX APPFW -> AP Host
 */
typedef struct {
    uint32_t mode; /* jx_sleep_mode_t */
    int32_t result; /* jx_err_t */
} __BYTE_PACKED__ jx_sys_sleep_resp_t;
```

The valid value range for the `jx\_sleep\_mode\_t` enum in the structure corresponds to the various low-power scenario states defined in Section 2.5.1; the enum definition is as follows:

```
typedef enum {
    JX_SLEEP_MODE_IDLE           = 0x00,
    JX_SLEEP_MODE_STANDBY       = 0x01,
    JX_SLEEP_MODE_SLEEP         = 0x02,
    JX_SLEEP_MODE_SHUTDOWN      = 0x03,
    // to be complete
} jx_sleep_mode_t;
```

For a sleep request, the sleep mode must be configured according to the state transition diagram specified in Section 2.5.1; any invalid request will be notified via the error code included in the sleep response.

■ Wake-up Request/Response Interface:

```
/**
 * @struct jx_sys_wakeup_req_t
 * @brief payload of MSG_ID: IPC_SYS_WAKEUP_REQ, MSG_Direction: AP Host -> JX APPFW
 */
typedef struct {
    uint32_t mode; /* jx_sleep_mode_t */
} __BYTE_PACKED__ jx_sys_wakeup_req_t;

/**
 * @struct jx_sys_wakeup_resp_t
 * @brief payload of MSG_ID: IPC_SYS_WAKEUP_RESP, MSG_Direction: JX APPFW -> AP Host
 */
typedef struct {
    uint32_t mode; /* jx_sleep_mode_t */
    int32_t result; /* jx_err_t */
} __BYTE_PACKED__ jx_sys_wakeup_resp_t;
```

The usage method for the Wake-up Request/Response interface is the same as that for the Sleep Request/Response interface. To facilitate the unified management of state information on both the X100 and AP sides, the AP side can use the State Get Interface (jx_sys_state_req_t/jx_sys_state_resp_t) to query the current software state of the X100.

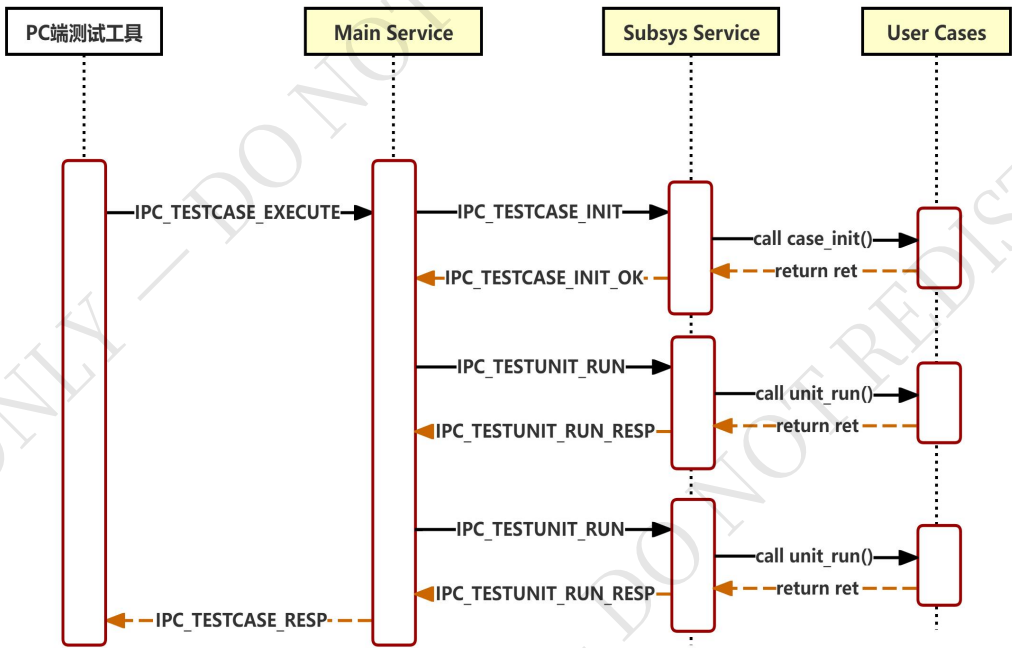
2.7 Test Framework Usage

This chapter introduces the overall structure of the Testcase framework, the message flow, and how to write and integrate a custom test case, helping developers quickly extend test cases using this framework.

- **Framework Overview:** Section 2.6.1 provides a complete timing flowchart for Case execution, indicating the key source code locations.
- **Quick Start:** Follow Section 2.6.2 to complete the development, registration, and execution of your first case in just a few minutes.
- **Quick Reference List:** Section 2.6.3 provides a comprehensive quick reference list; after development, you can use this list to check for any missing items.

2.7.1 Framework Overview

Time Series Overview:



Key source code:

Sound code file	Explain
jx_api_test_msg.h	IPC Message Structure Definition
jx_api_subsys.h	jx_subsys_id_t – Subsystem ID Enumeration
testcase_service_entry.h	Core Data Structures and API Declarations
testcase_service_entry.c	Case Registry: s_jx_testcase_map[], along with the table lookup execution logic
testcase_mgr.c	Message Routing
case_template.c	Case Example Code

2.7.2 Quick Start

The `app/services/main/misc/app\_testcase/` directory provides a ready-to-use template `case\_template.c`, which includes a four-part set consisting of `init/`, `unit\_1/`, `unit\_2/` and `cleanup/`, as well as the `unit\_map[]` registry. All constraints (e.g., function signatures, usage of `unit\_map` and `extra\_data`, `init/`/`cleanup` requirements, etc.) are documented in the comments within the C template file.

★When creating a case, please refer to and modify this template directly; this chapter only describes the provisioning steps.

As an example, consider adding a new case named `case\_my\_demo`: suppose two unit functions, `case\_my\_demo\_init` and `case\_my\_demoCleanup`, as well as `case\_my\_demo\_unit\_map[]`, have been defined.

1. Add Header File Declaration:

Open the file `app/services/main/misc/app\_testcase/app\_case\_api.h`.

Following the format of the `case\_template.c` section, add a `case` declaration:

```
/* case_my_demo.c */  
  
extern jx_test_unit_handler_t case_my_demo_unit_map[];  
jx_err_t case_my_demo_init(jx_void_t);  
jx_err_t case_my_demo_cleanup(jx_void_t);  
jx_err_t case_my_demo_unit_basic(jx_u32_t *param);  
jx_err_t case_my_demo_unit_with_data(jx_u32_t *param);  
/*-----*/
```

2. Register in the Global Table:

Open the file `app/services/main/testcase\_service/testcase\_service\_entry.c`.

In `s\_jx\_testcase\_map[]`, add the following to the `FW\_X100\_C2` macro block:

```
{(jx_u8_t *) "utest.app_tc.my_demo",  
    case_my_demo_init, case_my_demo_cleanup, case_my_demo_unit_map},
```

★If the Case code is distributed across subsystems (under the `subsys/` directory),
append it to the corresponding macro block.

3. Configuration Setup:

Create an SConscript file in the directory where the reference template is located;

Kconfig Prerequisite:

- `JX\_TESTCASE\_ENABLE = y`

If you need to place a Case into a custom directory, you can create a similar SConscript; however, if the Case code is located within a subsystem (under the `subsys/` directory), do not add the `FW\_X100\_C2` constraint;

4. Test version firmware compilation:

Simply add `-t--testcase` to the compilation command; for example:

```
./pkg/x100_build_pack.sh -p antman_pcmr -t --testcase
```

5. PC-side parameter entry: (@Chen Hao— Please help complete this section further.)

Field	Explain
subsys_id	Refer to platform/include/jxapi/jx_api_subsys.h for the values. For example, SUBSYS_APPFW = 0x35
case_name	Must exactly match the string in s_jx_testcase_map[]
unit_name	Single unit name; or enter "JX_NULL"; the framework will execute all units in sequence – if any unit fails, execution stops.
para[0]	The meaning of the two 32-bit parameters passed to the unit shall be defined by the respective case.
para[1]	The meaning of the two 32-bit parameters passed to the unit shall be defined by the respective case.

2.7.3 Quick Reference List

Document	Operate	Explain
app_my_demo_case.c	New-built	Caserealize
xxx_case_api.h	Edit/Create	If implemented under an existing Case path, simply modify the function and the 'unit_map' declaration; if implemented under a custom new path, add them.
testcase_service_entry.c	Revise	Add a registry entry to s_jx_testcase_map[];
SConscript	No change/ Add new	If implemented under the existing Case path, no modifications are required; If implemented under a custom new path, it will be added;

3. Display

3.1 DP Edid Configuration

3.1.1 dp json

Depending on the PCMR or DP screen casting mode, configure the `dp\_cfg.json` file; this configuration file is primarily used to set up the X100Dp capabilities.

```
"single_edid":{
  "Property": 0,
  "Struct": "jx_dp_edid"      zdzhang, last month • [Jx10x55-222] feat(appfw): add dp custom config_
},
"len":{
  "Type": "uint32_t",
  "Value": 128
},
"data":{
  "Type": "uint8_t",
  "Column": 256,
  "Value":[
    "0x00", "0xFF", "0xFF", "0xFF", "0xFF", "0xFF", "0xFF", "0x63", "0x18", "0x00", "0x00", "0x00", "0x00", "0x00", "0x00",
    "0x00", "0x00", "0x01", "0x04", "0x00", "0xC0", "0x6C", "0x78", "0x1A", "0x00", "0xC9", "0xA0", "0x57", "0x47", "0x98", "0x27",
    "0x12", "0x48", "0x4C", "0x00", "0x00", "0x00", "0x01", "0x01", "0x01", "0x01", "0x01", "0x01", "0x01", "0x01", "0x01",
    "0x01", "0x01", "0x01", "0x01", "0x01", "0x01", "0x03", "0x57", "0x80", "0x18", "0x71", "0x38", "0x20", "0x40", "0x58", "0x2C",
    "0x45", "0x00", "0xC4", "0x8E", "0x21", "0x00", "0x00", "0x1E", "0x02", "0x3A", "0x80", "0x18", "0x71", "0x38", "0x20", "0x40",
    "0x58", "0x2C", "0x45", "0x00", "0xC4", "0x8E", "0x21", "0x00", "0x00", "0x1E", "0x00", "0x00", "0x00", "0xFC", "0x00", "0x67",
    "0x72", "0x61", "0x76", "0x69", "0x74", "0x79", "0x0A", "0x20", "0x20", "0x20", "0x20", "0x20", "0x20", "0x20", "0x20", "0x20",
    "0x00", "0x17", "0xF1", "0x08", "0x8C", "0x1E", "0x00", "0x0A", "0x20", "0x20", "0x20", "0x20", "0x20", "0x20", "0x00", "0xEC"
  ]
},
}
```

The EDID parameter is written to the X100; subsequently, the host device uses DP to recognize the X100's capabilities and determine information such as resolution and frame rate. The parameters configured for JSON must include the following two parameters: one is the EDID array. If only the Base block is used, the array size is 128 bytes; if the ETX block is used, the configured array should be 256 bytes – the first 128 bytes represent the Base block, and the remaining 128 bytes represent the Ext block; together, these two blocks constitute a single array.

1. EDID Array

Enter either 128 bytes or 256 bytes (use EXT block) when filling the array;

2. EDID Length

If Base block is used, fill in 128; if EXT block is used, fill in 256;

3.1.2 Edid Modification Tool

There are numerous EDID editing tools available online; you can also generate your own EDID using custom tools. Below is a tool for generating EDID data using DP logic analysis – provided for reference only.

1. Download path: <https://www.quantumdata.com/downloads.html>
2. Download and Install

teledynetec.com/protocolanalyzer/solutions-video

HDMI and DisplayPort Analyzers HDMI Test Instruments HDMI Analog Video Generator/Legacy Products

manufacturing facilities.

HDMI and DisplayPort Protocol Analyzers/Generators

The industry's leading video test platforms provide an unmatched combination of functional and compliance testing for next-gen HDMI and DisplayPort systems.

96G HDMI Video Analyzer/Generator
quantumdata M42zh

Video generation with diagnostic tools
Deep analysis for fast time-to-insight
Comprehensive protocol compliance testing for sources and sinks
API for automated testing

48G HDMI Video Analyzer/Generator
quantumdata M41h

Video generation with diagnostic tools
Deep analysis for fast time-to-insight
Comprehensive protocol compliance testing for sources and sinks
API for automated testing

80G DisplayPort Video Analyzer/Generator
quantumdata M42de

Video generation & analysis
Supports HBR3 & UHBR rates
Comprehensive compliance testing
API for automated testing

teledynetec.com/protocolanalyzer/quantumdata-m42de

TELEDYNETEC
Products Technologies Industries & Applications Services Support Buy

Protocol Analyzers > DisplayPort > quantumdata M42de M42de

quantumdata M42de Video Analyzer/Generator

Overview Details Adapters Request a Quote

80G Video Analyzer/Generator

The quantumdata M42de Video Analyzer/Generator provides an unprecedented combination of functional and compliance testing for video, audio and protocol lanes for DisplayPort 2.1.

[Dataset](#) [Software](#)

DisplayPort 1.4/2.1 HBR3/UHBR
TECHNOLOGY VERSIONS LANE RATES

teledynetec.com/support/softwaredownloads/pagedocuments.aspx?standardid=26&series=678

Support Software Downloads

Tool Library
Instrument Service
FAQ/Knowledgebase
Device/Security
Software Downloads
Technical Help
Request Info
Aval & M222
Events & Training
Update Profile

Protocol Test Solutions Software

Analysis Software for DisplayPort

Download and install the Teledyne LeCroy Protocol Analysis Software from the Teledyne LeCroy website to optimize your Protocol test systems, updatable in the field. Functionality available, and customized with your company's logo. This file can be viewed and analyzed on your computer.

To get the right software for your needs, please select one of the Protocol Test Solutions from the list below. Once the product is selected, you will see the list of available software starting with the latest available. You can also search for the software by using the right-hand side search feature.

In the software list below, click [Download](#) to begin the installation process.

Notes:
The right-hand side is a short form if you are not a registered Teledyne LeCroy user.

Select Protocol Technology or search for software

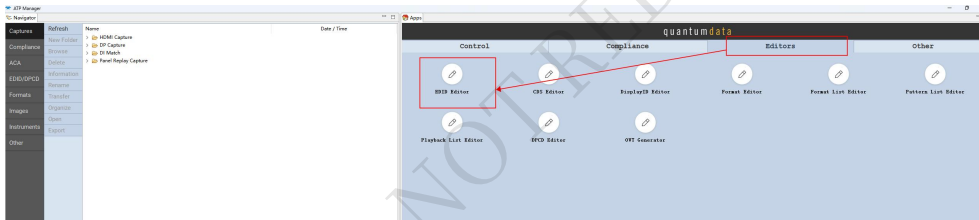
View archival software versions for DisplayPort

Software Release Version	Version	File Size	Release Date	
Instrument Firmware: M4 Series for M41/M42/M42de	7.70	103 MB	1/2/2026	Download
M41/M42/M42de 7.70 Release				
ATP Manager Graphical User Interface	7.70	132 MB	1/2/2026	Download
Application Programming Interface (API)	7.70	24.9 MB	1/2/2026	Download
Video Protocol Suite Graphical User Interface	7.70	132 MB	1/2/2026	Download

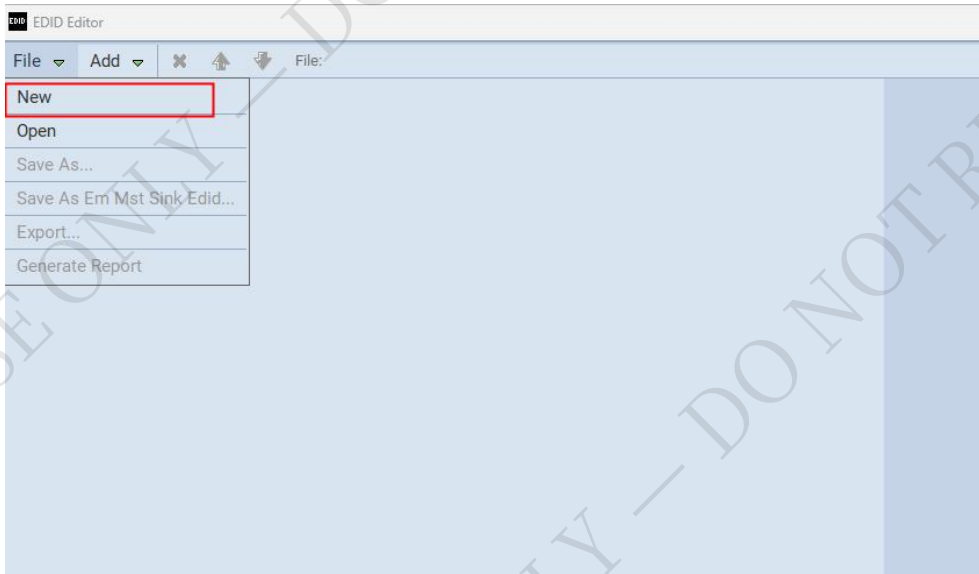
名称	修改日期	类型	大小
atp-api-r7.50.04	2025/6/16 9:26	压缩(zip)文件夹	6,957 KB
m4xx-k-5.4.31-amd64-r7.50.04	2025/6/16 12:25	DEB 文件	847,220 KB
r_980mgr_7.50.04.47064_win32	2025/6/14 12:20	Windows Installer ...	136,518 KB
videoguise7.50	2025/6/16 10:03	应用程序	135,844 KB

3. Tool Usage

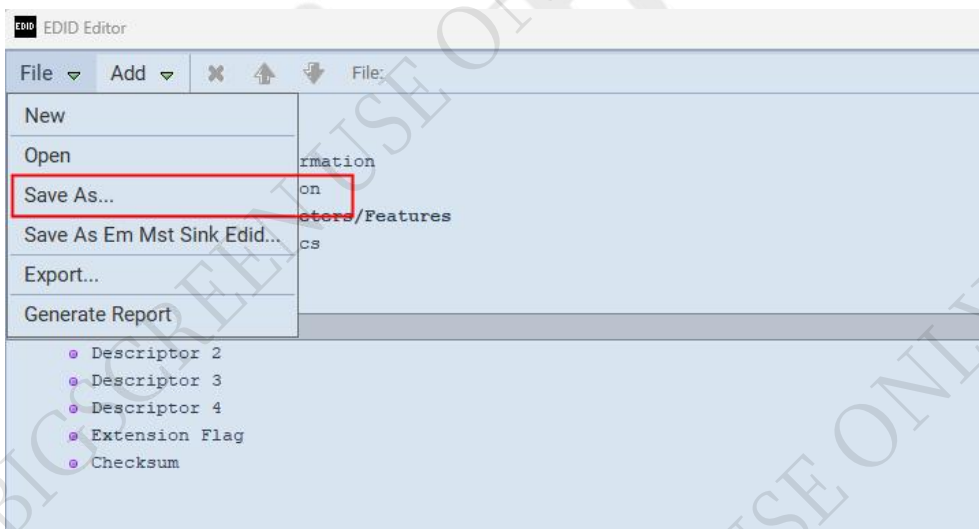
a) Open ATP MANAGER



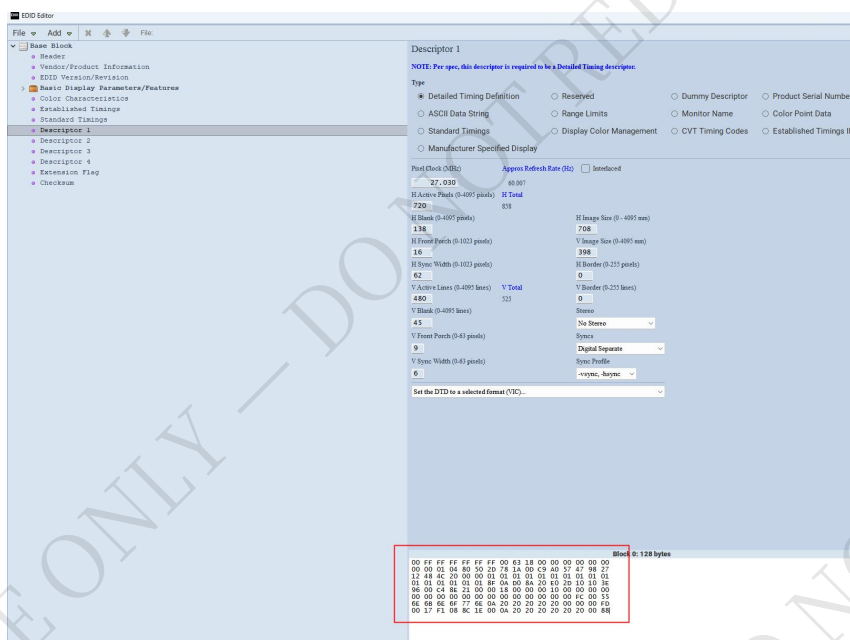
B) Create New EDID



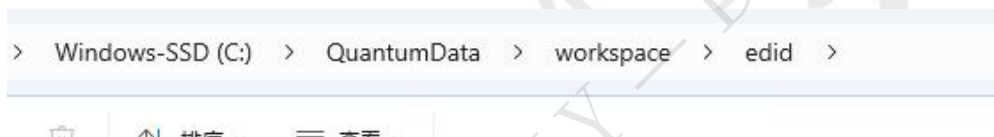
c) Save new EDID



d) EDID Data: Can Be Selected and Copied Directly Within the Tool; E) File Save Path: The



Quantum Data Directory Located on the System Drive.



3. 1. 3 Timing

If the Width or Height exceeds 4096, or if the Pixel Clock exceeds 655 MHz, the timing parameters must be configured within the CTA block; for other cases, only the basic configuration information needs to be set in the Base block – in such scenarios, the Base Block can be used directly, with additional EXT Blocks added as needed.

Base Block:

1. Header

Generally, no modification is required;

2. Vendor/Product information

Fill in as needed;

3. EDID Version/Revision

Adjust as needed; you can use the default values;

4. Basic Display Parameters/Features

1) Video Input Definition

Generally, it is Digital; select the supported Color Bit Depth;

2) Horz and Vert Screen Size

You can enter the screen size according to the specified ratio, or choose "Unknown/Undefined"; however, some TX platforms may require you to specify the screen size.

3) Display Transfer Characteristic

Modify as needed; default values can be used;

4) Feature Support

Select the supported RGB/YUV format according to your requirements;

5. Color Characteristics

Customize as needed; you may use the default values;

6. Established Timings

Check the boxes as needed; if you don't need them, just leave them unchecked.

7. Standard Timings

Check the boxes as needed; if you don't need them, just leave them unchecked;

8. Descriptor X

All custom timing parameters are configured here; Descriptor 1 represents the default "prefer timing" setting. If there is only one timing configuration, the subsequent values (2/3/4) can be set to "Reserve" or follow the specified configuration.

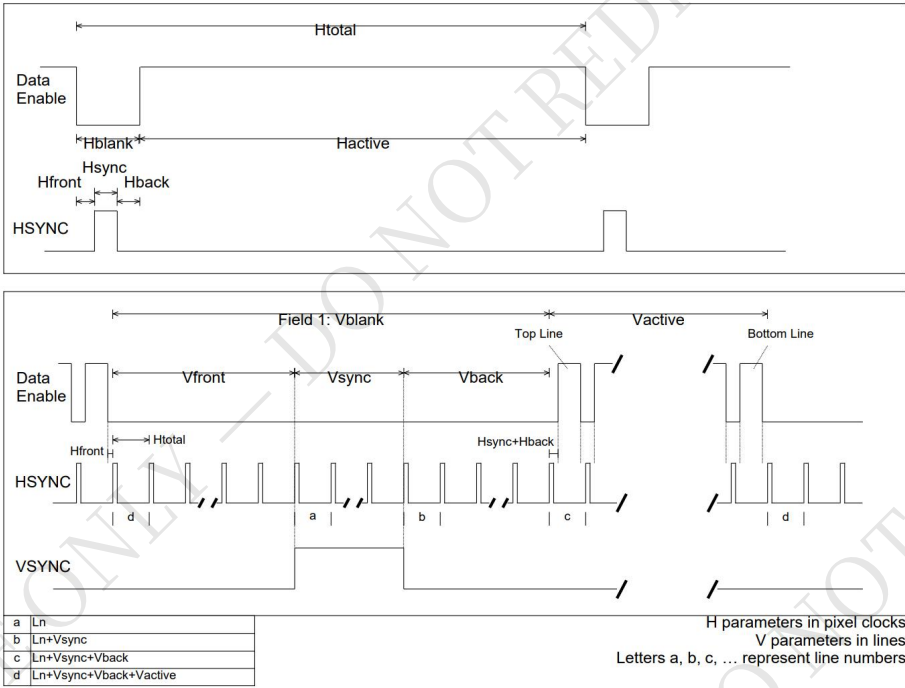
NOTE: Per spec, this descriptor is required to be a Detailed Timing descriptor.

1. 选择 Detailed Timing Definition

2. 填写 timing 参数，从上到下分别是
H-active: width pixel数
H-blank: width blank总宽度
H-front porch: sync前blank 宽度
H-sync width: sync 宽度
V方向参数同上

3. 计算帧率需要的Pixel clock, 计算方式是 $H\text{-total} \times V\text{-total} \times \text{fps}$

4. 选择 sync profile: 即 sync 的 Polarity, 需要全部选择+



9. Extension Flag

It is automatically generated based on whether an EXT Block is present; no modification is required.

10. Checksum

Automatically generated – no manual modifications required; once the timing configuration is set up, simply run the "Apply" command – it will update automatically;

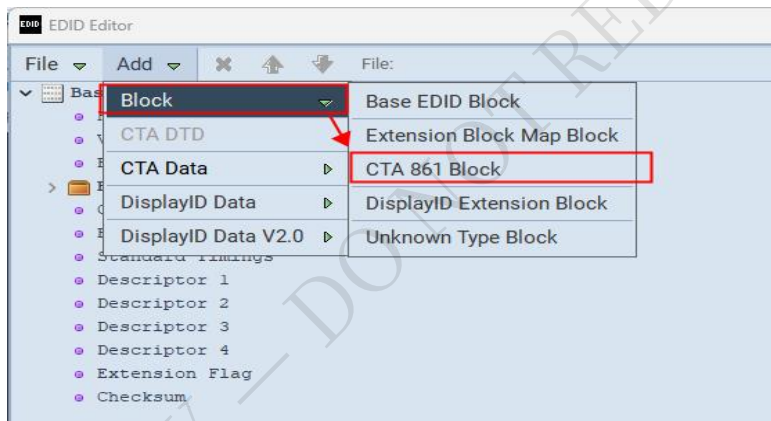
If only the Base Block is used, the EDID array configured for the system requires only 128 bytes of the Base Block;

CTA Block:

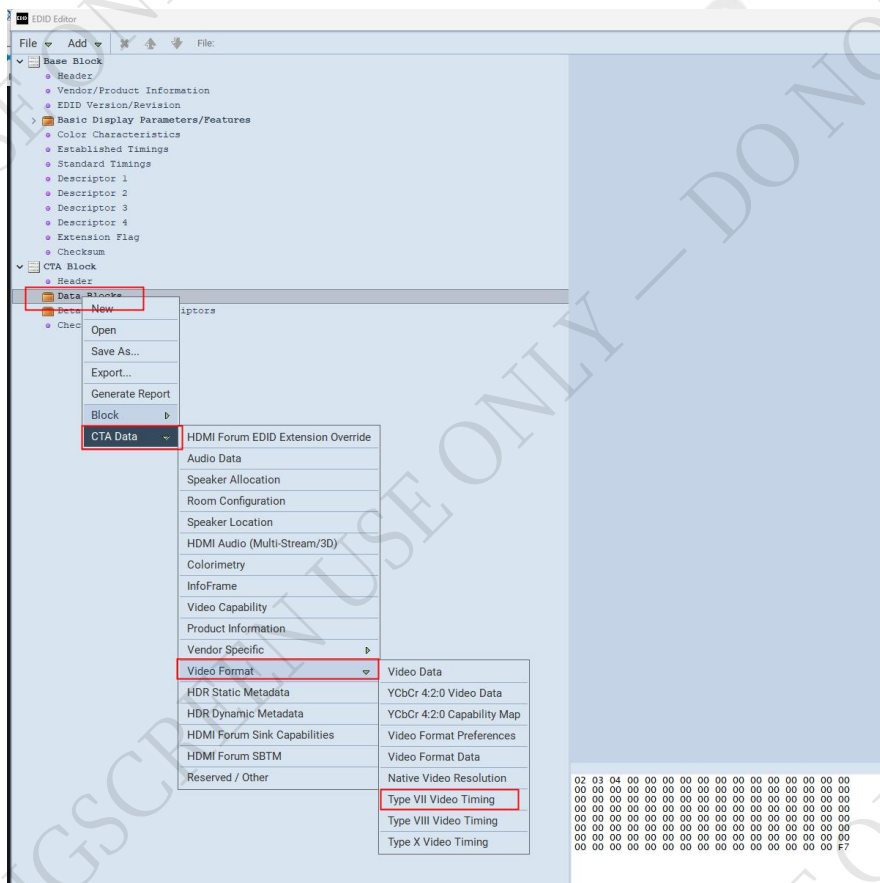
1. When any of the following conditions is met, the Base Block is no longer able to meet the requirements, and a CTA Block must be used:

- A. The width in pixels exceeds 4095;
- B. The number of height pixels exceeds 4095;
- C. Pixel Clock exceeds 655.35 MHz;
- D. Other conditions under which EXT Block is required;

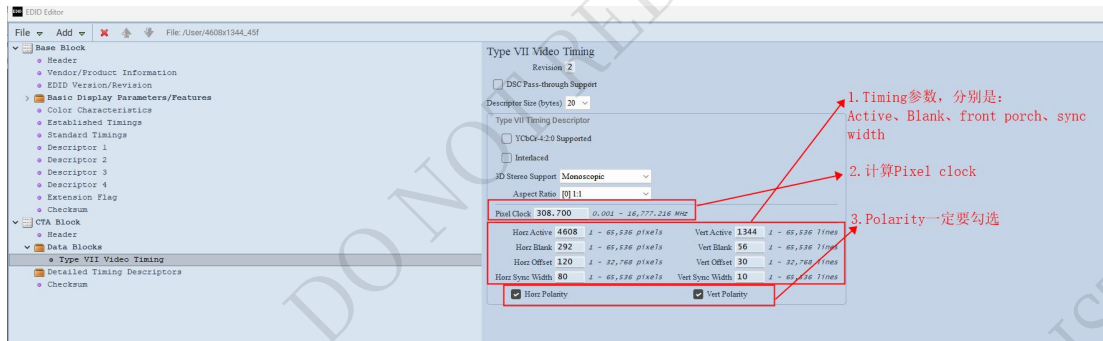
2. Create CTA Block



3. Create Timing

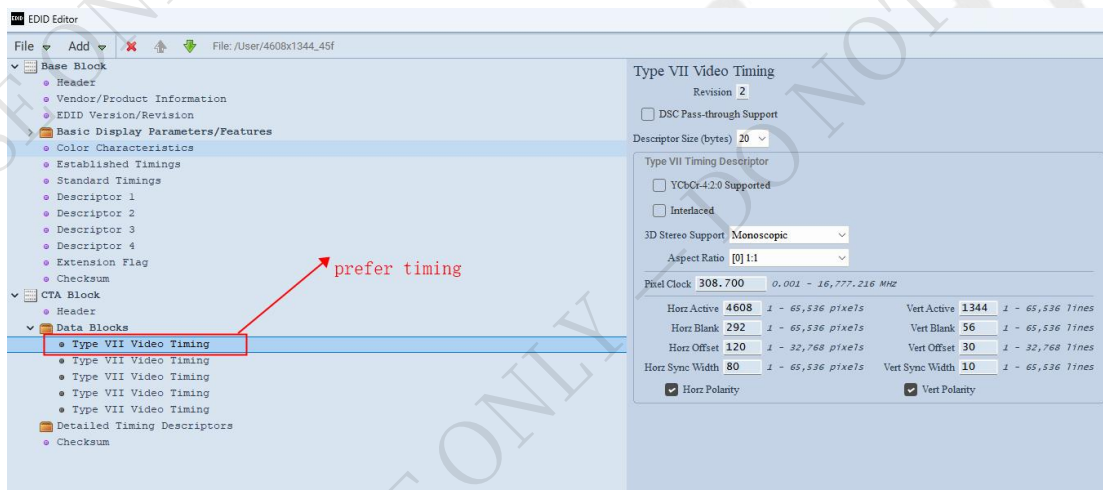


4. Specific Timing Configuration



5. If You Need to Support Multiple Timing Parameters, You Can Repeat the Creation Process as Described in Step 3; However, a Maximum of 5 Parameters Can Be Supported.

The "prefer" timing is the first:



6. When EXT Block is used, the EDID data configured for the system is 256 bytes (base 128 bytes + ext 128 bytes);

3.2 Panel Configuration

3. 2. 1 Panel JSON Directory Structure

- ├── dss1_custom_cfg.json # No modifications required
- ├── panel_cfg_0 # First screen information
 - ├── dss_dsi_tx_cfg.json #Mipi DSI core and PHY-related parameters
 - ├── dss_panel_dev_cfg.json
 - ├── panel_bl_cfg.json # Backlight parameter information
 - ├── panel_gpio_cfg.json # Power-on control timing sequence
 - ├── panel_para_cfg.json # Panel timing-related configurations
 - ├── panel_seq_cfg.json # Screen initialization code
- ├── panel_cfg_1 # Second screen information (if available)
 - ├── dss_dsi_tx_cfg.json
 - ├── dss_panel_dev_cfg.json
 - ├── panel_bl_cfg.json
 - ├── panel_gpio_cfg.json
 - ├── panel_para_cfg.json
 - ├── panel_seq_cfg.json

3. 2. 2 Panel JSON File Description

Document name	Act on
dss1_custom_cfg.json	Dependency relationship
panel_cfg_0	Screen 0 – Corresponding Parameter Settings
dss_dsi_tx_cfg.json	Mipi DSI and PHY related parameter settings
dss_panel_dev_cfg.json	Dependency relationship
panel_bl_cfg.json	Reserved backlight parameter settings
panel_gpio_cfg.json	Panel Power-On Control Parameter Settings
panel_para_cfg.json	Transmission frame timing-related porch configuration
panel_seq_cfg.json	Panel initialization parameters command; Screen on/off command

3. 2. 3 Panel Power-On

Interface	Act on
dss_panel_power_init	Power Initialization

dss_panel_power_on	panelpower on
dss1_panel_power_reset	panelresetting
dss1_panel_power_off	Panel Power Off

It drives the panel's ascending and descending operations as well as the initialization sequence based on the Panel JSON configuration.

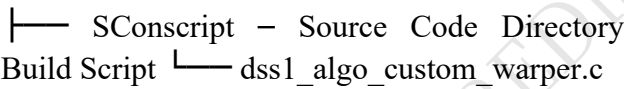
3.2.4 DSI Operation

Interface	Act on
dss1_tx_panel_timing_init	dsi init
dss1_tx_panel_init	Panel Initialization
dss1_tx_panel_timing_trigger	Trigger Mipi data transmission
dss1_tx_panel_display_on_off	Panel: Turn on/Off
dss1_tx_panel_deinit	Panel Deinitialization
dss1_tx_panel_tpg	X100 – Output TPG – Turn on the screen
dss1_tx_panel_send_cmd	Send an Mipi command to the panel.
dss1_tx_panel_esd_status_get	Detect panel status
dss1_tx_panel_emduty_set	Set Panel Duty Brightness

3.3 Algorithm WGF

3.3.1 Directory Structure

```
|—— SConscript          # Root Directory Build Script
|—— inc/                 # Public Header File Directory
|   |—— dss1_pq_algo_adapter.h
|   |—— dss1_pq_algo_api.h
|—— lib/                 # Dependency library header file directory
|   |—— dss_lib_algo_warp_grid_fuse.h
|   |—— dss_lib_common_type.h
|—— src/                 # Demo Source Code Directory
```

3.3.2 File Specification

Document name	Act on
dss1_pq_algo_adapter.h	Customized algorithm adaptation layer header file – functions that require user-defined implementation; these functions are invoked by the jx_lib side.
dss1_pq_algo_api.h	`jx_lib` – External API header file, defining the functions and structures exposed on the jx side.
dss_lib_algo_warp_grid_fuse.h	Warp Grid Fuse Algorithm Library Header File
dss_lib_common_type.h	Algorithmic generic type definition header file – defines platform-agnostic data structures, enumerations, etc.
dss1_algo_custom_warper.c	Custom Warper algorithm implementation file – Sample
SConscript	Source code directory build script – configures source file compilation and linking rules

3.3.3 Direction for Use

Customized Warp Algorithm Implementation

- The custom Warper logic is implemented in the file `src/dss1\_algo\_custom\_warper.c`; users should implement their custom algorithm within this file, and `jx\_lib` will invoke it at runtime.
- params: Points to the Warp algorithm parameters, including all algorithmic parameters related to the Warp operation; for details, refer to the documentation in dss_lib_algo_warp_grid_fuse.h;
- scene_dt: Points to runtime environment variables, including pipe ID, FRM ID, etc.;
- opt: Select the type of Warp Table to generate; two options are supported;
 - WGF_OPT_ROTATE_VDO_RT_WGF: Computes and generates a video warp table; the result is written to wgf_param->out.vdo_warp_tbl (a pre-initialized address space that cannot be modified);
 - WGF_OPT_ROTATE_GFX: Computes and generates a GFX warp table; the result is written to wgf_param->out.gfx_warp_tbl (a pre-initialized address space that cannot be modified);

3.3.4 Matters Need Attention

- Environment Dependency: Before compiling, ensure that the DSS1 Platform SDK environment, the SCons tool, and the cross-compilation toolchain are correctly configured;
- Header file dependencies: The 'lib/' directory contains dependency library header files; these files should not be modified arbitrarily and must remain consistent with the DSS platform version;
- Interface compatibility: The header files in the inc/ directory must strictly adhere to the DSS1 Platform API specification; any modifications made must be synchronized with the platform-side code.
- Build Guidelines: Do not modify the SConscript build script without authorization; if you need to adjust compilation parameters, please contact the platform maintenance team.
- For a frame rate of 90 fps, the WGF algorithm must complete its computation in less than 7 ms to prevent frame loss.

3.4 Performance Tuning

3.4.1 Control Command Path

Screen on/off control, brightness control, and repositioning control are implemented by sending a `control msg` via the APP, which is ultimately invoked through the `dss1\_control` interface. This section first introduces the control command transmission pathway to help readers understand how to organize and send `control msg` messages.

1. Dss1_control API Definition

```
jx_err_t dss1_control(jx_handle_t handler, jx_dss_cmd_id_t id, jx_dss_ctrl_param_t *param);
```

2. The Relevant `cmd\_id` and Structure Are Defined in `jx\_api\_dss\_msg.h`.

```
typedef enum {  
    /** for DSS common run cmd use */  
    DSS_CMD_COMMON_BASE = 0x00,  
    DSS_CMD_GET_VERSION = 0x01, ///< get fw version of display  
    DSS_CMD_GET_STATUS = 0x02, ///< get dss fw status  
    DSS_CMD_GET_CAPABILITY = 0x03, ///< get capability of dss  
    DSS_CMD_SET_ALS = 0x04, ///< update als for dss  
    DSS_CMD_SET_TEMPERATURE = 0x05, ///< update temperature for dss  
    DSS_CMD_GET_TEMPERATURE = 0x06, ///< get temperature for dss  
    DSS_CMD_REPORT_TEMPERATURE = 0x07, ///< report temperature of dss to AP  
  
    /** for X100 DSS1 run cmd use */  
    DSS1_CMD_BASE = 0x400,  
    DSS1_CMD_GET_BINNING = 0x401, ///< get binning of dss1 screen  
    DSS1_CMD_SET_TUNING = 0x402, ///< set tuning bin  
    DSS1_CMD_SET_TUNING_BIN = 0x403, ///< set tuning bin  
    DSS1_CMD_SET_PO_PARA = 0x404,  
    DSS1_CMD_SET_LAYER = 0x405, ///< cfg layer of display  
    DSS1_CMD_SET_BL = 0x406, ///< config backlight level  
    DSS1_CMD_SET_FPS = 0x407, ///< config frame rate of display  
    DSS1_CMD_GET_FRAME_ATTR = 0x408, ///< config frame attr, jx_frame_attr_t  
    DSS1_CMD_SET_SCENARIO_ID = 0x409, ///< config scenario id  
    DSS1_CMD_UPDATE_DP_CONFIG = 0x40A, ///< update dp config param  
  
    /**TODO: 0x410 to 0x415 not used, temporary add to avoid ap_hal compile error!!!  
    DSS1_CMD_SET_ALS = 0x410, ///< update als for dss  
    DSS1_CMD_GET_ALS = 0x411, ///< get als of dss  
    DSS1_CMD_REPORT_ALS = 0x412, ///< report als of dss to AP  
    DSS1_CMD_SET_TEMPERATURE = 0x413, ///< update temperature for dss  
    DSS1_CMD_GET_TEMPERATURE = 0x414, ///< get temperature for dss  
    DSS1_CMD_REPORT_TEMPERATURE = 0x415, ///< report temperature of dss to AP  
  
    DSS1_CMD_SET_TEMP_CLUT = 0x416, ///< config clut depend on temperature  
    DSS1_CMD_SET_FPS_CLUT = 0x417, ///< config clut depend on fps  
    DSS1_CMD_SET_TEMP_PRC_LUT = 0x418, ///< config prc lut depend on temperature  
    DSS1_CMD_SET_FPS_PRC_LUT = 0x419, ///< config prc lut depend on fps  
    DSS1_CMD_SET_TM = 0x41A,  
  
    DSS1_CMD_SEND_APP_EVENT = 0x420, ///    DSS1_CMD_GFX_EFFECT_CTRL = 0x421, ///    DSS1_CMD_LAYER_CTRL = 0x422, ///  
    DSS1_CMD_CAPTURE = 0x450, ///< start capture  
    DSS1_CMD_RECORDING_START = 0x451, ///< start record  
    DSS1_CMD_RECORDING_STOP = 0x452, ///< stop record  
    DSS1_CMD_RECORDING_PAUSE = 0x453, ///< pause record  
    DSS1_CMD_RECORDING_RESUME = 0x454, ///< resume record  
  
    DSS1_CMD_SET_DEBUG = 0x601,
```

```
typedef struct {
    jx_void_t *in para;          ///< in para: input parameters
    jx_u32_t in size;            ///< in size: length of input parameters
    jx_void_t *out para;         ///< out para: return output parameters, memory allocated by subsystem, NULL if no data
    jx_u32_t out size;           ///< out size: return length of output parameters, ZERO if no data
    jx_u8_t need free;           ///< need_free: return whether need APPFW to free memory, 1 means need, 0 means NO need
} __BYTE_PACKED__ jx_dss_ctrl_param_t;
/** INTERNAL_USE_END */

/**
 * @struct jx_dss_control_req_t
 * @brief payload of MSG_ID: IPC_DSS_CONTROL_REQ, MSG_Direction: APP main thread -> Main/2nd DPU Service
 */
typedef struct {
    uint8_t module_id;          /* jx_dss_module_id_t */
    uint8_t reserved[3];
    uint32_t cmd_id;             /* jx_dss_cmd_id_t */
    int32_t in_size;             ///< input parameter size
#ifdef WIN32
    uint8_t in_para[0];         ///< input parameter
#endif
} __BYTE_PACKED__ jx_dss_control_req_t;
```

3. For Usage Examples, Refer to App_msg_proc.c and Dss1_mgr.c.

(1) The APP sends control messages via the main service (this example pertains to the Recording configuration).

```
jx_err_t cmd_scenario_screen_recording(jx_ipc_msg_t *msg)
{
    jx_err_t ret = JX_OK;
    JX_ASSERT(msg);
    jx_screen_recording_req_t *payload = (jx_screen_recording_req_t *) (msg->data);
    if (payload) {
        jx_dss_control_req_t *dss_req = (jx_dss_control_req_t *) jx_malloc(sizeof(jx_dss_control_req_t) + sizeof(jx_dss_cmd_set_recording_t));
        if (dss_req) {
            if (0 == payload->type) {
                dss_req->cmd_id = DSS1_CMD_RECORDING_STOP;
            } else if (1 == payload->type) {
                dss_req->cmd_id = DSS1_CMD_RECORDING_START;
            } else if (2 == payload->type) {
                dss_req->cmd_id = DSS1_CMD_CAPTURE;
            }
            dss_req->module_id = MODULE_DSS_MAIN;
            dss_req->in_size = sizeof(jx_dss_cmd_set_recording_t);
            jx_dss_cmd_set_recording_t *record = (jx_dss_cmd_set_recording_t *) (dss_req->in_para);
            record->type = payload->type;
            record->pipe_sel = payload->pipeline;
            record->format = payload->format;
            record->fps_mode = payload->fps_mode;
            record->stride_align = payload->stride_align;
            if (payload->crop_width > 0 && payload->crop_height > 0) {
                record->crop_startx = payload->crop_startx;
                record->crop_starty = payload->crop_starty;
                record->crop_width = payload->crop_width;
                record->crop_height = payload->crop_height;
            } else {
                record->crop_startx = 0;
                record->crop_starty = 0;
                record->crop_width = payload->width;
                record->crop_height = payload->height;
            }
            jx_void_t *dss1_handler = service_get_handle(DSS1_SERVICE);
            jx_void_t *self = service_get_handle(MAIN_SERVICE);
            ret = service_send_msg(self, dss1_handler, IPC_GLB_DSS, IPC_DSS_CONTROL_REQ, dss_req,
                                  sizeof(jx_dss_control_req_t) + sizeof(jx_dss_cmd_set_recording_t));
            jx_free(dss_req);
        }
    }
    return ret;
}
```

2 Main M/R Messages Are Forwarded to DSS1 M/R, Where They Are Then Processed by the DSS1 Service.

() _ g _ g

```
static jx_err_t dss1_service_control(jx_ipc_msg_t *msg)
{
    jx_err_t ret = JX_OK;
    JX_ASSERT(JX_NULL != msg);
    JX_ASSERT(JX_NULL != msg->data);

    static jx_dss_control_req_t *s_info = JX_NULL;
    static jx_u32_t s_total_size = 0; control_req_t 是ipc_msg中带入的消息，对于大消息需要多包拼接

    jx_dss_control_req_t *payload = (jx_dss_control_req_t *) (msg->data);
    jx_dss_control_req_t *p = (jx_dss_control_req_t *) jx_realloc(s_info, s_total_size + msg->length);
    if (p) {
        s_info = p;
    } else {
        jx_free(s_info);
        s_info = JX_NULL;
        s_total_size = 0;
        return JX_NOMEM;
    }

    jx_u8_t *p_char = (jx_u8_t *) s_info;
    p_char += s_total_size;
    jx_memcpy(p_char, payload, msg->length);
    s_total_size += msg->length;

    if (0x0 == (msg->flag & API_MSG_FLAG_MORE_PACKAGE)) {
        APP_LOGD("dss ctrl cmd = 0x%x total size = %u msg->length = %u ", s_info->cmd_id, s_total_size, msg->length);
        s_total_size = 0;
        jx_dss_ctrl_param_t param = {0};
        param.in_para = s_info->in_para;
        param.in_size = s_info->in_size;
        ret = dss1_control(s_dss1_info.dss1_handle, s_info->cmd_id, &param);
        if (JX_OK != ret) {
            APP_LOGE("dss1 control failed!"); 最终给到dss1_control的消息是 ctrl_param_t
        }
        if (DSS_CMD_SET_ALS == s_info->cmd_id || DSS1_CMD_SET_SCENARIO_ID == s_info->cmd_id) {
            jx_free(s_info);
            s_info = JX_NULL;
            return ret;
        }
    }

    jx_dss_control_resp_t *resp = (jx_dss_control_resp_t *) jx_malloc(sizeof(jx_dss_control_resp_t) + param.out_size);
    if (resp) {
        jx_memset(resp, 0, sizeof(jx_dss_control_resp_t) + param.out_size);
        resp->module_id = MODULE_DSS_MAIN;
        resp->cmd_id = s_info->cmd_id;
        resp->result = ret;
        resp->out_size = param.out_size;
        if (param.out_size && param.out_para) {
            jx_memcpy(resp->out_para, param.out_para, param.out_size);
        }
    }
}
```

3.4.2 Screen On/Off Control

Screen on/off control is managed via an APP command interaction framework; the backlight is controlled by sending messages over IPC. cmd_id = DSS1_CMD_SET_BL

```
/**
 * @struct jx_dss_cmd_set_backlight_t
 * @brief MSG_Direction: Dss Service->Dss lib
 * @brief payload of MSG_ID: IPC_DSS_CONTROL_REQ - DSS1_CMD_SET_BL/DSS2_CMD_SET_BL
 */
typedef struct {
    uint32_t bl_level;
} __BYTE_PACKED__ jx_dss_cmd_set_backlight_t;
```

bl_level = 0 – Screen off

bl_level = 1 – Screen on

The screen-on/off duration is scheduled and controlled by the user.

3.4.3 Brightness Control

Brightness control is managed via an APP command interaction framework, with brightness adjustment performed by sending messages over IPC.


```
cmd_id = DSS1_CMD_GFX_EFFECT_CTRL
```

```
typedef enum {  
    JX_GFX_CTRL_DISTANCE           = 0x01,  
    JX_GFX_CTRL_FOV                = 0x02,  
    JX_GFX_CTRL_RECENTER           = 0x03,  
    JX_GFX_CTRL_BRIGHTNESS        = 0x04,  
} jx_gfx_effect_ctrl_type_t;  
typedef struct {  
    uint32_t type;                //jx_gfx_effect_ctrl_type_t  
    int32_t val;  
} __BYTE_PACKED__ jx_dss_cmd_gfx_effect_ctrl_t;
```

Within the `data` Structure,

```
type = JX_GFX_CTRL_BRIGHTNESS;
```

```
val = [0, 1024]
```

The display system's brightness control supports both Auto Mode and Manual Mode, with built-in minimum and maximum brightness protection; the default setting is Auto Mode.

When val = 0, trigger auto_mode.

When `val` is in the range (0, 1024), switch `auto\_mode` to `manual\_mode` and set the brightness according to the user-provided `val`. The input `val` will be clamped between `duty\_min` and `duty\_max`.

3.4.4 Repositioning Control

Relocalization control is managed via the APP command interaction framework and is implemented by sending messages through IPC.

```
cmd_id = DSS1_CMD_GFX_EFFECT_CTRL
```

The data remains the `jx\_dss\_cmd\_gfx\_effect\_CTRL\_t` structure.

```
type = JX_GFX_CTRL_RECENTER,
```

```
val = 0
```


3.4.5 Color Parameter Control

The relevant interfaces are located in `src/application/pq\_algo/inc/dss1\_pq\_algo\_api.h` /**

* @brief Set color adjustment parameters for PQ algorithm.

*

* This interface is used to configure color tuning parameters such as

* brightness, contrast, saturation, hue, or other color-related controls

* for the specified display pipeline.

*

* @param[in] pipe_id Display pipeline ID. 0:Left 1:Right

* @param[in] sub_pipe VIDEO/GRAPHIC. 0:VIDEO, 1:GRAPHIC

* @param[in] color_para Pointer to dss1_algo_color_para_t structure.

*

* @return

* - JX_OK : Parameters were applied successfully.

* - Others : Failed to apply parameters.

*/

```
jx_err_t dss1_pq_algo_set_color_param(jx_u8_t pipe_id, jx_u8_t sub_pipe, jx_void_t  
*color_para);
```

```
struct dss1_algo_color_para_t {  
    jx_float_t contrast; //-1, 1  
    jx_float_t saturation; //-1, 1  
    jx_float_t color_temperature; //-1, 1  
    jx_float_t tint; //-1, 1  
};
```

Parameter	Meaning	Input range	Replenish
c	Contrast	[-1, 1]	0 Indicates no change
s	Saturation	[-1, 1]	0 Indicates no change
t	Temperature, Color Temperature	[-1, 1]	0 Indicates no change
ti	Tint,hue	[-1, 1]	0 Indicates no change

instance 1:

```
c = 0.15f; // Slightly increase contrast
s = 0.10f; // Slightly increase saturation
t = 0.00f; // Color temperature remains constant
ti = 0.00f; // Constant hue
```

Example 2: Disable color parameter adjustment

Set all input parameters to `0.0f` to disable color adjustment:

```
c = 0.0f;
s = 0.0f;
t = 0.0f;
ti = 0.0f;
```

3.4.6 Color Filter Control

The relevant interfaces are located in `src/application/pq\_algo/inc/dss1\_pq\_algo\_api.h` /**

```
* @brief Set color filter configuration for PQ algorithm.
*
* This interface is used to configure color filter effects such as LUT-based
* filters, style filters, or other color transformation effects for the
* VST or GFX path.
*
* @param[in] pipe_id    Display pipeline ID. 0:Left 1:Right
* @param[in] sub_pipe    VIDEO/GRAPHIC. 0:VIDEO, 1:GRAPHIC
* @param[in] filter_para  Pointer to dss1_algo_color_filter_t structure.
*
* @return
* - JX_OK                : Filter configuration was applied successfully.
* - Others                : Failed to apply filter configuration.
*/
```

```
jx_err_t dss1_pq_algo_set_color_filter(jx_u8_t pipe_id, jx_u8_t sub_pipe, jx_void_t
*filter_para);
```

```
struct dss1_algo_color_filter_t {
    jx_bool_t enable; // en or disable filter function
    jx_u8_t apply_mode; // 0:instant 1:transition 2:fixed_alpha
    jx_u8_t lut_source; // built-in or custom
    jx_u32_t *custom_lut; // pointer of lut table, [2187]
    jx_u8_t base_lut_id;
    jx_u8_t target_lut_id;
    jx_u32_t transition_frames;
    jx_float_t alpha;
};
```

The Color Filter Interface supports the following features:

- (1) Supports triggering filter changes;
- (2) Supports custom LUTs;
- (3) Supports LUT intensity adjustment; adjustment modes include: Gradient/Immediate Switch/Multi-Frame Smoothing. Input parameters include:

Variable	Input range	Windows default	Meaning	Explain
enable	TRUE / FALSE	FALSE	Is GM enabled?	Functional switch
apply_mode	0: MODE_INSTANT / 1: MODE_TRANSITION / 2: MODE_FIXED_ALPHA	MODE_INSTANT	Switching mode	Gradient / Switch Now / Fixed Intensity
lut_source	0 :LUT_BUILTIN / 1 :LUT_CUSTOM	LUT_BUILTIN	LUTsource	Built-in LUT or user-defined LUT; when 'lut_source' is not specified, the default is 0: LUT_BUILTIN.
custom_lut	NULL / valid pointer	NULL	User-defined LUT, length: 2187	Target LUT source = LUT_CUSTOM – applies when this is set.
base_lut_id	[0, 4]	0	Built-inLUT	Input when 'apply_mode = INSTANT' Output; Starting LUT during TRANSITION; Used as blend base during FIXED_ALPHA
target_lut_id	[0, 4]	0	Target LUT	Color filter final state
transition_frames	[1, 1024]	1	Transition Frame Count	Only MODE_TRANSITION applies.
alpha	[0, 1]	0.0	Mixed Filter Intensity	Only MODE_FIXED_ALPHA is applicable.

Example 1: The user wants to switch the screen theme immediately:

enable = true;

apply_mode = GM_BLEND_MODE_INSTANT;

```
base_lut_id    = 0;
target_lut_id  = 2;
```

Example 2: Suitable for scenarios requiring smooth transitions—such as scene switching, mode switching, or UI animations.

```
enable         = true;

apply_mode     = GM_BLEND_MODE_TRANSITION;

base_lut_id    = 0;
target_lut_id  = 1;
transition_frames = 8;
```

Example 3: Fixed Ratio Mixing Mode

Ideal for scenarios where customers wish to implement a "Style Strength" slider.

```
.enable        = true;

apply_mode     = GM_BLEND_MODE_FIXED_ALPHA;

base_lut_id    = 0;
target_lut_id  = 1;
alpha          = 0.3f;

among:

alpha = 0.0:    Use base_lut_id exclusively.
alpha = 1.0:    Use `target_lut_id` exclusively.
alpha = 0.3:    Approximately 70% base LUT + 30% target LUT
```

Example 4: Using a custom LUT:

When the built-in LUT does not meet the customer's color grading requirements, a custom LUT can be imported.

```
static jx_u32_t user_lut[2187] = { ... };

enable         = true;

apply_mode     = GM_BLEND_MODE_TRANSITION;

lut_source     = GM_BLEND_LUT_CUSTOM;

custom_lut     = user_lut;

base_lut_id    = 0;
target_lut_id  = 4;
transition_frames = 8;
```

matters need attention:

The custom LUT length must be 2187; during a single power-on cycle, the Custom LUT will overwrite the original LUT 4; after this overwrite, LUT 4 will continue to refer to this Custom LUT until the device is powered on again or the platform reloads the LUT configuration.

When the target LUT uses a custom LUT, ensure that `lut\_source = 1`.

3.4.7 Sharpening Control

The relevant interfaces are located in `src/application/pq\_algo/inc/dss1\_pq\_algo\_api.h`/ **

* @brief Set sharpness gain for PQ algorithm.

*

* This interface is used to configure the sharpness enhancement gain

* for the specified display pipeline.

*

* @param[in] pipe_id Display pipeline ID. 0:Left 1:Right

* @param[in] sharpeness_gain Sharpness gain value. [0-1.0]

*

* @return

* - JX_OK : Sharpness gain was applied successfully.

* - Others : Failed to apply sharpness gain.

*/

```
jx_err_t dss1_pq_algo_set_sharpeness_gain(jx_u8_t pipe_id, jx_float_t sharpeness_gain);
```

Gain = 0 – no sharpening effect

Gain = 1 – Maximum sharpening effect

4. Camera

4. 1 Camera Sensor Configuration

Camera customization parameters are configured using JSON format, in compliance with the fundamental JSON configuration specifications. To enhance accuracy and usability in code implementation, certain constraints and extensions are applied during configuration; for detailed information, please refer to the "JSON Format Specification" document published by the Toolset.

As shown in Figure 4.1, in terms of layout, the camera customization parameters are organized using a tree structure that extends downward from the top level. The top-level JSON file primarily serves as a summary of all camera configuration information across the entire project; it contains links to the JSON files at the next level, which describe the configuration details for each individual logical camera. Following this pattern, the structure continues downward, linking to the JSON files containing the sensor configuration, OTP configuration, CSI configuration, and PHY configuration corresponding to the actual hardware components; it also includes links to files belonging to other subsystems—for example, the BSP-related files are used during the power-on and power-off procedures for the sensor. Each subsystem is interconnected in a hierarchical manner, and this file focuses specifically on camera-related aspects.

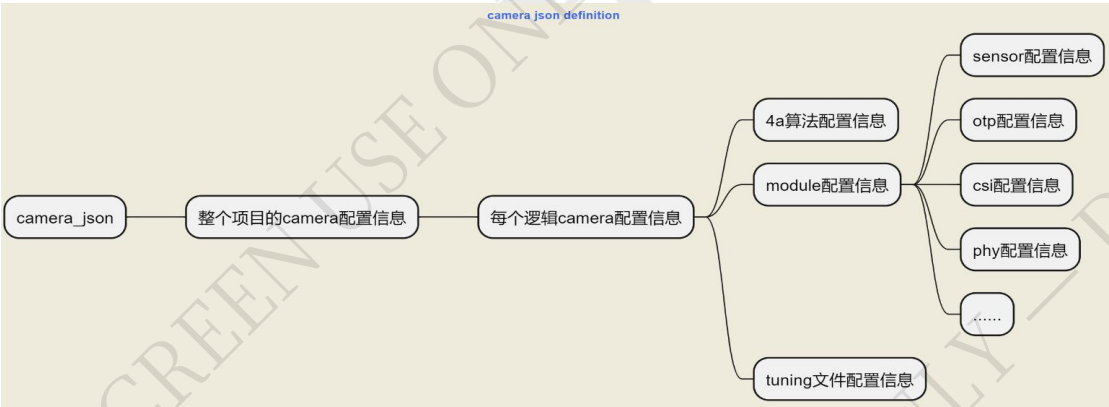


Figure 4.1: Camera JSON File Framework Diagram

4. 1. 1 Sensor JSON Directory Structure

Table of Contents: X100_antman_A1_20260520_200228\config\cam\jx10x_style\antman

- ├── all_cam_custom_cfg.json #The sole entry point for Camera JSON files
- ├── cam_vst.json # vst logical cam
- ├── cam_slam.json # slam logical cam
- └── cam_et.json

- └── cam_ft.json
 - └── Module # cam – Module Path
 - └── cam_diff_cfg_atm.json # Difference Configuration
 - └── cam_power_cfg_atm.json # Physical camera power supply configuration
 - └── imx480_sunny.json # Physical Camera Module
 - └── sensor # Physical sensor
 - └── imx480_2328x1744_1128M_90fps_slave_atm.json
 - └── OC0TB1B_400x400_90fps_raw8_slave.json

4. 1. 2 Sensor JSON File Description

This section primarily introduces the JSON files related to Camera configuration and their respective descriptions. The X100 chip is first configured at the "logical camera" level, then linked to the JSON files corresponding to each "physical camera" under that logical camera; finally, within these physical camera JSON files, the JSON files to which the sensor module should be linked are configured. At this level, users can view the JSON files used for configuring module-specific differences, power supply configurations, and module sensor settings. To configure these files, locate the relevant JSON files using the specified JSON path and follow the configuration steps outlined in the following guide. The table below provides an overview of the files required for this process.

Document name	Description	Remarks
all_cam_custom.json	Users need to configure all required camera settings, configuring them on a per-logical-camera basis; the X100 supports the configuration of up to 9 logical cameras. For a detailed list of the available types, refer to Section 3.3.2.	This document serves as the sole entry point for users to edit the Camrera JSON file.
cam_vst.json	This file allows users to specify the number of physical cameras required for the logical cameras defined in all_cam_custom.json, as well as the relevant configuration settings for the sensor modules to which these physical cameras should be linked. Depending on the type of logical camera, the appropriate logical-camera-specific configuration file should be referenced; for example, the vst camera file is used here – the same approach applies to other logical camera types.	Other files of the same type (e.g., cam_main.json, cam_ir.json) have the same structure; the only difference lies in their file names or enumerations.

imx480_sunny_l.json	This file allows users to specify cam_vst.json (its This section contains the physical camera-related configurations entered in the (e.g., `logical-camera` file), such as module-specific configuration parameters, power supply configurations, and the JSON file name and path linked to the module sensor settings configuration. Using the IMX480 configuration as an example, other physical camera configurations follow a similar pattern.	The model can be copied and renamed based on the actual physical camera model being used, for example. imx586_xxx.json。
imx480_1872x1096_1128M_90fps_slave.json	This document contains the sensor configuration settings associated with the sensor connected in the imx480_sunny_l.json file (similar files exist for other physical cameras). Here, users can configure parameters such as the sensor's device address, power-on/off procedures, and the registers required for sensor initialization.	For other sensors (e.g., IMX 471, OV7251), locate the corresponding JSON file; if such a file does not exist, create a new. json file accordingly.
cam_power_config.json	This document contains the power supply-related configuration settings for the sensor connected in the imx480_sunny_l.json file (other physical camera files follow a similar format). Here, users should verify whether the sensor's power supply configuration aligns with the hardware resource usage.	Unified power supply configuration management.
cam_diff.cfg	This document primarily describes the module-specific configuration settings; users are not permitted to modify them.	This is a read-only configuration.

4.1.3 Configure Logical Camera

- a. Users should modify the logical camera settings according to the camera they intend to use.

As shown in Figure 4.1.3.1, modify the all_cam_custom_cfg.json file; users should pay particular attention to the parameters within the red box, while the parameters within the blue box require custom configuration by the user.

Parameter	Meaning	Explain	Remarks
-----------	---------	---------	---------

logical_cam_num	Number of Logical Cameras required (i.e., how many logical cameras the system will enable this time)	1 ~ 9 (integer) The entered value determines the outcome. Fix cam_logical_cfg_array The code will actually be compiled or parsed. The first N terms	For example, `logical_cam_num = 3` → only the element at index 3 of the array will be read. 1、Items 2 and 3; Items 4–9 are omitted.
cam_logical_cfg_array	Number with capacity 9 Group – Save all Logical Camera configurations supported by X 100.	Fixed length: 9; each entry consists of two columns (logical_cam_role & logical_cam_setting).	The X100 supports up to 9 different Logical modes. Camera
logical_cam_role	Logical Camera Enumerated Type (Corresponding Figure) (4.1.3.2 – "Options")	Taken from the enumeration list in Figure 4.1.3.2 below.	Must be in uppercase with an underscore. It must be used in conjunction with logical_cam_setting. JSON file matching
logical_cam_setting	This logic camera corresponds to Logical Cam JSON file path (with.json extension)	Relative path, e.g., cam_vst.json	Each logical_cam_role must have a unique JSON document. Item

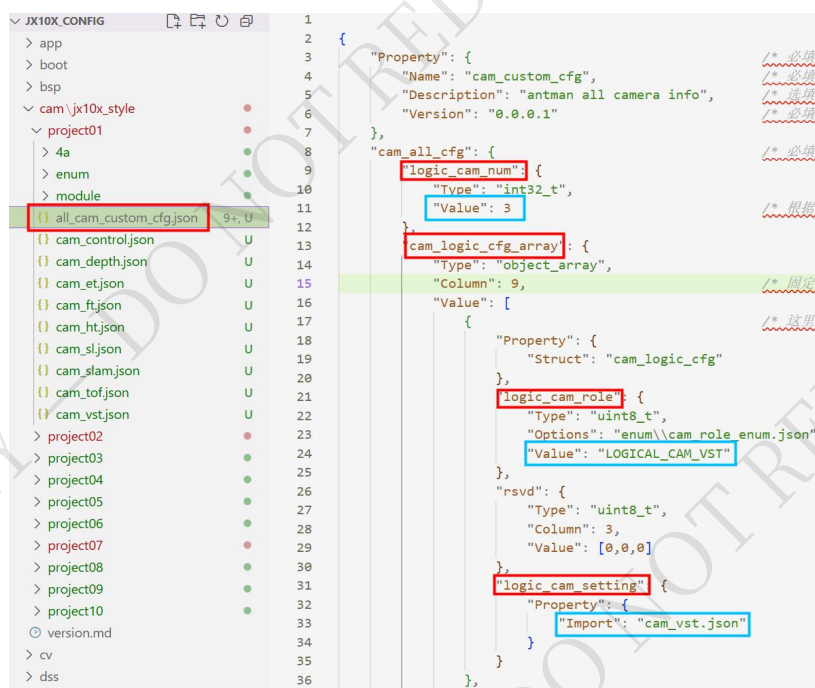


Figure 4.1.3.1 – Logical Camera Modification

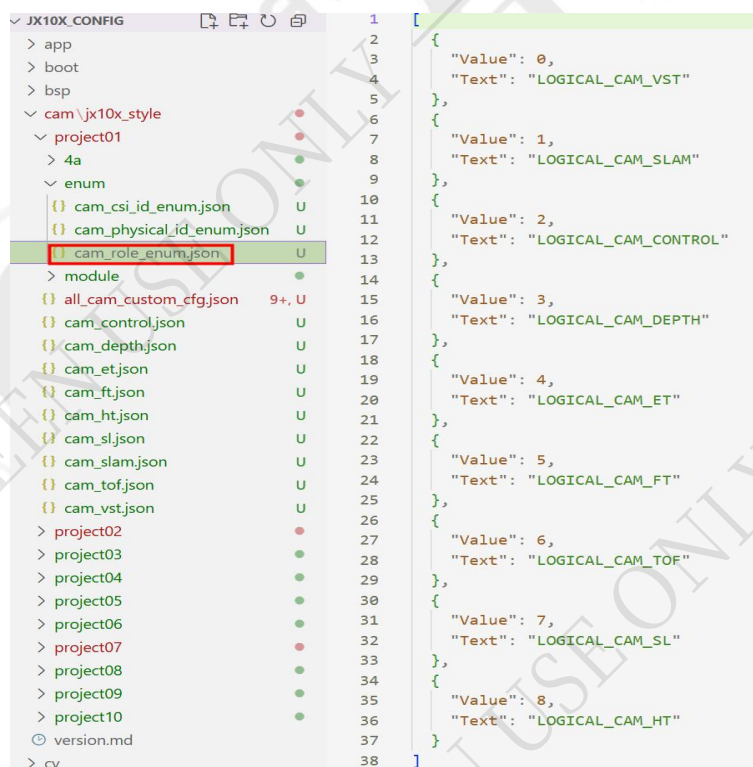


Figure 4.1.3.2 – Logical Camera Enumerated Type

b. Configure using the Logical Camera unit.

As shown in Figure 4.1.3.3, make the necessary modifications to the logical camera JSON file that the user intends to use; for example, use `cam\_vst.json` as a sample – other logical camera files follow the same pattern and will not be described in further detail. The user should pay particular attention to the parameters within the red box; the parameters within the blue box are those that the user needs to customize and configure.

Parameter	Meaning	Explain	Remarks
phys_cam_num	Actual project usage Number of physical cameras.	(Using two VST sensors as an example)	X100 – Sequential Configuration
post_streamon_enable	Same Logical Camera (Logical Camera) Head) at sensor inlet Whether fsync is enabled (Synchronous) for multi-channel support Start simultaneously.	Default open	For most scenarios, the default settings are sufficient.
cam_cfg_t	Logical Camera Configuration information for each Physical Camera.	Array-based compilation and parsing	Each array parameter corresponds to a Physical Camera
phys_id	The ID (enumeration value) of this Physical Camera in the X100	CAM_PHYSICAL_VST_L (Left VST)	The enum definition is located in `enum/cam_physical_idEnum.json`, as shown in the figure. 4.1.3.4 As shown,
csi_id	The Physical Camera used	CAM_CSI_VST_L (Location of the left-side VST) CSI channel)	Users can configure the relevant physical camera according to the demo version.

	CSI Channel (enum value).		
module_cfg	Path to the camera module information configuration file associated with this physical camera	Using imx480_sunny_1.json Take...as an example	

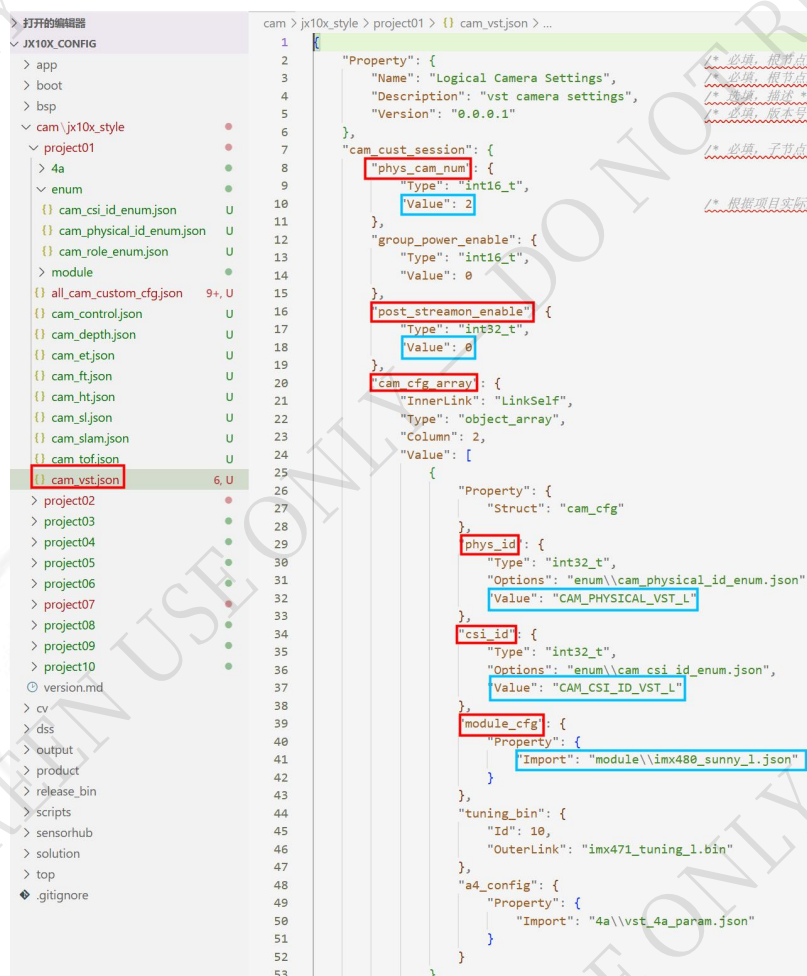


Figure 4.1.3.3 – Logical Camera File Modification



Figure 4.1.3.4 – Physical Camera ID Enumerated Type

4.1.4 Configuration– Physical Camera

Configure information related to the physical camera.

As shown in Figure 4.1.4.1, the user needs to make modifications to the physical camera JSON file being used; the file path is located in the "module/" subfolder and corresponds to the configuration specified in Section 4.1.3, b. (under "module_cfg"). For example, use the file imx480_sunny_l.json. The user should pay particular attention to the parameters highlighted in the red box; the parameters highlighted in the blue box are those that the user needs to customize and configure.

Parameter	Meaning	Explain	Remarks
cam_module	Including physical Camera (Physical Camera) (Head) The overall structure of all sub-modules Physical configuration object. Phase In `cam_cfg` Outermost module_cfg, Further subdivided internally hfov、diff_cfg、 power_cfg、	Take the IMX480 as an example.	

	sensor_cfg, fno_x1000, etc.		
hfov	The horizontal field of view of this camera (Single). Angle (°).	Take the IMX480 as an example.	Must be consistent with the calibration values specified in the actual sensor datasheet.
diff_cfg	X100 Side-Differentiated Configuration	If a phase appears in the tool For different parameters, the values specified in `diff_cfg` shall prevail.	Optional; most projects do not require any modifications.
power_cfg	Sensor power-on/off related hardware information configuration file path (relative path).	Typically includes parameters such as power supply GPIO, PMIC, and delay.	
sensor_cfg	Sensor initialization/ Mode Setup – Configuration File Path (relative path).	Take the IMX480 as an example.	Also uses a relative path; file content is based on this path. sensor datasheet
fno_x1000	Sensor aperture value (f-number), i.e., f/ Price	Take the IMX480 as an example.	Must match the actual lens aperture.

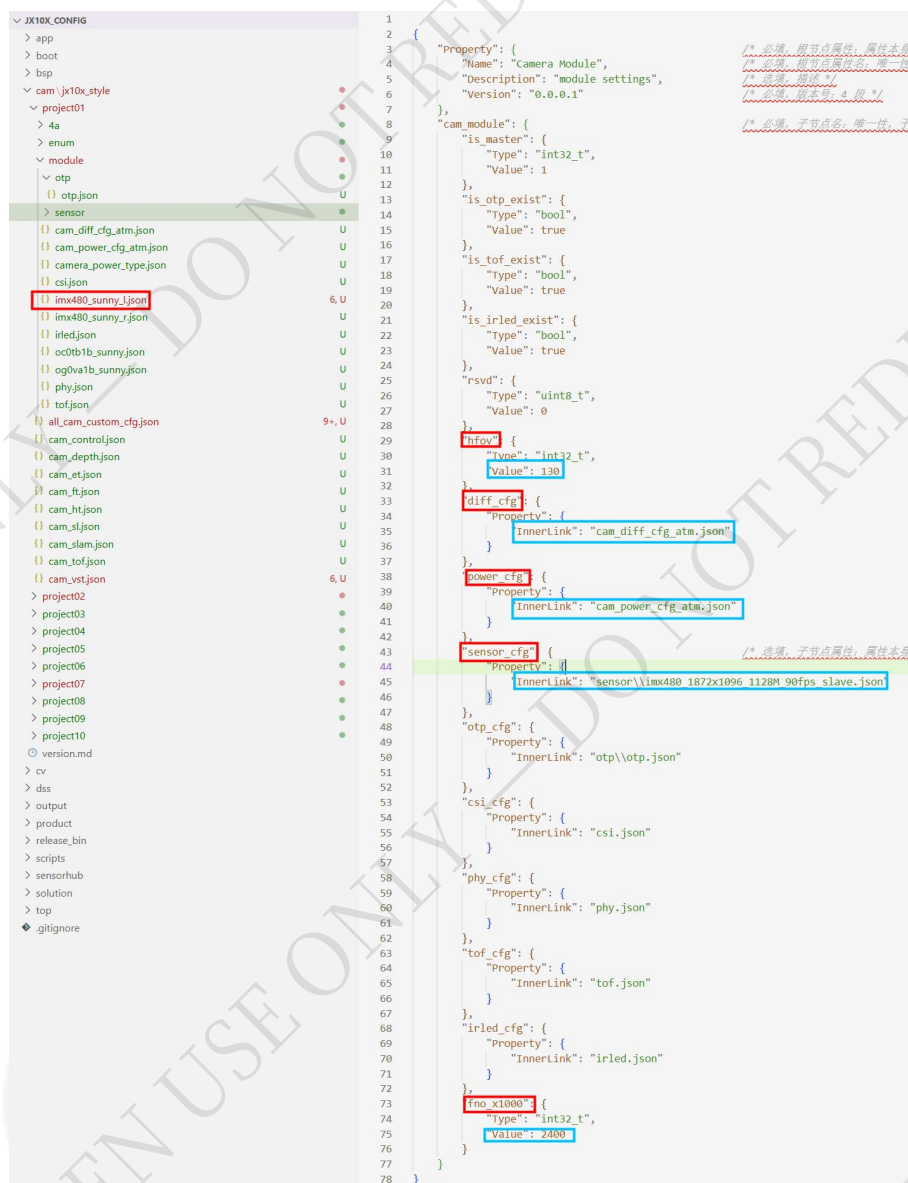


Figure 4.1.4.1 – Physical Camera Related Configuration

4.1.5 Configure Sensor Module Parameters

a. Configure Sensor Module-Related Information.

As shown in Figure 4.1.5.1, modify the sensor JSON file required by the user; the file path is located in the "module//sensor/" subfolder, corresponding to the "sensor_cfg" configuration described in Section 4.1.4. Here,

Take `imx480\_1872x1096\_1128M\_90fps\_slave.json` as an example. Users should pay close attention to the parameters within the red box; the parameters within the blue box need to be customized and configured by the user.

"sensor_name": Users simply need to enter the name of the sensor they are using.

"module_name": Users simply need to fill in the information regarding the sensor module they are using.



Figure 4.1.5.1 Sensor-Related Configuration

Figure 4.1.5.2 shows the configuration settings related to Sensor Slave Info. Users should pay particular attention to the parameters within the red box; the blue box contains parameters that users need to configure manually.

Parameter	Meaning	Explain	Remarks
slave_addr	Sensor – Slave device address on the I ² C bus (8-bit; read/write address excluded) Bit)	For example, for the IMX 480, enter 0x10.	Enter a 7-bit address.
addr_type	I ² C Address Width Class Type (length of register address)	The default value is 1.	Most sensors use 16-bit resolution. Address
data_type	I ² C – Data Width Type (length of register data)	The default value is 1.	Most Sensor registers have a width of 8 bits.

	Degree)		bit
sensor_id_reg_addr_high	The address of the register containing the 8-bit Chip-ID Location	Take the IMX480 as an example.	Majority ISP/Sensor Manual Session Clearly mark both IDs. Register (High/Low), from the sensor DSheet provided
sensor_id_high_byte	The actual 8-bit Chip-ID value	Take the IMX480 as an example.	
sensor_id_reg_addr_low	The address of the register containing the Chip-ID's lower 8 bits Location	Take the IMX480 as an example.	
sensor_id_low_byte	The actual value of the lower 8 bits of the Chip-ID	Take the IMX480 as an example.	

The "i2c_id" and "i2c_freq" parameters in Figure 4.1.5.2 are identical to those defined in "diff_cfg"; therefore, the configuration specified in "diff_cfg" shall prevail, and no further elaboration is required here.

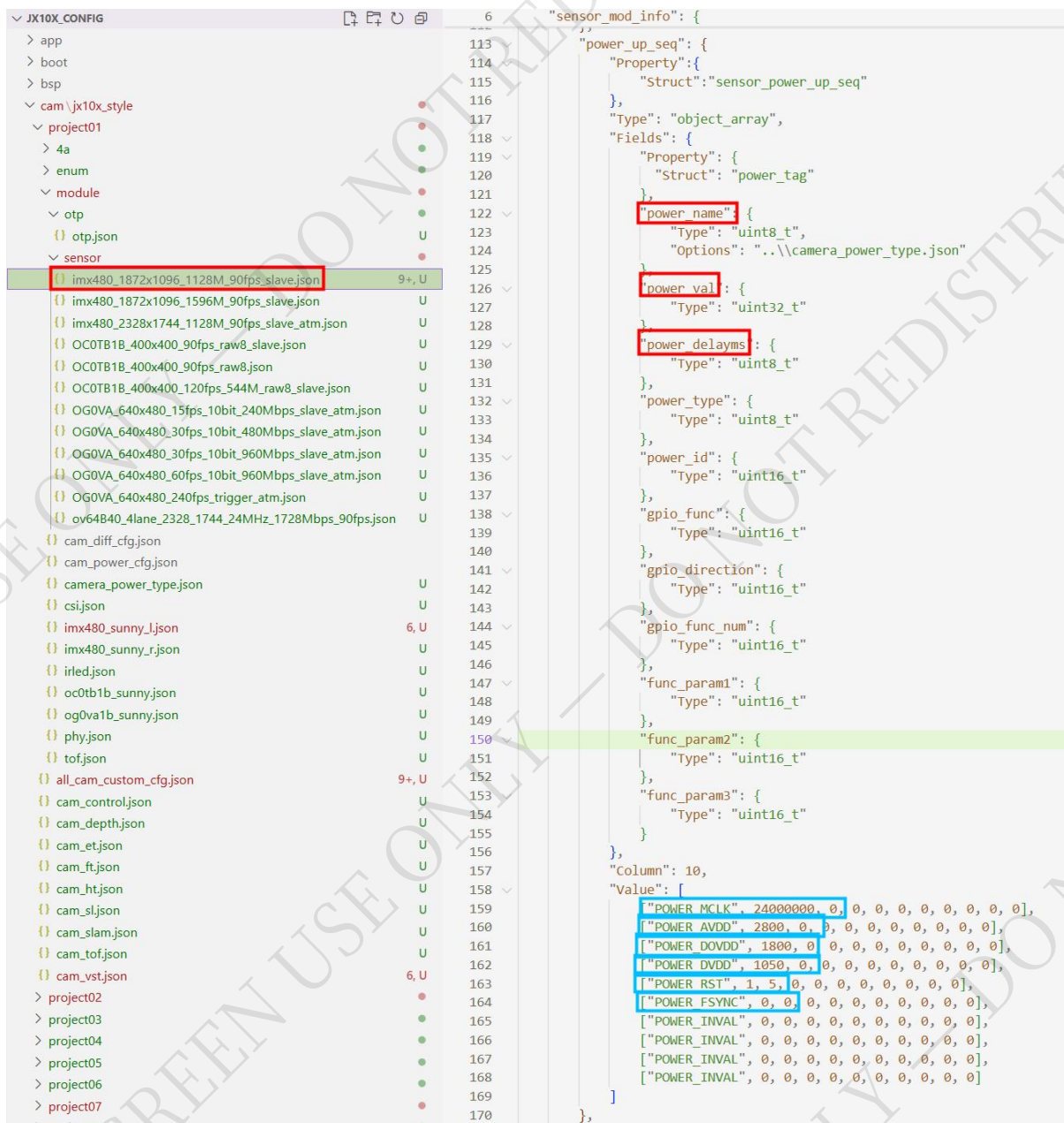


Figure 4.1.5.3 – Sensor Power-Up Related Configuration

▼ JX10X_CONFIG	1	[
> app	2	{
> boot	3	{
> bsp	4	{
> cam \jx10x_style	5	{
> project01	6	{
> 4a	7	{
> enum	8	{
> module	9	{
> otp	10	{
> otp.json	11	{
> sensor	12	{
> imx480_1872x1096_1128M_90fps_slave.json	13	{
> imx480_1872x1096_1596M_90fps_slave.json	14	{
> imx480_2328x1744_1128M_90fps_slave_atm.json	15	{
> OC0TB1B_400x400_90fps_raw8_slave.json	16	{
> OC0TB1B_400x400_90fps_raw8_slave.json	17	{
> OC0TB1B_400x400_120fps_544M_raw8_slave.json	18	{
> OG0VA_640x480_15fps_10bit_240Mbps_slave_atm.json	19	{
> OG0VA_640x480_30fps_10bit_480Mbps_slave_atm.json	20	{
> OG0VA_640x480_30fps_10bit_960Mbps_slave_atm.json	21	{
> OG0VA_640x480_60fps_10bit_960Mbps_slave_atm.json	22	{
> OG0VA_640x480_240fps_trigger_atm.json	23	{
> ov64B40_4lane_2328_1744_24MHz_1728Mbps_90fps.json	24	{
> cam_diff_cfg.json	25	{
> cam_power_cfg.json	26	{
> camera_power_type.json	27	{
> csi.json	28	{
> imx480_sunny_l.json	29	{
> imx480_sunny_r.json	30	{
> ired.json	31	{
> oc0tb1b_sunny.json	32	{
> og0va1b_sunny.json	33	{
> phy.json	34	{
> tof.json	35	{
> all_cam_custom_cfg.json	36	{
> cam_control.json	37	{
> cam_depth.json	38	{
> cam_et.json	39	{
> cam_ft.json	40	{
> cam_ft.json	41	{
> cam_ft.json	42	{
> cam_ft.json	43	{
> cam_ft.json	44	{
> cam_ft.json	45	{
> cam_ft.json	46	{

Figure 4.1.5.4: 'Sensor Power Name' Enumerated Type

As shown in Figure 4.1.5.5, this section describes the sensor init-related configurations, which are configured in the form of an array. The parameters "reg_addr", "reg_value", and "delay_time" constitute a set of configuration parameters; for detailed information, please refer to the following section. This example uses the IMX480 chip as the target device for configuration. Users should pay particular attention to the parameters highlighted in the red box; the parameters within the blue box are those that users need to configure customarily.

"Column": The number of internal sensor registers that need to be configured during sensor initialization.

"reg_addr": The internal sensor register address that needs to be configured during sensor initialization.

"reg_value": The value to be written into the sensor's internal register during sensor initialization.

"delay_time": The delay value required after configuring this sensor register.

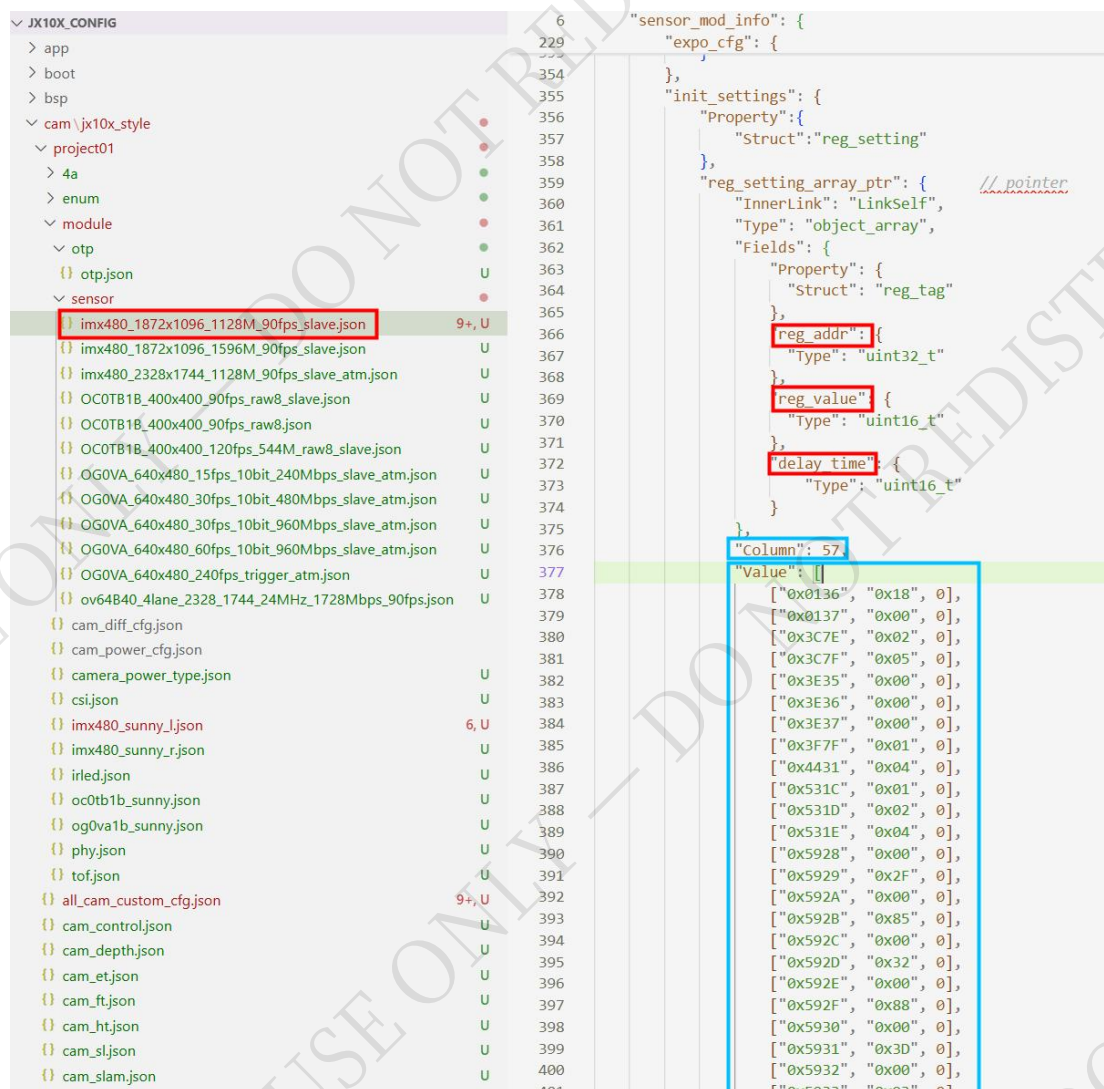


Figure 4.1.5.5 Sensor Init Configuration

As shown in Figure 4.1.5.6, this section describes the Sensor Set Mode configuration; the register configuration is similar to that of the Sensor Init configuration and will not be elaborated upon here. Users should pay particular attention to the parameters within the red box; the parameters within the blue box are those that users need to customize and configure. (Note: Two modes are available for configuration; the configuration for the first mode takes precedence.)

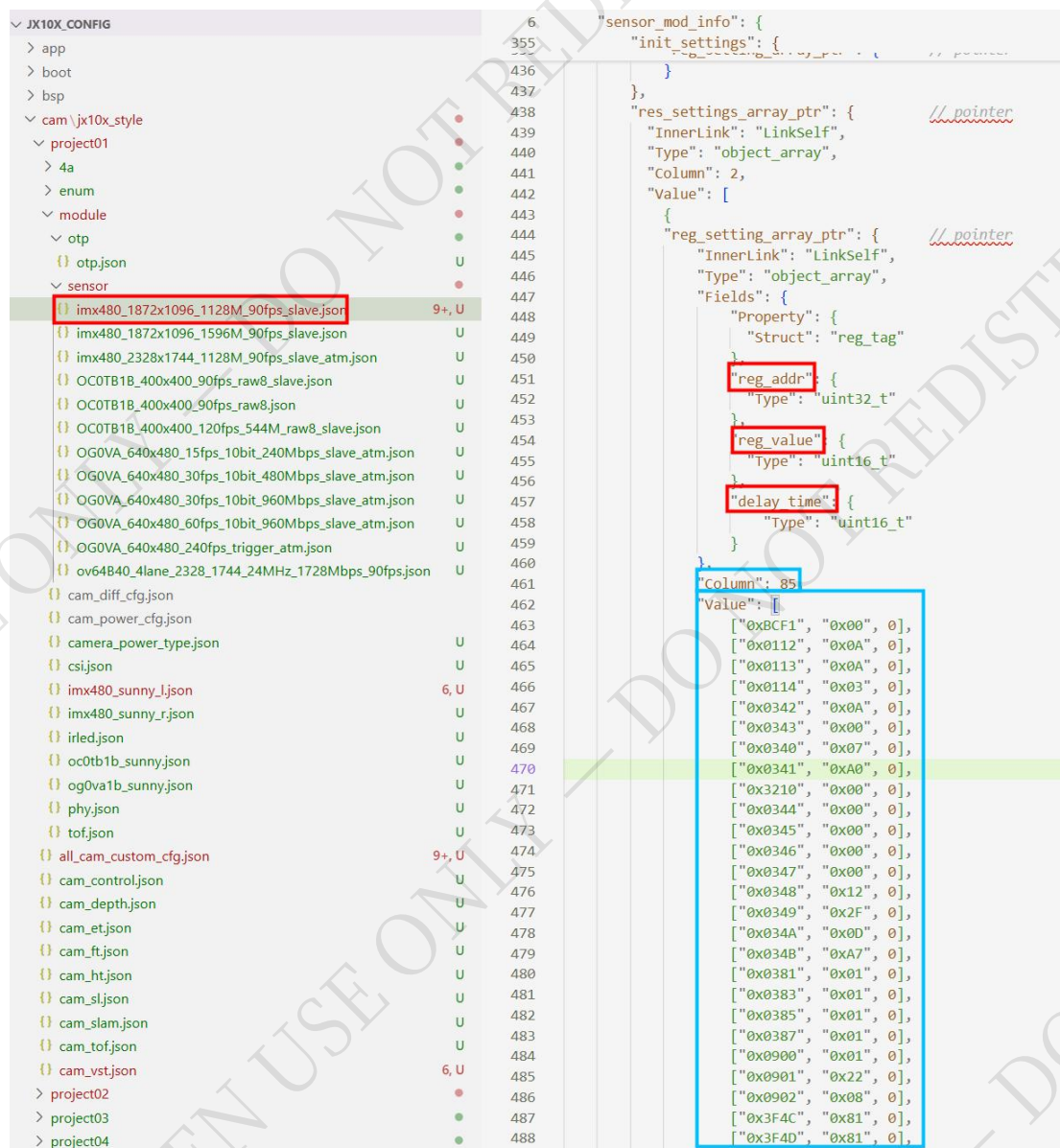


Figure 4.1.5.6 Sensor Set Mode Register Configuration

As shown in Figure 4.1.5.7, this section illustrates the configuration of parameters related to the Sensor Set Mode. Users should pay particular attention to the parameters within the red box; the blue box indicates the user.

Parameters That Require Custom Configuration.

Parameter	Meaning	Unit	Remarks
-----------	---------	------	---------

out_width	Number of horizontal pixels in the output image under this operating mode (Width)	Pixel	Corresponds to the sensor's Active Area, excluding horizontal blanking.
out_height	Number of vertical pixels in the output image under this operating mode (Highness)	Pixel	Together with 'out_width', it determines the final resolution.
line_length_pclk	Total number of pixels The number of pixels (including horizontal blanking) is usually equal to the horizontal count. Number of clock cycles	Pixel	Relationship to 'out_width': If 'line_length_pclk ≥ out_width', then the difference is hblank。
frame_length_lines	Total number of rows per frame (including vertical rows) Blanking – the number of rows that need to be read per frame	Line	With out_height Relationship: 'frame_length_lines ≥ out_height'; the difference is then calculated. vblank。
tline	The time required for single-line sampling (i.e., one line of clock cycles) is determined by the pixel clock. (pclk)decide	Microsecond(μs)	Commonly used to calculate frame rate: $\text{fps} = 1 / (\text{tline} * \text{frame_length_lines})$ 。
fps	Number of frames per second that can be output under this mode	Frames per second (fps)	Determines jointly by tline and frame_length_lines
dt (data type)	Output image data format (pixel bit depth, row layout)		

	Columnar format)		
--	------------------	--	--

JX10X_CONFIG	6	"sensor_mod_info": {
> app	438	"res_settings_array_ptr": { // pointer
> boot	442	"value": [
> bsp	443	{
cam \jx10x_style	444	"reg_setting_array_ptr": { // pointer
project01	549	{
4a	550	"out_width": {
enum	551	"type": "uint16_t",
module	552	"value": 1872
otp	553	}
otp.json	554	"out_height": {
sensor	555	"type": "uint16_t",
imx480_1872x1096_1128M_90fps_slave.json	556	"value": 1096
imx480_1872x1096_1128M_90fps_slave.json	557	}
imx480_1872x1096_1596M_90fps_slave.json	558	"line_length_pclk": {
imx480_2328x1744_1128M_90fps_slave_atm.json	559	"type": "uint16_t",
OC0TB1B_400x400_90fps_raw8_slave.json	560	"value": 2560
OC0TB1B_400x400_90fps_raw8_slave.json	561	}
OC0TB1B_400x400_120fps_544M_raw8_slave.json	562	"frame_length_lines": {
OG0VA_640x480_15fps_10bit_240Mbps_slave_atm.json	563	"type": "uint16_t",
OG0VA_640x480_30fps_10bit_480Mbps_slave_atm.json	564	"value": 1962
OG0VA_640x480_30fps_10bit_960Mbps_slave_atm.json	565	}
OG0VA_640x480_60fps_10bit_960Mbps_slave_atm.json	566	"tline": {
OG0VA_640x480_240fps_trigger_atm.json	567	"type": "uint32_t",
ov64840_4lane_2328_1744_24MHz_1728Mbps_90fps.json	568	"value": 5664
cam_diff_cfg.json	569	}
cam_power_cfg.json	570	"fps": {
camera_power_type.json	571	"type": "uint8_t",
csi.json	572	"value": 90
imx480_sunny_l.json	573	}
imx480_sunny_r.json	574	"mode": {
lired.json	575	"type": "uint8_t",
oc0tb1b_sunny.json	576	"value": 0
og0va1b_sunny.json	577	}
phy.json	578	"vc": {
tof.json	579	"type": "uint8_t",
all_cam_custom_cfg.json	580	"value": 0
cam_control.json	581	}
cam_depth.json	582	"dt": {
cam_et.json	583	"type": "uint8_t",
cam_ft.json	584	"value": "0x2B"
cam_ht.json	585	}
cam_sl.json	586	"image_format": {
cam_slam.json	587	"type": "uint8_t",
cam_tof.json	588	"value": 3
cam_vst.json	589	}
project02	590	"bayer_pattern": {
project03	591	"type": "uint8_t",
project04	592	"value": 0
project05	593	}
project06	594	"emb_enable": {
	595	"type": "uint8_t",
	596	"value": 0
	597	}
	598	"emb_vc": {
	599	"type": "uint8_t",
	600	"value": 0
	601	}

Figure 4.1.5.7 Sensor Set Mode Configuration

As Shown in Figure 4.1.5.8, These Represent the Configuration Settings Related to the Sensor Stream On/off Functionality; These Stream On/off Configuration Settings Are Configured as an Array.

Users should pay close attention to the parameters within the red box; the blue box contains parameters that need to be customized by the user. These configurations are similar to those used in 'sensor init'; users should configure them according to the sensor specifications by entering the values as an array in the "Value" field.

```
6 "sensor_mod_info": {
838   "groupoff_settings": {
862   },
863   "stream_on_settings": {
864     "Property": {
865       "Struct": "reg_setting"
866     },
867     "reg_setting_array_ptr": { // pointer
868       "InnerLink": "LinkSelf",
869       "Type": "object_array",
870       "Fields": {
871         "Property": {
872           "Struct": "reg_tag"
873         },
874         "reg_addr": {
875           "Type": "uint32_t"
876         },
877         "reg_value": {
878           "Type": "uint16_t"
879         },
880         "delay_time": {
881           "Type": "uint16_t"
882         }
883       },
884       "Column": 1,
885       "Value": [{"0x0100", "0x01", 0}]
886     },
887   },
888   "stream_off_settings": {
889     "Property": {
890       "Struct": "reg_setting"
891     },
892     "reg_setting_array_ptr": { // pointer
893       "InnerLink": "LinkSelf",
894       "Type": "object_array",
895       "Fields": {
896         "Property": {
897           "Struct": "reg_tag"
898         },
899         "reg_addr": {
900           "Type": "uint32_t"
901         },
902         "reg_value": {
903           "Type": "uint16_t"
904         },
905         "delay_time": {
906           "Type": "uint16_t"
907         }
908       },
909       "Column": 1,
910       "Value": [{"0x0100", "0x00", 0}]
911     },
912   },
913   ...
914 }
915 }
```

Figure 4.1.5.8 Sensor Stream On/Off Configuration

As shown in Figure 4.1.5.9, these represent exposure-related configurations. Users should pay particular attention to the parameters within the red box; the blue box contains parameters that users need to customize and configure.

"max_analog_gain": The maximum analog gain value for the sensor. (Note: This value should be increased by a factor of 100.)

"vertical_offset": The maximum exposure offset supported by the sensor.

"max_line_count": The maximum number of exposure lines supported by the sensor.



Figure 4.1.5.9 – Exposure Control Info Configuration

As shown in Figure 4.1.5.10, this section illustrates the configuration settings for the Exposure and Gain registers. Users should pay particular attention to the parameters within the red box; the blue box contains parameters that the user needs to customize and configure manually.

"expo_reg_addr": The configuration address for the low-order register of the exposure-related sensor.

"expo_reg_count": Number of exposure-related sensor registers.

"ana_gain_addr": The configuration address for the low-order register of the analog gain-related sensor.

"ana_gain_count": Number of analog gain-related sensor registers.

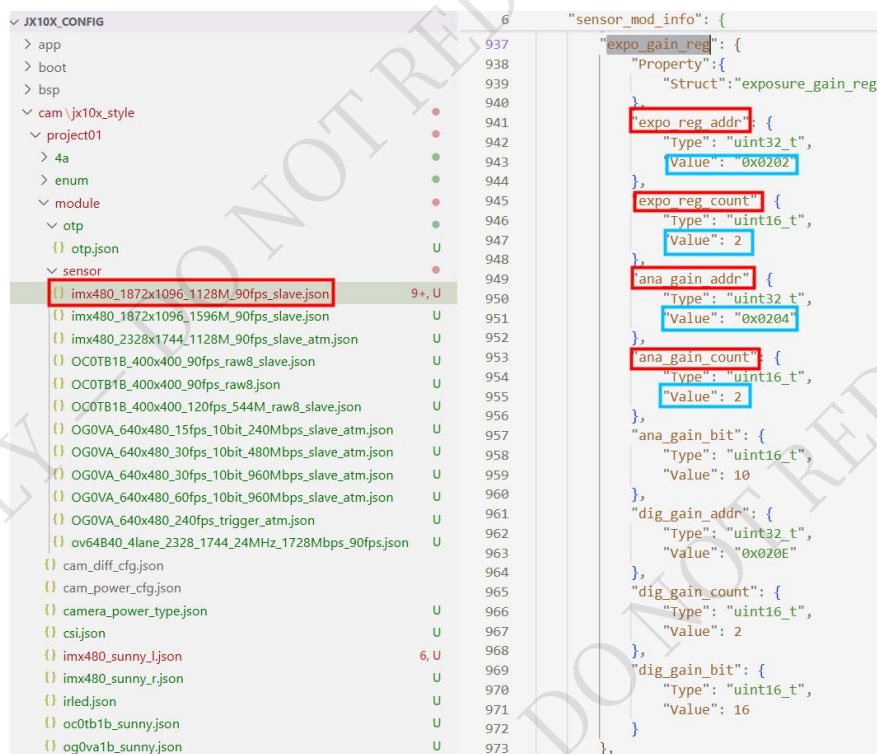


Figure 4.1.5.10 – Exposure and Gain Register Configuration

As shown in Figure 4.1.5.11, this section illustrates the configuration settings related to the mirror and flip functions. Users should pay particular attention to the parameters within the red box; the blue box contains parameters that the user needs to customize and configure manually. These configurations are similar to those for the 'sensor init' function: users should configure them according to the sensor specifications by entering the values as an array in the "Value" field.

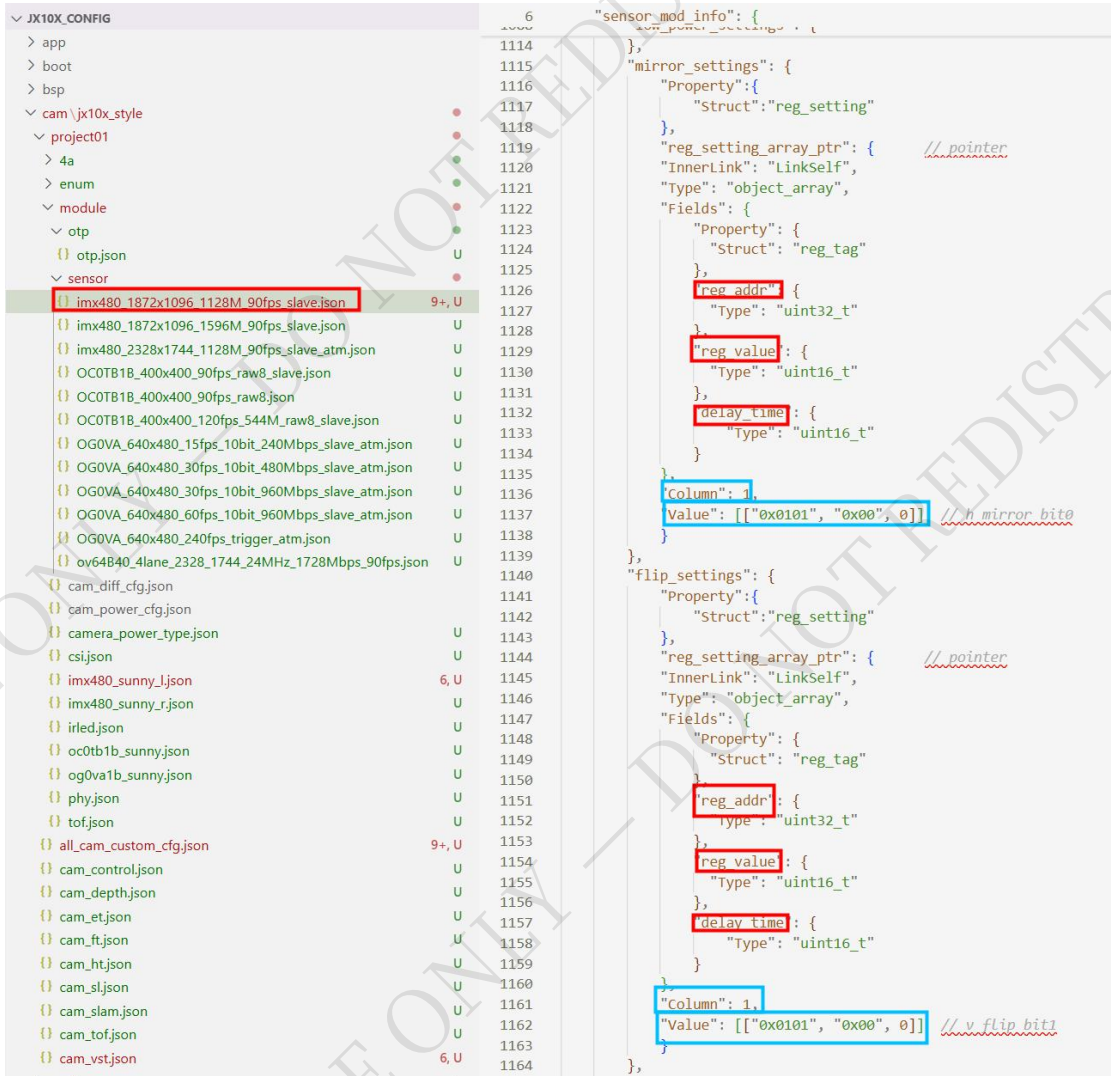


Figure 4.1.5.11 Mirror Flip Function Configuration

4.1.6 Configuration— Sensor Power Related Information

As shown in Figure 4.1.6.1, modify the `cam\_power\_cfg` file to configure the settings as an array; the parameter ` "Column" ` indicates that up to 15 groups are supported, arranged in the order of the physical cameras; the file path is located in the ` "module/" ` subfolder. Users should pay particular attention to the parameters highlighted in the red box; the parameters highlighted in the blue box are those that users need to customize and configure manually.

Parameter	Meaning	Explain	Remarks
cam_id	Physical Camera – The system's unique identifier	The user needs to find the number co-responding to the physical ID.	Generally, no configuration is required.

	(Enumeration), corresponding to the list in Figure 3.3.1.4	Group Parameters	
cam_mclk	This camera is used. MCLK (Main Clock)	Users can Under "Options"> Path Identify the configuration requirements. Configured camera power Type: Enumerated type – Details See Figure 3.3.10 for details.	Generally, no configuration is required.
mclk_type	mclk power supply hardware configuration	0: GPIO power supply; 1: PMIC Power Supply	The X100's side is powered by GPIO; typically, no additional configuration is required. Place
mclk_id	mclk – specific hardware identifier		"Options" link GPIO Pin Related Enumerations Information; Select PMIC Enter the specific PMIC. LOD Number, "Options" Chain Receive PMIC-related enumeration messages Breath.
mclk_gpio_func	Is the MCLK pin used as a GPIO function? Make use of	0: other function; 1: gpio function	
mclk_gpio_direction	If the GPIO functionality is to be used, configure this GPIO pin. Foot direction	0: Input; 1: Output	MCLK requires only an input clock signal; therefore, its direction is Import
mclk_gpio_func_num	The same GPIO can have multiple functions.		Use X100 with Function 0

	Number: Indicates the one currently in use. function- num		
mclk_func_param1	Reserved field – currently unused		Future expansion reservation

"cam_id": physical camera id。

"cam_mclk": The MCLK for the sensor. Users can locate the required camera power type enum value under the "Path" section in "Options"; the "Value" field specifies the MCLK power type. The user may choose not to configure this parameter and can instead use the default configuration.

"mclk_type": The hardware power supply configuration for mclk (0: GPIO power supply; 1: PMIC power supply); the X100 side uses the GPIO power supply.

"mclk_id": The specific power supply configuration for mclk. If you select "GPIO power supply," enter the specific GPIO pin number; the "Options" section will display the relevant enumeration information for that GPIO pin. If you select "PMIC," enter the specific PMIC LOD number; the "Options" section will display the relevant enumeration information for that PMIC.

"mclk_gpio_func": Whether the pin used for mclk is configured as a GPIO function (0: other function; 1: GPIO function); on the X100 side, this corresponds to a GPIO function.

"mclk_gpio_direction": The GPIO direction selection used for mclk (0: Input; 1: Output); the X100 side is configured as an input.

"mclk_gpio_func_num": The GPIO supports multiple functions; this value indicates the function number used for mclk. On the X100 side, the GPIO function corresponds to function 0.

"mclk_func_param1": Reserved parameter – currently not in use.

"mclk_func_param2": Reserved parameter – currently not in use.

"mclk_func_param3": Reserved parameter – currently not in use.

Other power name configurations are similar; no further elaboration is needed here.


```
* @return sensor_handle_t , sensor handle
*/

sensor_handle_t cam_md1_open(struct cam_md1_driver_info_t *cam_ctx);

/**
 * @brief camera module init
 * @param[in] handle, sensor handle
 * @return jx_err_t, Error code indicating the result of the operation
 */
jx_err_t cam_md1_init(sensor_handle_t handle);

/**
 * @brief set camera mode
 * @param[in] handle, sensor handle
 * @param[in] mode, sensor mode
 * @return jx_err_t, Error code indicating the result of the operation
 */
jx_err_t cam_md1_set_mode(sensor_handle_t handle, jx_u32_t mode);

/**
 * @brief set camera mode
 * @param[in] handle, sensor handle
 * @param[in] info, sensor stream on info
 * @return jx_err_t, Error code indicating the result of the operation
 */
jx_err_t cam_md1_pre_stream_on(sensor_handle_t handle, struct cam_md1_stream_on_info_t
*info);

/**
 * @brief camera module stream on
 * @param[in] handle, sensor handle
 * @return jx_err_t, Error code indicating the result of the operation
```

```
*/  
jx_err_t cam_md1_stream_on(sensor_handle_t handle);  
  
/**  
 * @brief camera module stream off  
 * @param[in] handle, sensor handle  
 * @param[in] off_mode, sensor stream off mode  
 * @return jx_err_t, Error code indicating the result of the operation  
 */  
jx_err_t cam_md1_stream_off(sensor_handle_t handle, jx_u32_t off_mode);  
  
/**  
 * @brief camera module close  
 * @param[in] handle, sensor handle  
 * @return jx_err_t, Error code indicating the result of the operation  
 */  
jx_err_t cam_md1_close(sensor_handle_t handle);  
  
/**  
 * @brief camera module probe  
 * @param[in] cam_ctx, cam info from json config  
 * @return jx_err_t, Error code indicating the result of the operation  
 */  
jx_err_t cam_md1_probe(struct cam_md1_driver_info_t *cam_ctx);
```

4.3 Sensor OPT Interface

OTP identifies the corresponding EEPROM interface by matching the `eeprom\_name` field in the OTP JSON payload with the `eeprom\_name` field in the corresponding EEPROM interface configuration file.

The Corresponding Description Is Shown in the Figure Below:

```
"Property": {  
    "Name": "eeprom",  
    "Version": "0.0.0.0"  
},  
"eeprom_info": {  
    "eeprom_usable": {  
        "Type": "bool",  
        "Value": true  
    },  
    "eeprom_name": {  
        "Type": "string",  
        "Length": 32,  
        "Value": "P24C64F-A4H-MIR"  
    }  
},
```

```
struct eeprom_ctrl_t eeprom_ctrl_1 = {  
    "P24C64F-A4H-MIR",  
    {  
        .module_info = p24c_get_module_info,  
        .awb_info     = p24c_get_awb_info,  
        .lsc_info      = p24c_get_lsc_info,  
        .otp           = p24c_get_otp,  
    }  
};
```

The Main Interfaces Include:

/**

* @brief get sensor otp data

```
* @param[in] buf is address to save otp data
* @param[in] eeprom_info_ptr is eeprom info from json config
* @return jx_err_t Error code indicating the result of the operation
*/
jx_err_t sensor_get_otp(jx_void_t *buf, struct eeprom_info_t *eeprom_info_ptr);

/**
* @brief get sensor otp data
* @param[in] cmd is eeprom_info_cmd, awb or lsc or module info.
* @param[in] eeprom_info_ptr is eeprom info from json config
* @param[in] buf is address to save otp data
* @return jx_err_t Error code indicating the result of the operation
*/
jx_err_t sensor_get_otp_info(jx_u32_t cmd, struct eeprom_info_t *eeprom_info_ptr, jx_void_t
*buf);
```

4.4 Irled Control Interface

Irled currently offers the following three control methods:

1. Sensor strobe directly controls the IR LED: The sensor strobe output signal is connected directly to the IR LED via hardware.

The Control Sensor Strobe register is used to control the IRLED; 2. The chip incorporates built-in external FSync and Strobe signal control functionality, which is implemented via the chip's Strobe output signal.

To control an IR LED directly, the pin connected to the IR LED must be configured as a strobe function; 3. To control an IR LED via GPIO, the pin connected to the IR LED must be configured as a GPIO function.

The IRLED DRV interface is used to obtain control methods and manage corresponding configurations; the primary interfaces include `irled_probe`, `irled_open`, `irled_get_cfg`, `irled_ctrl`, and `irled_close`.

Method 1 requires a direct connection between the sensor strobe and the IR LED hardware; the IR LED can be configured for constant-on, constant-off, or fine-grained per-frame control using control registers. The primary interface is ``cam_md1_set_strobe_io_direction``.

cam_md1_set_strobe_out_method , cam_md1_set_strobe_out_regctl ,
cam_md1_set_strobe_out_frameshift, cam_md1_set_strobe_out_framespan。

Method 2 requires the chip to have an integrated sensor with an external strobe output pin connected directly to the IRLED hardware; the `function` attribute of this pin must be configured as the "strobe output" function. The hardware directly controls the duration and timing of IRLED illumination for each frame, without requiring any interface intervention.

Method 3 follows the same connection scheme as Method 2; however, Method 3 introduces an additional interface to address the limitation of Method 2 in controlling the IR LED to remain permanently on or off. The core principle involves switching the pin to a GPIO function to disable the chip's built-in strobe signal, then configuring the pin as a GPIO and using the `irled\_ctrl` interface to drive the GPIO pin high or low to control the IR LED's persistent on or off state.

The main interfaces are as follows:

```
/**
 * @brief probe of irled drv
 * @param[in] logic_id logic camera id
 * @param[in] ptr cam mdl drv info, include camera information
 * @return irled_handle_t irled handle, include irled information
 */
irled_handle_t irled_probe(jx_u8_t logic_id, struct cam_md1_driver_info_t *ptr);

/**
 * @brief irled drv open
 * @param[in] logic_id logic camera id
 * @param[in] ptr irled handle
 * @return void
 */
jx_void_t irled_open(jx_u8_t logic_id, irled_handle_t handle);

/**
 * @brief get irled drv handle
 * @param[in] logic_id logic camera id
 * @param[in] phys_index physical camera index in each logic seriels
 * @return irled_handle_t irled handle, include irled information
```

```
*/  
irled_handle_t irled_get_cfg(jx_u8_t logic_id, jx_u8_t phys_index);  
  
/**  
 * @brief irled drv control  
 * @param[in] logic_id logic camera id  
 * @param[in] phys_index physical camera index in each logic seriels  
 * @param[in] ptr irled control parameter  
 * @return jx_err_t Error code indicating the result of the operation  
 */  
jx_err_t irled_ctrl(jx_u8_t logic_id, jx_u8_t phys_index, jx_void_t *ptr);  
  
/**  
 * @brief irled drv close  
 * @param[in] logic_id logic camera id  
 * @param[in] phys_index physical camera index in each logic seriels  
 * @return jx_err_t Error code indicating the result of the operation  
 */  
jx_err_t irled_close(jx_u8_t logic_id, jx_u8_t phys_index);  
  
/**  
 * @brief camera module set strobe io direction  
 * @param[in] handle, sensor handle  
 * @param[in] reg_addr, strobe io direction register addr  
 * @param[in] bit_shift, strobe io direction register valid bit start shift  
 * @param[in] bit_width, strobe io direction register valid bit width  
 * @param[in] value, strobe io direction register value,input or output  
 * @return jx_err_t, Error code indicating the result of the operation  
 */  
jx_err_t cam_md1_set_strobe_io_direction(sensor_handle_t handle, jx_u32_t reg_addr,  
jx_u32_t bit_shift, jx_u32_t bit_width,  
jx_u32_t value);
```



```

/**
 * @brief camera module set strobe out method
 * @param[in] handle,    sensor handle
 * @param[in] reg_addr,  strobe out method register addr
 * @param[in] bit_shift, strobe out method register valid bit start shift
 * @param[in] bit_width, strobe out method register valid bit width
 * @param[in] value,     strobe out method register value, register ctrl or shift span ctrl
 * @return jx_err_t,     Error code indicating the result of the operation
 */
jx_err_t cam_mdl_set_strobe_out_method(sensor_handle_t handle, jx_u32_t reg_addr,
jx_u32_t bit_shift, jx_u32_t bit_width,
                                jx_u32_t value);

/**
 * @brief camera module set strobe out register ctrl open or close
 * @param[in] handle,    sensor handle
 * @param[in] reg_addr,  strobe out register ctrl register addr
 * @param[in] bit_shift, strobe out register ctrl register valid bit start shift
 * @param[in] bit_width, strobe out register ctrl register valid bit width
 * @param[in] value,     strobe out register ctrl register value, ctrl open or close
 * @return jx_err_t,     Error code indicating the result of the operation
 */
jx_err_t cam_mdl_set_strobe_out_regctl(sensor_handle_t handle, jx_u32_t reg_addr, jx_u32_t
bit_shift, jx_u32_t bit_width,
                                jx_u32_t value);

/**
 * @brief camera module set strobe out frameshift
 * @param[in] handle,    sensor handle
 * @param[in] p_default, strobe out frameshift register default value, get from irled_get_cfg

```

* @param[in] custom_value, strobe out frameshift custom value, if not zero, p_default will not use

* @return jx_err_t, Error code indicating the result of the operation

*/

jx_err_t cam_md1_set_strobe_out_frameshift(sensor_handle_t handle, sensor_strobe_shift_t *p_default, jx_u64_t custom_value);

/**

* @brief camera module set strobe out_framespan

* @param[in] handle, sensor handle

* @param[in] p_default, strobe out framespan register default value, get from irled_get_cfg

* @param[in] custom_value, strobe out framespan custom value, if not zero, p_default will not use

* @return jx_err_t, Error code indicating the result of the operation

*/

jx_err_t cam_md1_set_strobe_out_framespan(sensor_handle_t handle, sensor_strobe_span_t *p_default, jx_u64_t custom_value);

4.5 2A Algorithm Interface

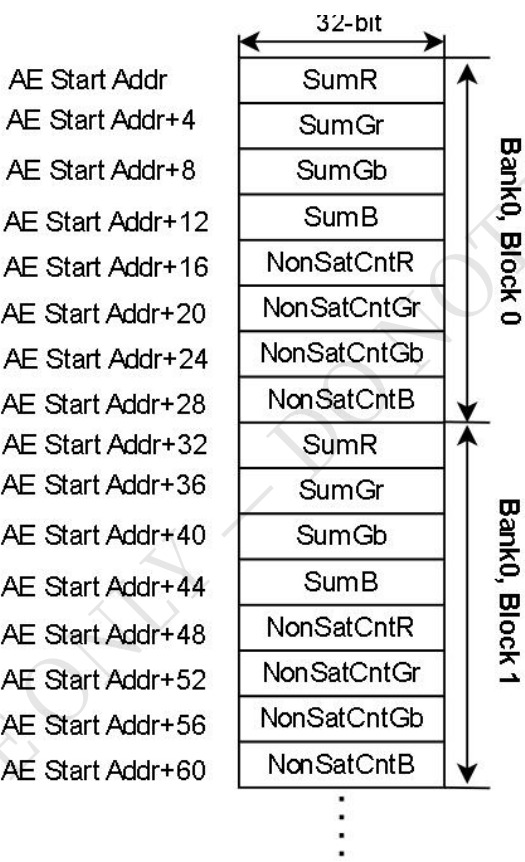
4.5.1 Statistical Format

1. info head

Each statistical record contains a 128-byte Info Head. The information contained in the Info Head – such as block/bank details – is transmitted by the software to the Algorithm Wrapper layer; therefore, the algorithm does not need to parse its contents.

2. AE Statistics

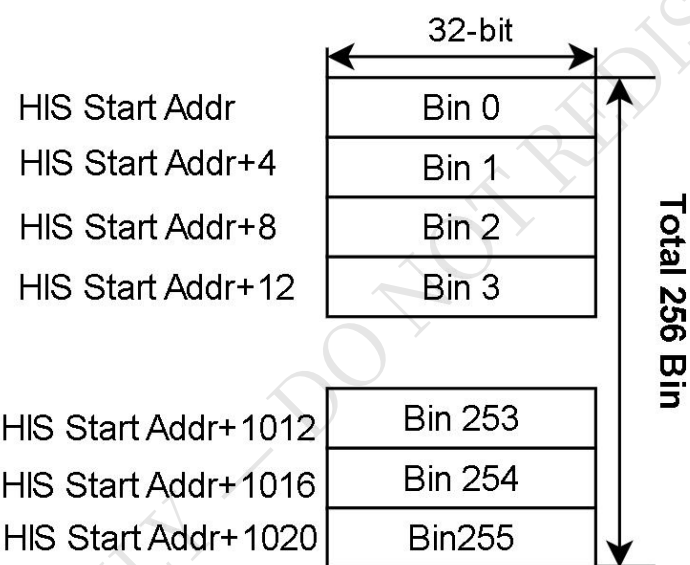
AE Statistics divides the image into 'block * bank' regions. The format is as follows:



Field	Explain
SumR	Total sum of unsaturated R pixel values within the region
SumGr	Total sum of unsaturated Gr pixel values within the region
SumGb	Sum of unsaturated Gb pixel values within the region
SumB	Sum of unsaturated B pixel values within the region
NonSatCntR	Number of unsaturated R pixels in the region
NonSatCntGr	Number of unsaturated Gr pixels in the region
NonSatCntGb	Number of unsaturated Gb pixels in the region
NonSatCntB	Number of unsaturated B pixels in the region

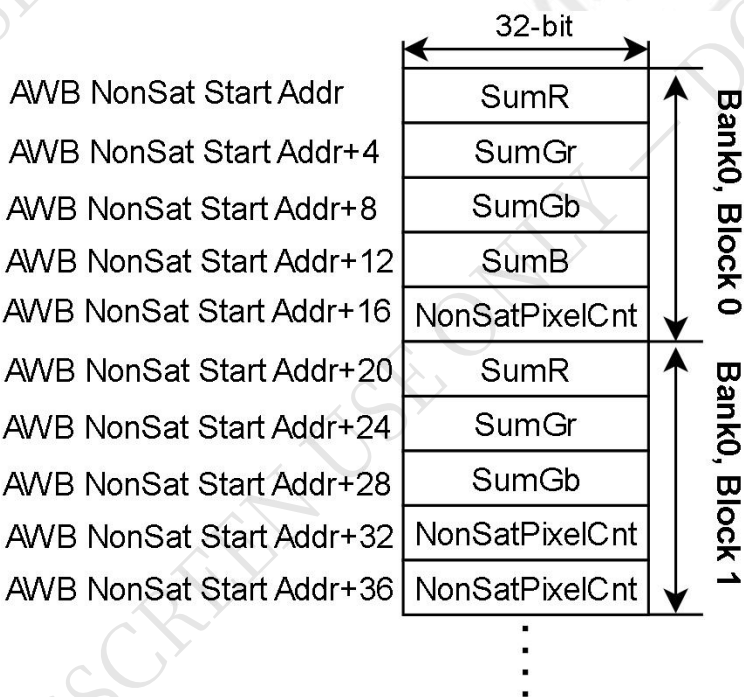
3. Histogram Statistics

hist divides the pixel values into 256 levels ([0~255]). Each level represents the pixel count in units of `uint32\_t`.



4. AWB Statistics

AWB Statistical Method Divides the Image Into 'block * Bank' Regions. The Format Is as Follows:



Field	Explain
SumR	Total Sum of Unsaturated R Pixel Values Within the Region
SumGr	Total sum of unsaturated Gr pixel values within the region
SumGb	Sum of unsaturated Gb pixel values within the region
SumB	Sum of unsaturated B pixel values within the region

NonSatCntPixelR	The number of unsaturated pixels within the region; if any single component reaches saturation, the pixel is considered saturated.
-----------------	--

- ◆ For the Mono Pipe AE, the statistics are calculated first, followed by the Info Head; for the Scene Pipe, the Info Head is calculated first, followed by the statistics;
- ◆ The Mono Pipe AE and Scene Pipe AE share the same statistical format, but the Scene Pipe AE does not include the yhist parameter.

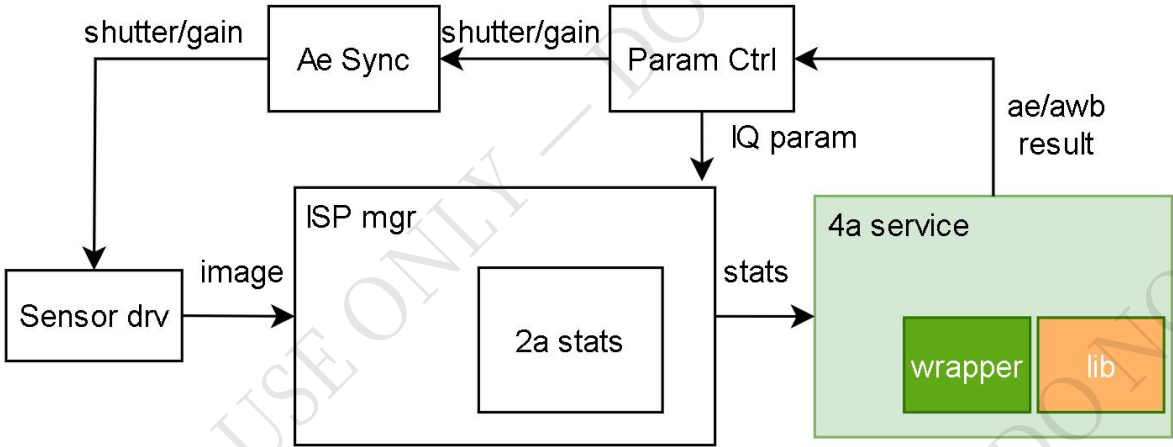
4.5.2 Code Directory Structure

```
Bu – include                                #Public Header File Directory
|   Bu—— isp hardware_ae_api.h
|   Bu—— isp hardware_api.h
|   Bu—— isp hardware_atf_api.h
|   Bu—— isp hardware_awb_api.h
|   └── SConscript
Bo——lib                                    #Dependent library header files
|   Bu—— libAE_cortex-r8.a
|   Bu—— libALSC_cortex-r8.a
|   Bu—— libATF_cortex-r8.a
|   Bu—— libAWB_cortex-r8.a
|   └── SConscript
Bu – SConscript
└── wrapper                                #Algorithm Integration Catalog
    Bu – a4_out_api.h
    Bu – cam_4a_awb_wrapper.c
    Bu – cam_4a_bayer_ae_wrapper.c
    Bu – cam_4a_common.h
    Bu – cam_4a_core.h
    Bu – cam_4a_mono_ae_wrapper.c
    Bu – cam_4a_mono_mixed_expo.h
    Bu – cam_4a_wrapper_util.h
    └── SConscript
```

Document	Description
isp_hardware_ae_api.h	AE Algorithm Input Parameter Definition
isp_hardware_api.h	Algorithm API definition includes init/process/deinit
isp_hardware_atf_api.h	ATF Algorithm Input Parameter Definition
isp_hardware_awb_api.h	AWB Algorithm Input Parameter Definition
libAE_cortex-r8.a	aealgorithms library
libAWB_cortex-r8.a	awbalgorithms library
libALSC_cortex-r8.a	alscalgorithms library
libATF_cortex-r8.a	atfalgorithms library
a4_out_api.h	4a Output Structure Definition
cam_4a_awb_wrapper.c	AWB Algorithm Integration
cam_4a_bayer_ae_wrapper.c	VST AE Algorithm Integration
cam_4a_common.h	4a Service Module: Data definition for interaction with other modules. Includes initialization parameters, algorithm type, and algorithm results.

cam_4a_core.h	4a Service Module Definition
cam_4a_mono_ae_wrapper.c	Mono AE Algorithm Integration
cam_4a_mono_mixed_expo.h	SLAM Exposure Alignment Interface
cam_4a_wrapper_util.h	Print the log buffer output by the algorithm

4.5.3 Control Flow



ParamCtrl: Pipe Tuning – parameter interpolation, which ensures that the shutter/gain and pipe parameters take effect simultaneously;

Ae Sync (Open Source): Exposure Center synchronization control; enables synchronization of shutter/gain and frame duration;

Sensor DRV (Open Source): Sends shutter speed, gain, and frame duration to the sensor.

4.5.4 Wrapper Implementation and Parameter Overview

wrapperAPI	Implementation step
wrapper_init	Prepare the parameters and call `algo->init()`

	Configure dynamic parameter dependency
wrapper_process	Prepare the parameters and call `algo->process()`
	Print Algorithm Library – log
wrapper_dync_cfg	Response to Dynamic Configuration
wrapper_deinit	Call `algo->deinit()` to free memory.

Algo-rithm	Paramete r-based Mould	Parameter
AE	Static state (init)	TuningBin/Sensor – Dimensions/Sensor Initial Expo- sure/Row Exposure/Aperture/Statistical Grid
	Trends (process)	Statistics / Corresponding exposure configuration / IMU / Anti-flick results / AWB results
	Algorithm Results	shutter/sensor-gain/isp-gain/bv/mid-gain/shadow-gain
	Dynamic allocation (dync_cfg)	Auto/Manual Switch/Exposure Pre-Activation Confi- guration
AWB	Static state (init)	TuningBin/Sensor – Dimension/Statistical Grid

Trends (process)	Statistics/AE Results
Algorithm Results	AWB Gain/Color Temperature/CCM
Dynamic allocation (dync_cfg)	Auto-Manual Switch

Bayer AE Wrapper – Initialization

```
jx_err_t bayer_ae_wrapper_init(jx_handle_t handle, struct  
a4_core_wrapper_init_params_t *init);
```

parameter declaration:

- 'handle': context handle
- 'init': Initialization parameters. Includes tuningbin/sensor dimensions, sensor initial exposure, line exposure, aperture, and statistical grid.

returned value:

- '0': Success
- Non-zero: Failure (returns errno)

Bayer AE Wrapper Algorithm Calculation

```
jx_err_t bayer_ae_wrapper_proc(jx_handle_t handle, struct cam_4a_rlt_t  
*proc_rlt);
```

parameter declaration:

- `handle`: context handle
- `proc\_rlt`: Algorithm calculation results, including shutter/sensor-gain/ISP-gain/BV/Midgain/Shadow-gain

Additionally, the `dependency` field in the `context` has been updated to reflect the parameters required by the algorithm—including AE statistical data, corresponding exposure configuration, IMU/anti-flick results, and AWB results.

returned value:

- `0`: Success
- Non-zero: Failure (returns errno)

Bayer AE wrapper deinitialization

jx_err_t bayer_ae_wrapper_deinit(jx_handle_t handle);

parameter declaration:

- `handle`: context handle

returned value:

- `0`: Success
- Non-zero: Failure (returns errno)

**jx_err_t bayer_ae_dync_cfg(jx_handle_t handle, struct
cam_4a_cfg_params_t *cfg);**

parameter declaration:

- `handle`: context handle
- `cfg`: Dynamic Configuration – Includes auto-manual switching/曝光预生效配置

returned value:

- `0`: Success
- Non-zero: Failure (returns errno)

AWB wrapper initialization

**jx_err_t awb_wrapper_init(jx_handle_t handle, struct
a4_core_wrapper_init_params_t *init);**

parameter declaration:

- `handle`: context handle
- `init`: Initialization parameters. Includes tuningbin/sensor size/statistical grid. Return value:

- `0`: Success
- Non-zero: Failure (returns errno)

AWB wrapper algorithm calculation

```
jx_err_t awb_wrapper_proc(jx_handle_t handle, struct cam_4a_rlt_t  
*proc_rlt);
```

parameter declaration:

- `handle`: context handle
- `proc\_rlt`: Algorithm calculation results, including awbgain/Color Temperature/CCM

Additionally, the `dependency` field in the context has been updated to reflect the parameters of the algorithmic dependency; this affects AWB statistics/AE results.

returned value:

- `0`: Success
- Non-zero: Failure (returns errno)

AWB wrapper deinitialization

```
jx_err_t awb_wrapper_deinit(jx_handle_t handle);
```

parameter declaration:

- `handle`: context handle

returned value:

- `0`: Success

- Non-zero: Failure (returns errno)

```
jx_err_t awb_dync_cfg(jx_handle_t handle, struct cam_4a_cfg_params_t  
*cfg);
```

parameter declaration:

- `handle`: context handle
- `cfg`: Dynamic configuration – includes auto-manual switching.

returned value:

- `0`: Success
- Non-zero: Failure (returns errno)

4.6 SLAM Exposure Center Synchronous Control

The synchronization method can be configured in the Sensor JSON configuration.

`SYNC_MODE_NONE = 0, // Hard synchronization for SOF alignment, but without exposure center synchronization`

`SYNC_MODE_SW = 1, //Soft synchronization`

`SYNC_MODE_HW = 2, // Hard synchronization; the sensor uses the low-power wake-up mode.`

Not supported yet

`SYNC_MODE_AE_ALIGN=3, // Individual exposure, then align shutter, adjust AD gain` The synchronous control method is implemented in `application/api/cam\_ae\_sync.c`. Each Logic Cam instance has an AE Sync object. Only SLAM requires exposure center synchronization.

AE Sync Initialization

jx_err_t cas_init(jx_handle_t *handle, struct cas_init_info_t *init);

Parameter Declaration:

- 'handle': context handle
- 'init': Initialization Parameters. Includes:
 - Logic ID: Logic CAMID
 - phys_num: Number of physical cameras
 - start_id: The smallest camera ID value
 - sync_mode: AE synchronization mode
 - fps: frame rate
 - csc_hdl: cam session control handle
 - cb: callback to cam session control
 - cas_share_info: Share information for SLAM/HT; allows viewing the mutual dynamic frame length configuration return value:
- '0': Success
- Non-zero: Failure (returns errno)

AE Sync Deinitialization

jx_err_t cas_deinit(jx_handle_t handle);

parameter declaration:

- 'handle': context handle
- Return value: None

AE Sync Event Handling

**jx_err_t cas_process(jx_handle_t handle, jx_u32_t cmd, jx_u32_t id,
jx_void_t *param);**

parameter declaration:

- 'handle': context handle
- 'cmd': Command
 - CAS_SOF: sof event

CAS_RAW_AE: The AE result computed by the AE algorithm, used in the SYNC_MODE_AE_ALIGN synchronization mode.

- `id`: cam id
 - `param`: Parameter Pointer
- returned value:
- `0`: Success
 - Non-zero: Failure (returns errno)

Safevent Handling

jx_err_t cas_sof_event(struct cas_ctx_t *ctx, jx_u32_t id, jx_u32_t frm_id);

parameter declaration:

- `ctx`: context handle
 - `id`: cam id
 - `frm\_id`: Frame Number
- returned value:
- `0`: Success
 - Non-zero: Failure (returns errno)

AE effective frame configuration

jx_err_t jx_err_t cas_eff_ae_process(struct cas_ctx_t *ctx, jx_u32_t id, jx_void_t *eff_rlt);

parameter declaration:

- `ctx`: context handle
 - `id`: cam id
 - `eff\_rlt`: AE Result + Effective Frame (frm_id)
- returned value:
- `0`: Success
 - Non-zero: Failure (returns errno)

Align and expose different cameras under the same Logic Cam; adjust sensor gain (used in SYNC_MODE_AE_ALIGN synchronization mode).

jx_err_t cas_adjust_ae_exp(struct cas_ctx_t *ctx, jx_u32_t id, jx_void_t *param);

parameter declaration:

- `ctx`: context handle
 - `id`: cam id
 - `param`: Parameter pointer. AE result
- returned value:
- `0`: Success
 - Non-zero: Failure (returns errno)

Soft synchronization; calculate frame length

jx_void_t cas_get_vts(struct cas_ctx_t *ctx, jx_u32_t phys_id, jx_u32_t frm_idx);

parameter declaration:

- `ctx`: context handle
 - `id`: cam id
 - `frm\_idx`: Frame Number Index
- returned value:
- `0`: Success
 - Non-zero: Failure (returns errno)

Soft synchronization; calculate frame length

jx_err_t cas_set_vts(struct cas_ctx_t *ctx, jx_u32_t id, jx_u32_t frm_id);

parameter declaration:

- `ctx`: context handle
- `id`: cam id

- `frm\_id`: Frame number return value:
- `0`: Success
- Non-zero: Failure (returns errno)

5. CV/DSP (DSP,@Zhou Zheng)

5.1 Various Algorithm Interfaces

5.2 DSP and R8 Data Interfaces

6. SensorecHub

6.1 Summary

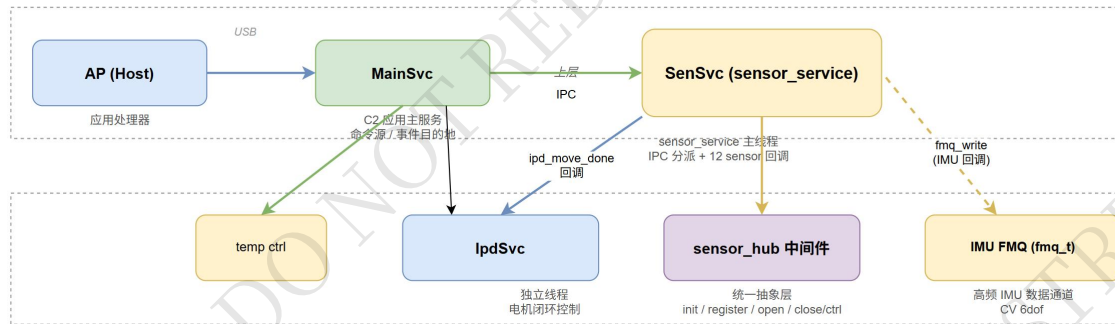
The Sensor Subsystem is a service within the system firmware responsible for the **sensor control plane**. It runs on the X100 C2/C3 cluster and collaborates with the Sensor Hub middleware and MainSvc to perform the following tasks:

- Register/Enable/Disable sensor
- Manage sensor drivers and devices
- Process sensor control commands issued by the AP.
- Receive event callbacks from sensor_hub and forward them to MainSvc/AP
- Maintain IPD (Eye Distance Adjustment) closed-loop motor control
- Temperature control strategy, fan control

Dependent Lower-Level Modules:

- sensor_hub_api.h —Sensor Middleware API
- sensor_custom.h —Product-level sensor configuration definition
- jxapi/jx_api_sensor_msg.h —Sensor IPC Message Definition

6.2 System Location



sensor子系统架构图

SenSvc Peer:

- **Upper Layer:** MainSvc (Command Source, Event Destination), AP (Direct Pass-through)
- **Lower:** SensorHub Middleware
- **bypass:**
 - IPD – Independent Thread; sensor_flow – Callback Registration Party
 - Temperature control strategy, Temperature Control System

6.3 IPC Interface

6.3.1 Command Dispatch Table

```
static service_cmd_cb_t s_sensor_cmd_cb[] = {
    {IPC_GLB_SYS,      IPC_SYS_STATE_REQ,
    sensor_service_get_state},
    {IPC_GLB_SYS,      IPC_SYS_VERSION_REQ,
    sensor_service_get_version},
    {IPC_GLB_SENSOR,   IPC_SENSOR_INIT,
    sensor_service_init},
    {IPC_GLB_SENSOR,   IPC_SENSOR_START,
    sensor_service_start},
    {IPC_GLB_SENSOR,   IPC_SENSOR_STOP,
    sensor_service_stop},
    {IPC_GLB_SENSOR,   IPC_SENSOR_CFG,
    sensor_service_cfg},
    {IPC_GLB_SENSOR,   IPC_SENSOR_OPEN,
    sensor_service_undefine_cmd},
    {IPC_GLB_SENSOR,   IPC_SENSOR_CONTROL_REQ,
    sensor_service_ctrl},
```

```
{IPC_GLB_SENSOR_DBG, IPC_SENSOR_DBG_CONTROL_REQ,
sensor_service_ctrl},
{IPC_GLB_SENSOR_DBG, IPC_SENSOR_DBG_START_REQ,
sensor_service_undefine_cmd},
{IPC_GLB_INTERNAL, IPC_INTERNAL_HEART_BEAT_REQ,
sensor_service_heart_beat_req},
};
```

6.3.2 Primary Command ID

Order	Meaning
IPC_SENSOR_CFG	Load sensor_custom + Initialize Hub
IPC_SENSOR_CFG_OK	CFG Receipt (including Sanity Result Bitmap)
IPC_SENSOR_INIT / INIT_OK	Single-type initialization query
IPC_SENSOR_START / START_OK	Start specified type
IPC_SENSOR_STOP	Stop
IPC_SENSOR_CONTROL_REQ	Control command (CALI/FAN/LED/IPD...)
IPC_SENSOR_CONTROL_RESP	Control Receipt
IPC_SENSOR_DATA_REPORT	Sensor Event Reporting
IPC_REPLY_ERRCODE	Error Receipt
IPC_INTERNAL_HEART_BEAT_REQ/RESP – Heartbeat	Heartbeat (Synchronous Frame ID)

6.3.3 Error Code

Error Code Format

```
31      | 30...28      | 27...22      | 21...16      | 15...0
reserve| err level    | subsys ID     | module ID    | err code
```

Three module IDs:

```
typedef enum {
    ERR_SENSOR_HUB = 0x01; // Middleware layer
    ERR_SENSOR_DRV = 0x02; // Driver layer
    ERR_SENSOR_SVC = 0x03; // Service layer
}
```

```
} jx_err_sensor_hub_module_t;
```

Common Error Codes:

Macro	Mod- ule	Wrong	Meaning
ERR_HUB_OPEN_FAIL	HUB	0x0001	Sensor 'open' failure
ERR_HUB_CLOSE_FAIL	HUB	0x0002	closebe defeated
ERR_HUB_CTL_FAIL	HUB	0x0003	Control failure
ERR_HUB_NOT_REGISTERED	HUB	0x0004	Sensor not registered
ERR_HUB_ALREADY_STARTED	HUB	0x0005	Started
ERR_HUB_INVALID_PARAM	HUB	0x0006	Parameter error
ERR_HUB_NO_LISTENER	HUB	0x0007	Not havereader
ERR_HUB_INIT_FAIL	HUB	0x0008	initbe defeated
ERR_HUB_NO_MATCH	HUB	0x0009	No matching driver
ERR_HUB_ALLOC_FAIL	HUB	0x000A	Memory failure
ERR_DRV_BUS_REQUEST_FAIL	DRV	0x0003	Bus request failed
ERR_DRV_XFER_FAIL	DRV	0x0004	Bus transmission failed
ERR_DRV_READ/WRITE_TIMEOUT	DRV	0x0005/6	Read/Write timeout
ERR_DRV_PROBE_FAIL	DRV	0x000A	probebe defeated
ERR_DRV_CHECK_ID_FAIL	DRV	0x000B	Chip ID verification failed
ERR_SVC_INVALID_PARAM	SVC	0x0001	Parameter error
ERR_SVC_ALREADY_STARTED	SVC	0x0002	Started
ERR_SVC_NOT_STARTED	SVC	0x0003	Not started
ERR_SVC_NO_SENSOR	SVC	0x0004	Sensor not found
ERR_SVC_ALLOC_FAIL	SVC	0x0005	Memory failure
ERR_SVC_SEND_FAIL	SVC	0x0006	IPC send failed
ERR_SVC_MSG_TOO_LONG	SVC	0x0007	Message is too long
ERR_SVC_CMD_NOT_FOUND	SVC	0x0008	Unknown command
ERR_SVC_CALI_INVALID	SVC	0x000A	Calibration invalid
ERR_SVC_OUT_OF_RANGE	SVC	0x000B	Out-of-range

Construct Macro:

SNS_ERR_HUB (level, err) // Middleware error; SNS_ERR_DRV (level, err) // Driver error; SNS_ERR_SVC (level, err) // Service error

6.3.4 CTRL Command Detailed Guide

sensor_service_ctrl assigns based on cmd_id:

cmd_id	Handle	Remarks
SENSOR_CMD_CALI_START	Invoke SNS_CTL_CALI_START for all handles of this type.	No sensor of this type found – return immediately.
SENSOR_CMD_CALI_STOP	Ditto	
SENSOR_CMD_FAN_SET	Match by ID; SNS_CTL_FAN_SET – Set duty cycle	
SENSOR_CMD_LED_SET	Match by ID; SNS_CTL_COMMON – set mode/RGB/Breathing	
SENSOR_CMD_KEY_TICK_CFG	Short/Long Press/Hold to resend ticks	
SENSOR_CMD_LOW_POWER	Set TEMP to SNS_CTL_LOW_POWER	
SENSOR_CMD_IPD_MOVE	ipd_service_move	
SENSOR_CMD_IPD_GET	ipd_service_get – Synchronous read	
Other	undefine	

After completing CTRL, construct the IPC_SENSOR_CONTROL_RESP response(result + out_para).

6. 4 Sensor_hub Key API

sensor_service decouples itself from the underlying sensor drivers via the SensorHub middleware. This chapter provides a detailed description of the key APIs, data structures, and error codes offered by SensorHub.

6.4.1 Core API Overview

Group by Invocation Phase:

life cycle:

```
jx_err_t sensor_hub_init(void *module_info);           // Initialization (CFG Phase)
jx_err_t sensor_hub_deinit(void);                     // Deinitialization
```

Device Registration(CFG phase; each sensor should perform this once):

```
void sensor_hub_register_dev(
    sensor_custom_hw_info_t *info; // Hardware information (bus/interrupt/power/sub-device)
    sensor_type_t type – Sensor type
    void *fac_cali, // 产线标定数据指针
    uint16_t cali_len – // Calibration data length
    void *user_data // 用户自定义数据
);
```

Equipment Query:

```
jx_s8_t sensor_hub_get_sensorlist(
    struct sensor_hub_info *list, jx_u8_t size,
    sensor_type_t type); // Query by type and return matching results
number
jx_bool_t sensor_hub_has_type(sensor_type_t type); // Does this type exist?
sensor_dev_t *sensor_hub_get_dev(jx_s32_t sd); // Retrieve the device object using 'sd'
jx_list_t *sensor_hub_get_hw_info_list(void); // 获取所有已注册硬
Item Information
```

Data Path(START Phase):

```
jx_void_t *sensor_hub_open(jx_s32_t sd, jx_float_t rate, sensor_event_cb
    cb); // Open and start the data stream
jx_err_t sensor_hub_close(jx_void_t *handle); // Stop data stream
jx_err_t sensor_hub_read(jx_void_t *handle,
    sensor_event_t *events,
    jx_u64_t size); // Active reading (Actual CB Mode)
(Primary)
```

Control Pathway(CTRL Phase):

```
jx_err_t sensor_hub_set_param(jx_void_t *handle,
    jx_u8_t type, // sensor_ctl_cmd_t
    jx_void_t *arg); //controlling parameter
```



```
jx_err_t sensor_hub_ctrl_param_with_sd(uint32_t sd, jx_u8_t type, jx_void_t *arg); // Control logic that does not require handling
```

Synchronous Sampling(special Scenarios, E.g., IPD):

```
jx_err_t sensor_hub_sample(jx_void_t *handle, jx_void_t *data, jx_u64_t size); // Synchronous sampling based on handle
jx_err_t sensor_hub_sample_with_sd(uint32_t sd, jx_void_t *data, jx_u32_t size); // Synchronous sampling based on SD
```

Default Callback:

```
void sensor_hub_register_default_cb(sensor_event_cb cb); // 未指定 cb 的 sensor 走这里
```

6.4.2 sensor_event_t

The Core Data Structure for Event Callbacks, Followed Immediately by a Zero-Length Array Payload:

```
typedef struct {
    jx_u64_t      timestamp; // US timestamp
    uint8_t      report_type; // RAW_DATA / EVENT
    jx_u8_t      idx; // Index for sensors of the same type
    jx_u8_t      reserved[2];
    sensor_type_t type; // Source Type
    jx_u8_t      data[0]; // Payload (see table below)
} __attribute__((packed)) sensor_event_t;
```

`report\_type` Determines the Payload Meaning:

- SENSOR_REPORT_RAW_DATA → payload is sns_<type>_data_t
- SENSOR_REPORT_EVENT → payload is sns_<event>_t

6.4.3 Control Command (Sensor_ctl_cmd_t)

```
typedef enum {
    SNS_CTL_INIT, // Default initialization
    SNS_CTL_ENABLE, //enable
    SNS_CTL_DISABLE, //disability
    SNS_CTL_RANGE, // Set range
    SNS_CTL_ODR, // Set output rate
}
```

```

SNS_CTL_SAMPLE,           // Single sampling
SNS_CTL_BATCH,            // Batch sampling
SNS_CTL_FLUSH,            // Refresh FIFO
SNS_CTL_CALI_START,       // Begin calibration
SNS_CTL_CALI_STOP,        // Stop calibration
SNS_CTL_FACTORY,          // Production Line Testing
SNS_CTL_UPDATE_ATTR,      // updated attribute
SNS_CTL_COMMON,           // General (LED/MOTION multiplexing)
SNS_CTL_FAN_SET,          // Set fan duty cycle
SNS_CTL_IPD_MOVE,         // IPD Step
SNS_CTL_KEY_SHORT_TICKS,  // Short button press (tick)
SNS_CTL_KEY_LONG_TICKS,   // Long-press button trigger (tick)
SNS_CTL_KEY_HOLD_REPEAT_TICKS – Key Hold Repeat Tick
SNS_CTL_LOW_POWER,        // Low-power notification (arG: jx_bool_t *)} sensor_
ctl_cmd_t;

```

6.4.4 5.6 Hardware Registration Structure

`sensor\_hub\_reGister\_dev` receives `sensor\_custom\_hw\_info\_t` which is actually `sensor\_hw\_info\_t`:

```

typedef struct {
    jx_list_t      node;           // Hub linked list node
    sensor_pf_info_t plat_info;    // Platform Information (Bus/Interrupt/Power)
    sensor_module_t module;        // Module information (array of name/man-
    ufacturer/subdev)
} sensor_hw_info_t;

```

plat_info (sensor_pf_info_t) :

```

typedef struct {
    jx_u8_t  bus_type;           // I2C / SPI
    jx_u8_t  bus_id;             // Bus Number
    jx_u8_t  slave_addr;         // I2C slave address
    jx_u8_t  bus_dummy;
    jx_u8_t  i3c_addr;           // Reserved
    jx_u32_t min/max_bus_freq_khz;

    jx_u8_t  irq_pin;            // Interrupt Pin
    jx_u8_t  pull_type;

```

```
jx_u8_t trigger_type; // Edge/Level
jx_u8_t iomux_func;

`sensor_pwr_info_t pwr_info; // VDD/VDDIO configuration` `sensor_
pf_info_t`;
```

6.4.5 Driver Layer API (Drv_sensor.h)

Designed for Developers, With Seamless Integration at the Upper Layer:

```
sensor_t *sensor_allocate_device(void);
jx_err_t sensor_register(sensor_t *dev); // Register the device with the core: jx_
err_t sensor_free_device(sensor_t *dev);
jx_err_t sensor_driver_register(struct sensor_match_id *dev); // Register the driver; jx_list_t * sensor_get_
dev_list(jx_u8_t *nums);
jx_list_t *sensor_get_drv_list(jx_u8_t *nums);
extern void sensor_report_event(sensor_event_t *event); // Driver function for actively reporting events void
imu_CLK_CONFIG(uint32_t fs); // IMU Clock Configuration
void imu_clk_disable(void);
```

The driver must implement `sensor\_hw\_ops`:

```
struct sensor_hw_ops {
    jx_err_t (*open)(sensor_t *sensor, jx_u8_t flags);
    jx_err_t (*ctl)(sensor_t *sensor, sensor_ctl_cmd_t cmd, jx_void_t *arg);
    jx_err_t (*close)(sensor_t *sensor);
};
```

6.4.6 sensor_report_t

sensor_service_xxx_cb is the unified structure used for packaging sensor_event_t into IPC messages:

```
typedef struct {
    uint8_t report_type;
    uint8_t reserved[3];
    union {
        uint32_t event_type; // Event type
        uint32_t sensor_type; // Sensor type} type;
    uint64_t timestamp; // us
    uint32_t frame_id; //frame number
```

```
uint8_t data[0];          // payload
} __BYTE_PACKED__ sensor_report_t;
```

6.4.7 Typical Invocation Flow

```
// 1. CFG Phase: Register hardware (based on `sensor_custom_dev_t`)
sensor_hub_register_dev(&hw_info, SENSOR_IMU, fac_cali, cali_len, NULL);

// 2. Query available sensors
struct sensor_hub_info list[10];
int cnt = sensor_hub_get_sensorlist(list, 10, SENSOR_IMU);
int sd = list[0].sd; // 取第一个

// 3. START Phase: Open and initiate the data stream
void *handle = sensor_hub_open(sd, 100.0f, my_imu_cb); // 100Hz

// 4. Handling in the callback
void my_imu_cb(sensor_event_t *event) {
    sns_imu_data_t *p = (sns_imu_data_t *)event->data;
    // event->timestamp, event->frame_id, p->imu_data[i].accel/gyro/temp
}

// 5. CTRL Phase: Dynamic Control
sensor_hub_set_param(handle, SNS_CTL_RANGE, &range);
sensor_hub_set_param(handle, SNS_CTL_ODR, &new_rate);

// 6. Synchronous Sampling (IPD Liner)
sensor_hub_sample_with_sd(liner_sd, &raw, sizeof(raw));

// 7. STOP Phase
sensor_hub_close(handle);
```

6.5 Event Distribution Chain

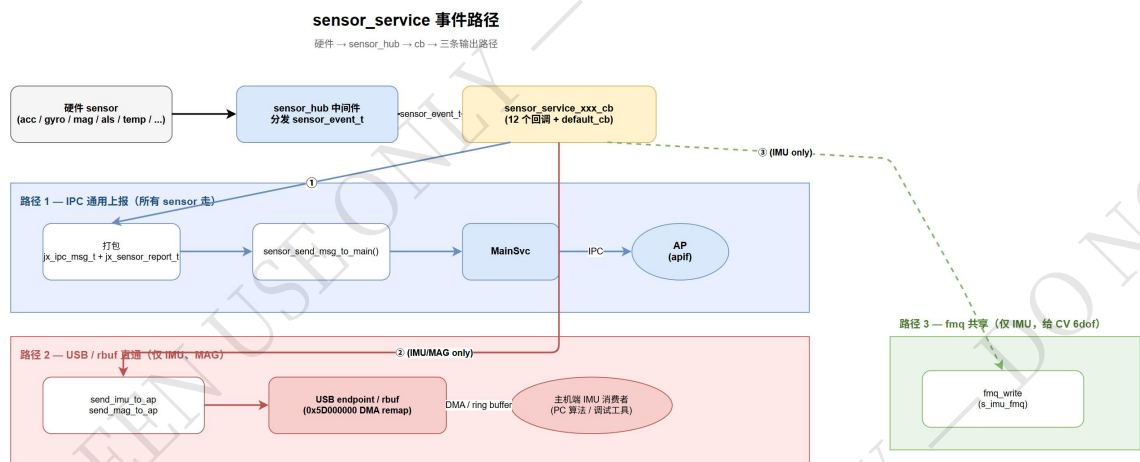
6.5.1 Registration Phase

Register CB Based on the Sensor Type Upon Startup:

```
g_sensor_service_info.sensor_cbs[SENSOR_IMU] = sensor_service_imu_cb;
g_sensor_service_info.sensor_cbs[SENSOR_TEMP] = sensor_service_temp_cb;
g_sensor_service_info.sensor_cbs[SENSOR_ALS] = sensor_service_als_cb;
g_sensor_service_info.sensor_cbs[SENSOR_FLK] = sensor_service_flk_cb;
g_sensor_service_info.sensor_cbs[SENSOR_FAN] = sensor_service_fan_cb;
g_sensor_service_info.sensor_cbs[SENSOR_PROX] = sensor_service_prox_cb;
g_sensor_service_info.sensor_cbs[SENSOR_HALL] = sensor_service_hall_cb;
g_sensor_service_info.sensor_cbs[SENSOR_MAG] = sensor_service_mag_cb;
g_sensor_service_info.sensor_cbs[SENSOR_KEY] = sensor_service_key_cb;
g_sensor_service_info.sensor_cbs[SENSOR_ROTATION] = sensor_service_rotation_cb;
g_sensor_service_info.sensor_cbs[SENSOR_MOTION] = sensor_service_motion_cb;

sensor_hub_register_default_cb(sensor_service_def_cb);
```

6.5.2 Event Path

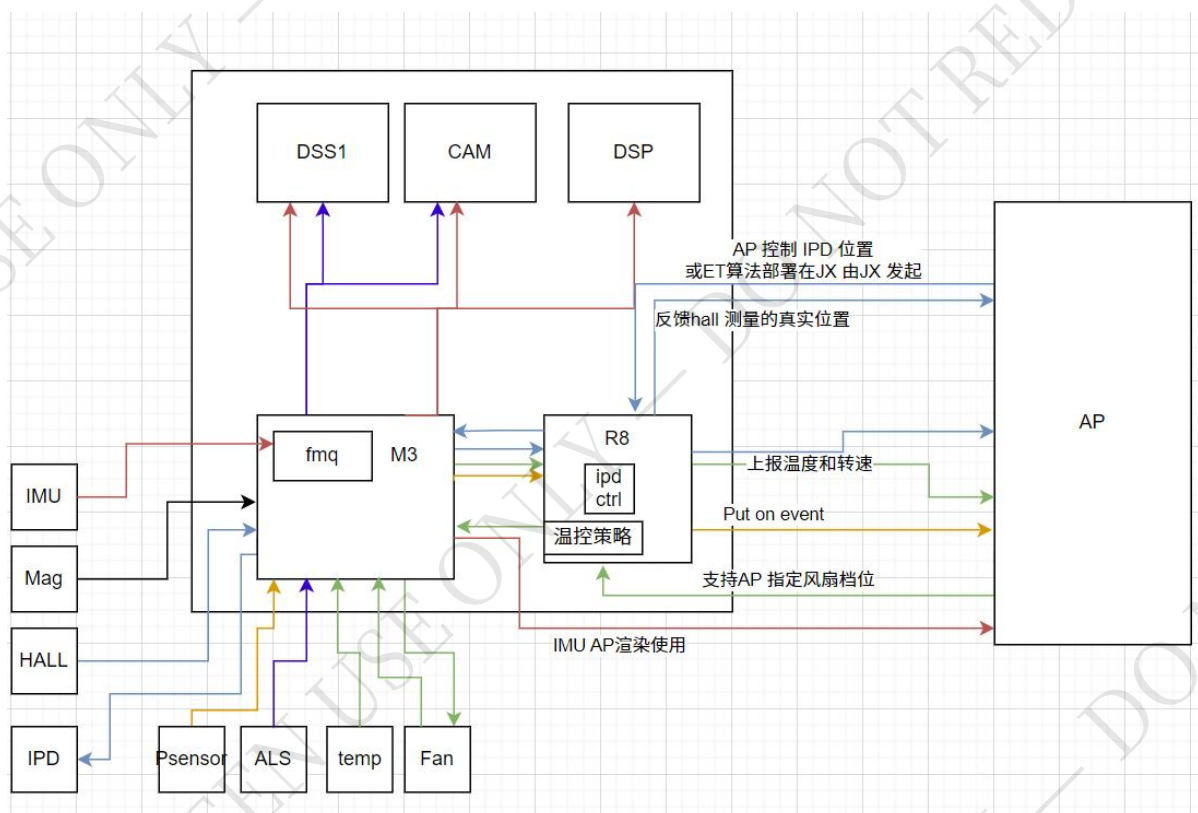


CB: After Triggering, Three Output Paths:

#	Way	Suit	Realize
①	IPC General Reporting	All sensors	Pack jx_ipc_msg_t + jx_sensor_report_t → sensor_send_msg_to_main() → MainSvc → AP (apif)

#	Way	Suit	Realize
②	USB / rbuf Direct Through	IMU, MAG only	send_imu_to_ap / send_mag_to_ap → USB endpoint (0x5D000000 DMA remap) / rbuf
③	fmqenjoy together	OnlyIMU	fmq_write(s_imu_fmq) → CV Service – 6DOF Consumer

6.5.3 Sensor_flow Data Stream



6.6 IPD Subsystem

IPD_Service.C represents the more complex component within sensor_service; the independent thread IpdSvc handles the closed-loop control for the stepper motor and linear sensor.

6.6.1 Hardware

- **IPD Stepper Motor**— Adjusts interpupillary distance

- **Liner Sensor**—Type is configurable(`ipd_liner_sensor_type`; typically Hall, ADC, or a dedicated position sensor)

6.6.2 Calibration Parameters

```
typedef struct {  
    rt_bool_t is_valid;  
    float a;           // 线性系数  
    float b;           // 偏移  
    float ipd_big;      // 最大瞳距 mm  
    float ipd_small;    // 最小瞳距 mm  
} cali_para_t;
```

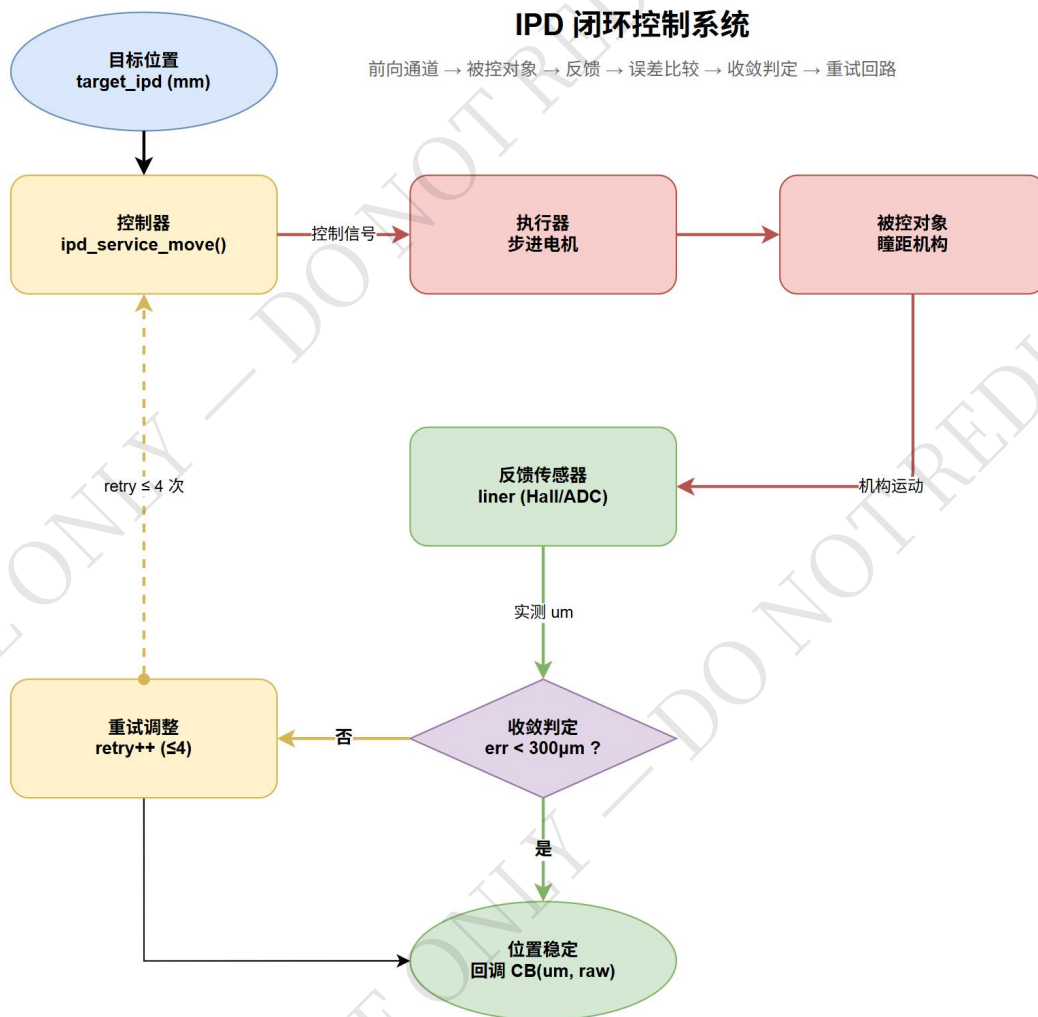
```
ipd_mm = raw * a + b
```

6.6.3 Two Movement Modes

```
ipd_service_move(req, ack_ap, cb)
```

```
|  
├── ABS_MODE  
|   1. Verify the `cali` parameter (is_valid & & a!= 0)  
|   2. Boundary Check ( $mm \in (ipd\_small, ipd\_big)$ )  
|   3. `ipd_get_value()` – Reads the current linear value  $\rightarrow mm$   
|   4.  $\Delta mm = (mm - curr\_mm) / 2$  ← Eye-to-eye distance; the motor moves only  
half the distance |   5. steps =  $|\Delta mm| / step\_mm$   
|   6. step_motor_move(steps, dir)  
|  
└── REL_MODE  
    ├── With calibration  $\rightarrow$  Set ABS_MODE ( $ipd\_curr\_mm + mm$ )  
    └── No calibration  $\rightarrow$  Directly set to  $mm / step\_mm$  step size
```

6.6.4 8.4 Closed-Loop Retry



Closed-loop core:IPD_Service_Process— The worker thread waits for the motor to send a completion signal, then re-reads the linear feedback and proceeds accordingly.

target_ipd supports two modes:

Pa-tt-ern	Target	Demarcate	Retry
ABS	Absolute position (mm)	Requires (parameters a, b)	If the error <300 μm or retry = 4 → Operation completed; otherwise, call ipd_move(ABS_MODE) and retry.
REL	Relative displacement(mm)	Non-essential	Single-step direct callback – no retry

6. 7 Sensor_custom Integration

CFG phase product-level sensor topology(config/sensorhub/sensor_custom.json):

6.7.1 Sensor_custom_dev_t Structure

```
typedef struct sensor_custom_dev_t {
    jx_u32_t          magic_number;
    imu_module_info_t imu0_module;    // acc/gyro range
    `imu_module_info_t imu1_module; // The second IMU`
    als_module_t      als_module;     // + fac cali
    psensor_module_t  psensor_module; // + threshold[3]
    magnetic_module_t magnetic_module; // + threshold[3]
    ipd_module_t      ipd_module;     // ipd_num + ipd_instance[2]
    motion_module_t   motion_module;  // enabled + cfg + hw
    led_module_t      led_module;     // led_num + leds[4]
    adc_module_t      adc_module;     // adc_num + adcs[4]
    key_module_t      key_module;     // key_num + keys[10]
    hall_module_t     hall_module;    // hall_num + halls[4]
    liner_module_t    liner_module;   // liner_num + liners[2]
    touch_module_t    touch_module;   // touch_num + touches[4]
    fan_module_t      fan_module;     // fan_num + fans[4]
    temp_ctrl_t       temp_ctrl;      // 5-stage temperature threshold + fan PID[3]
    rotation_module_t rotation_module; // + rot_dia
} sensor_custom_dev_t;
```

6.7.2 sensor_hub_add_hw_info

For CFG, Call ENABLE_*_SENSOR Using the Macro Type:

```
sensor_hubadd_hw_info(custom_cfg)
├─#ifdef ENABLE_IMU_SENSOR    → sensor_hub_register_dev(imu0,
    SENSOR_ ...)
    IMU, ...)
├─#ifdef ENABLE_MAG_SENSOR    → sensor_hub_register_dev(mag,
    SENSOR_ ...)
    MAG, ...)
├─#ifdef ENABLE_ALS_SENSOR    → sensor_hub_register_dev(als,
    SENSOR_ ...)
    ALS, ...)
├─#ifdef ENABLE_KEY_SENSOR    → sensor_hub_register_dev(keys[i],
    SENSOR_ ...)
    KEY, ...)
├─#ifdef ENABLE_TEMP_SENSOR   → sensor_hub_register_dev(temp,
    SENSOR_TEMP, ...)
```

```
└─#ifdef ENABLE_FAN_SENSOR    → sensor_hub_register_dev(fans[i],  
SENSOR_FAN, ...)  
└─#ifdef ENABLE_LED_DRV      → sensor_hub_register_dev(leds[i],  
SENSOR_LED, ...)  
└─#ifdef ENABLE_IPD_DRV      → sensor_hub_register_dev(ipd,  
SENSOR_IPD, ...)  
└─#ifdef ENABLE_LINER_SENSOR → sensor_hub_register_dev(liners[i],  
SENSOR_LINER, ...)  
└─#ifdef ENABLE_HALL_SENSOR  → sensor_hub_register_dev(halls[i],  
SENSOR_HALL, ...)  
└─#ifdef ENABLE_TOUCH_SENSOR → sensor_hub_register_dev(touchs[i],  
SENSOR_TOUCH, ...)  
└─#ifdef ENABLE_ADC_SENSOR   → sensor_hub_register_dev(adcs[i],  
SENSOR_ADC, ...)  
└─#ifdef ENABLE_MOTION_SENSOR → sensor_hub_register_dev(motion,  
SENSOR_MOTION, ...)  
└─#ifdef ENABLE_ROTATION_SENSOR → sensor_hub_register_dev(rot,  
SENSOR_ROTATION, ...)  
└─#ifdef ENABLE_PROX_SENSOR  → sensor_hub_register_dev(psensor,  
SENSOR_PROX, ...)
```

Each sensor type must be registered only when the ENABLE_*SENSOR macro is enabled; otherwise, that sensor type will be skipped in sensor_service_init/start. This mechanism allows for flexible deployment of sensors across R8 and M3 devices; it is recommended to deploy sensors requiring low-power wake-up capability on the M3 devices and the remaining sensors on the R8 devices.

6.7.3 Auto-Start

Sensor_auto_start— The Sensor Is Automatically Enabled Upon Completion of the CFG:

```
KEY(3, 30Hz) / TEMP(1, 30Hz) / FAN(4, 30Hz) /  
IPD(1, 30Hz) / ADC(1, 30Hz) / LED(4, 30Hz) /  
PROX(1, 2Hz) / ROTATION(1, 30Hz)
```

The remaining sensors (IMU/MAG/MOTION/ALS/HALL/LINER) are activated by the AP on demand.

6.7.4 sanity_test

During the CFG Phase, Iterate Over All Registered Sensors and Use Sensor_Hub_Get_Dev (Sd) to Verify Whether the Corresponding Idx Can Be Found; if the Operation Fails, Return the Failure Bitmap to the AP.

Bitmap Layout:

- **General Type:** Failure Bit= 1 <<type (e.g. 1 <<SENSOR_MAG indicates MAG failure)
- **IMU:** Failure Flag= SENSOR_IMU0_SANITY_ERR_MASK <<sensor_idx (Each IMU occupies an independent bit; both IMU0 and IMU1 failures can be reported simultaneously)

```
// sensor_sanity_test:333
if (sanity_fail) {
    if (type == SENSOR_IMU) {
        sanity_ret = SENSOR_IMU0_SANITY_ERR_MASK << sensor_idx;
    } else {
        sanity_ret = 1 << type;
    }
}
```

6.8 Temperature Control Strategy

6.8.1 Monitoring Target

Monitor the Internal Temperature Nodes of the Chip and Use the Maximum Value Among All Nodes as the Temperature Strategy Input.

6.8.2 Temperature Data Source

Node	Module
1	ISP
2	DSP
3	CPU
4	CPU
5	DPU
6	DSP
7	NPU
8	ISP
9	DPU
10	PERI_TOP

Nod e	Module
11	DSP
12	DSP
13	MONO
14	DPU2
15	APIF_ DATA

Strategy Input: Use the maximum value among all nodes (1–15) as the temperature control input.

6.8.3 Temperature Setting Definition

Temperature control is available in five settings; Settings 1–3 adjust the fan speed using a temperature control strategy.

Gear	Trigger Temperature	Fan Strategy	Explain
Gear 1	< 53° C (55–2)	Minimum RPM (Default: 50%)	Silence Interval
Gear 2	≥ 55° C, < 70° C	PID Adaptive (50%–100%)	Adaptive Interval
Gear 3	≥ 70° C, < 85° C	100% constant rotational speed	High-power range
Gear 4	≥ 85° C	Report high-temperature abnormality event	High-temperature anomaly
Gear 5	≥ 100° C	Automatic Power Off Protection (PVT ALARMB)	Ultra-high temperature (irreversible)

1. Detailed Description for Each Gear Level

Gear 1: Silent Mode

- entry condition: temperature <53°C
- Exit Condition: Temperature ≥ 55°C (2°C hysteresis)
- Fan speed: Minimum rotational speed (Default: 50%; configurable)
- control strategy: Fixed minimum rotational speed operation

- **Goal:** Maintain quiet operation – ideal for low-load scenarios.

Gear Position 2: Adaptive Range (PID Control)

- **entry condition:** temperature $\geq 55^{\circ}\text{C}$
- **Exit Condition:** Temperature $< 53^{\circ}\text{C}$ (2°C hysteresis)
- **temperature range:** $55^{\circ}\text{C} - 70^{\circ}\text{C}$
- **Fan speed:** 50% – 100% Adaptive Adjustment
- **control strategy:** Use the PID algorithm to maintain the temperature at an appropriate level.
- **Goal:** Balance noise levels and heat dissipation

Gear 3: Maximum Power Range

- **entry condition:** temperature $\geq 70^{\circ}\text{C}$
- **Exit Condition:** Temperature $< 65^{\circ}\text{C}$ (5°C hysteresis)
- **temperature range:** $70^{\circ}\text{C} - 85^{\circ}\text{C}$
- **Fan speed:** 100% Constant Speed
- **control strategy:** Maximum heat dissipation
- **Goal:** Rapid temperature reduction

Gear 4: High-Temperature Anomaly

- **Entry condition:** Temperature $\geq 85^{\circ}\text{C}$
- **Exit condition:** Temperature $< 80^{\circ}\text{C}$ (5°C hysteresis)
- **movement:**
 - Report high-temperature abnormal events to the upper tier.
 - The upper layer can disable certain functions to reduce power consumption.
- **Objective:** Notify higher authorities to take further measures
- **movement:**
 - Report high-temperature abnormality event
 - The upper layer can disable certain functions to reduce power consumption.
- **Objective:** Notify higher authorities to take further measures

Gear 5: Ultra-High Temperature

- **Entry condition:** Temperature $\geq 100^{\circ}\text{C}$
- **Trigger Method:** Chip PVT (Process, Voltage, Temperature) Interrupt – ALARMB (100°C)
 - **movement:**
 - Trigger PVT ALARM interrupt
 - Execute `pvt\_irq\_handler\_lv2()` Call `jx\_pmic\_all\_off()` Chip Power Down
- **Recovery:** Irrecoverable – requires manual restart
- **Objective:** Hardware Protection

6.8.4 Ultra-High Temperature Protection Implemented

The code is located at subsys/sensorhub/sensor_drv/dev/pvt_temp.c.

```
// PVT Interrupt Handler – Level 1 (Warning)
jx_void_t pvt_irq_handler_lv1(jx_handle_t para)
{
    When the temperature reaches 80 °C, a warning is triggered.
    // Optional: Reduce power consumption or report events

// PVT Interrupt Handler – Level 2 (Ultra-High Temperature Power Down)
jx_void_t pvt_irq_handler_lv2(jx_handle_t para)
{
    When the temperature reaches 100 °C, the chip is powered off.
    jx_pmic_all_off();
}

// PVT Initialization
static void pvt_temp_init(void)
{
    // Initialize PVT module
    jx_pvt_temp_init();

    // Installation Level 1 – Interrupt (80 °C)
    jx_pvt_temp_alarm_install(PVT_IRQ_ALARMA_RISING, 80,
pvt_irq_handler_lv1, NULL);
```

// Installation Level 2: Thermal shutdown (100 °C) – Automatic power-off at extreme temperatures

```
jx_pvt_temp_alarm_install(PVT_IRQ_ALARMB_RISING, 100,
pvt_irq_handler_lv2, NULL);
}
```

6.8.5 PVT Interrupt Level Description

Rank	Trigger Temperature	Interrupt Handling	Explain
ALARMA (1v1)	80° C	pvt_irq_handler_lv1	Temperature warning – helps reduce power consumption
ALARMB (1v2)	100° C	pvt_irq_handler_lv2	Ultra-high temperature – Chip power-down

6.8.6 Hysteresis Control Mechanism

Hysteresis control prevents sudden fluctuations in fan speed caused by frequent gear shifting. The hysteresis range is fixed and cannot be configured.

6.8.7 Hysteresis Interval

Gear shift	Hysteresis interval	Explain
Mode 2 → Mode 1	2° C	Exit Mode 2 when the temperature drops to 53 °C (55–2).
Mode 3 → Mode 2	5° C	Exit Mode 3 when the temperature drops to 65 °C (70–5).
Mode 4 → Mode 3	5° C	Exit from Mode 4 when the temperature drops to 80 °C (85–5).
Pattern 5	Not have	Once achieved, the device will be powered off immediately; it will not be restored.

```
// Hysteresis control pseudocode
if (temp >= LEVEL2 && current_mode < MODE2) {
    set_mode(MODE2); // 进入模式 2
} else if (temp <(LEVEL2-2) && current_mode == MODE2) {set_
mode(MODE1); // Exit Mode 2 and return to Mode 1
}
```

```
if (temp >= LEVEL3 && current_mode < MODE3) {
    set_mode(MODE3);
} else if (temp < (LEVEL3 - 5) && current_mode == MODE3) {
    set_mode(MODE2);
}

if (temp >= LEVEL4 && current_mode < MODE4) {
    set_mode(MODE4);
} else if (temp < (LEVEL4 - 5) && current_mode == MODE4) {
    set_mode(MODE3);
}

if (temp >= LEVEL5) {
    power_off(); // Shut down – no recovery
}
```

6.8.8 Fan Control

1. Fan Support

- Supports multiple fans operating simultaneously
- The default control strategy applies uniform control to all fans.

2. Fan Control Interface

```
// Set whether the fan is controlled by a control strategy
jx_void_t temp_ctrl_set_fan_auto(uint16_t id, uint16_t is_auto);

//parameter declaration
// id: Fan ID (0, 1, 2). . .)
// is_auto: 0-Manual control; not affected by the
temperature control strategy // 1-Automatic control,
influenced by the temperature control strategy
```

3. Fan Control Example

```
// Set Fan 0 to automatic control mode
```

```
temp_ctrl_set_fan_auto(0, 1);

// Set Fan 1 to manual control mode (not subject to
temperature control strategy): temp_ctrl_set_fan_auto(1,
0);

// Set all fans to automatic control mode
for (int i = 0; i < FAN_COUNT; i++) {
    temp_ctrl_set_fan_auto(i, 1);
}
```

4. Fan Speed Setting

In Automatic Mode, the Fan Speed Is Determined by the Temperature Setting:

Gear	Fan speed
Gear 1	temp_ctrl_fan_min (Default: 50%)
Gear 2	PID Adaptive Regulation (50% – 100%)
Gear 3	temp_ctrl_fan_max (100%)

6.8.9 PID Control Algorithm

Gear Position 2 Utilizes a PID Algorithm for Adaptive Regulation; It Supports the Configuration of Either a Simple Positional PID or a Curve-Based PID Controller.

1. PID Parameters

Parameter	Explain	Windows default
Kp	Coefficient of proportionality	To be optimized
Ki	Integral coefficient	To be optimized

Parameter	Explain	Windows default
Kd	Differential quotient	To be optimized

2. PID Control Logic

```
// PID Control Pseudocode
int16_t error = current_temp - setpoint;
integral += error * dt;
derivative = (error - prev_error) / dt;

output = Kp * error + Ki * integral + Kd * derivative;
fan_speed = clamp(MIN_SPEED + output, MIN_SPEED, MAX_SPEED);
```

6.8.10 Temperature Protection Mechanism

1. Gear 4-Abnormal High Temperature

- **Trigger condition:** Reaching the 4th gear temperature threshold
- **movement:**
 - Report high-temperature abnormal events to the upper tier.
 - The upper layer can temporarily disable certain functions to reduce power consumption.
- **Restore:** Automatic restoration upon temperature drop

2. Gear 5-Ultra-High Temperature

- **Trigger condition:** The temperature reaches the gear 5 temperature threshold ($\geq 100^{\circ}\text{C}$)
- **Trigger Method:** Chip PVT (Process, Voltage, Temperature) Interrupt – ALARMB
- **movement:**
 - PVT ALARM – Interrupt Trigger (100°C)
 - Execute `pvt\_irq\_handler\_lv2()` Call `jx\_pmic\_all\_off()` Chip Power Down
- **Recovery:** Irrecoverable – requires manual restart

6.8.11 Event Reporting

1. Temperature Event Notification

Send an SENSOR_EVENT_TEMP_HIGH event notification to the upper layer:

Parameter values	Meaning	Trigger Condition
0	Normothermia	Shift to Gear 3 and back to Gear 2
1	Hyperpyrexia	Select Gear 3
2	Temperature too high – exceeds gear 4	Select Gear 4

2. Event Definition

```
typedef enum {  
    TEMP_NORMAL = 0; // Normal temperature  
    TEMP_HIGH_LEVEL3 = 1; // High temperature (between Mode 3 and  
    Mode 4) TEMP_HIGH_LEVEL4 = 2; // High temperature exceeding Mode  
    4  
} temp_event_level_t;  
  
//event structure  
typedef struct {  
    SENSOR_EVENT_T base;  
    temp_event_level_t level; // Temperature event level  
} SENSOR_EVENT_TEMP_HIGH;
```

6.8.12 Configuration Interface

1. Temperature Control Configuration Parameters

```
{  
    "temp_ctrl": {
```

```
"temp_ctrl_on": {
  "Type": "bool",
  "Value": true
},
"temp_ctrl_level2": {
  "Type": "float",
  "Value": 55.0
},
"temp_ctrl_level3": {
  "Type": "float",
  "Value": 70.0
},
"temp_ctrl_level4": {
  "Type": "float",
  "Value": 85.0
},
"temp_ctrl_level5": {
  "Type": "float",
  "Value": 100.0
},
"temp_ctrl_fan_min": {
  "Type": "uint32_t",
  "Value": 50
},
"temp_ctrl_fan_max": {
  "Type": "uint32_t",
  "Value": 100
},
"temp_ctrl_fan_pid": {
  "Type": "float",
  "Column": 3,
  "Value": [
    -3,
    -0.05,
    -0.5
  ]
}
}
```

2. Configuration Parameters Explanation

Parameter	Type	Price	Explain
temp_ctrl_on	bool	true	Temperature detect switch
temp_ctrl_level2	float	55.0	Gear 2 Threshold (Enter Adaptive Range)
temp_ctrl_level3	float	70.0	Gear 3 Threshold (Enter Maximum Power Range)
temp_ctrl_level4	float	85.0	Gear 4 Threshold (High-Temperature Abnormality)
temp_ctrl_level5	float	100.0	Gear 5 Threshold (PVT ALARMB – Over-Temperature Power Off)
temp_ctrl_fan_min	uint32	50	Fan minimum speed (%)
temp_ctrl_fan_max	uint32	100	Maximum fan speed (%)
temp_ctrl_fan_pid	float[]	[-3, -0.05, -0.5]	PID parameters [Kp, Ki, KD]

7. Drivers

7.1 I2C/SPI/UART/GPIO Driver

7.1.1 UART

The X100 chip's UART driver is implemented using Synopsys DesignWare IP, providing interfaces for serial port initialization, data transmission and reception, interrupt control, and DMA transfer. The header file is located in `platform/device/include/uart.h`.

1. Data Type

uart_config_t — Serial port configuration structure

```
typedef struct {  
    jx_u32_t bps; /* Baud Rate */  
    jx_u8_t parity; /* Check bit */  
    jx_u8_t stop_bits; /* Stop bit */  
    jx_u8_t data_bits; /* Data bits */  
    jx_u8_t flow_ctrl; /* Flow control enable */  
    jx_void_t (*callback_input) (jx_u8_t rx); /* Receive interrupt  
callback */} uart_config_t;
```

Baud rate enumeration

Macro	Price	Explain
UART_BPS_DEFAULT	9600	Default baud rate
UART_BPS_19200	19200	-
UART_BPS_38400	38400	-
UART_BPS_115200	115200	-

*You can also enter a custom baud rate value directly (e.g., `3000000`); the driver will then automatically calculate the frequency division factor based on the clock source.

Checkbit Enumerated Type

Macro	Price	Explain
UART_PARITY_NONE	0	No verification
UART_PARITY_ODD	1	Odd parity
UART_PARITY_EVEN	2	Even parity check

Stop Bit Enumerated Value

Macro	Price	Explain
UART_STOPBITS_I	0	1 Stop bit
UART_STOPBITS_I_V	1	1.5 Stop bit
UART_STOPBITS_II	2	2 Stop bit

Data Bit Enumeration

Macro	Price	Explain
UART_DATABITS_V	0	5 Bit data
UART_DATABITS_VI	1	6 Bit data
UART_DATABITS_VII	2	7 Bit data

UART_DATABITS_VIII	3	8 Bit data
--------------------	---	------------

Flow Control Enumeration

Macro	Price	Explain
UART_FLOWCTRL_DISABLE	0	Disable hardware flow control
UART_FLOWCTRL_ENABLE	1	Enable hardware flow control (RTS/CTS)

Interrupt Control Enumerated List

Macro	Price	Explain
UART_RX_INT_DISABLE	0	Disable receive interrupt
UART_RX_INT_ENABLE	1	Enable reception interrupt
UART_TX_INT_DISABLE	2	Disable Send Interrupt
UART_TX_INT_ENABLE	3	Enable send interrupt

2. API Interface

jx_uart_init — Serial port initialization

function:

Configure the UART baud rate, parity bit, stop bit, data bit, and flow control, and enable the FIFO.

parameter:

- `dev`: UART device instance obtained via a board-level configuration macro (e.g., `DEVICE\_BASE\_CONSOLE\_IO`).
- `cfg`: Serial Port Configuration Parameters

returned value:

- `0`: Success
- Negative value: Failure (returns errno)

instance:

```
const device_ins_t uart = (device_ins_t)DEVICE_BASE_CONSOLE_IO;
uart_config_t cfg = {
    .bps          = UART_BPS_115200,
    .parity       = UART_PARITY_NONE,
    .stop_bits    = UART_STOPBITS_I,
    .data_bits    = UART_DATABITS_VIII,
    .flow_ctrl    = UART_FLOWCTRL_DISABLE,
};
```

```
jx_uart_init(uart, &cfg);
```

jx_uart_getchar — Receive a single byte

function:

Read one byte from the UART receive FIFO.

parameter:

- `dev`: UART Device Instance
- `poll`:
 - `JX\_TRUE`: Polling mode – waits blockingly until data is available.
 - `JX\_FALSE`: Non-queroying mode – returns-1 immediately when no data is available.

returned value:

- `[0, 255]`: Received bytes
- `-1`: No data (in non-querulous mode)
- `-EBUSY`: Conflict when asynchronous reception is enabled

instance:

```
/* Non-blocking read */
```

```
int ch = jx_uart_getchar(uart, JX_FALSE);
```

```
if (ch >= 0) {
```

```
    /* Valid data received */
```

```
}
```

```
/* Blocking Wait */
```

```
int ch = jx_uart_getchar(uart, JX_TRUE);
```

jx_uart_putchar — Send a single byte

function:

Send a single byte to the UART. The operation blocks until the transmit FIFO is full before writing data.

parameter:

- `dev`: UART Device Instance
- `out\_char`: Bytes to be sent

instance:

```
jx_uart_putchar(uart, 'A');
```

jx_uart_puts — Send multiple bytes

function:

Send multiple bytes to the UART until the FIFO is full (non-blocking filling).

parameter:

- `dev`: UART Device Instance
- `tx\_data`: Data Transmission Buffer
- `size`: Number of bytes sent

returned value:

- Actual number of bytes transmitted
- `-ENOTSUP`: Send (Disabled)

instance:

```
jx_u8_t buf[] = "Hello\r\n";  
int sent = jx_uart_puts(uart, buf, sizeof(buf) - 1);
```

jx_uart_interrupt_set — Enable/Disable interrupt

function:

Set the UART transmit/receive interrupt enable state.

parameter:

- `dev`: UART Device Instance
- `mask`: Interrupt control value; refer to Section 2.7 for the enumeration of possible values.

instance:

```
jx_uart_interrupt_set(UART, UART_RX_INT_ENABLE); /* Enable receive interrupt */  
jx_uart_interrupt_set(UART, UART_TX_INT_DISABLE); /* Disable transmit interrupt */
```

jx_uart_fifo_clear — Clear FIFO

Function: Clear the UART transmit/receive FIFO.

parameter:

- `dev`: UART Device Instance

Return value: `JX\_OK`

jx_uart_tx_empty — Check whether the transmit FIFO is empty

function:

Check whether both the transmit shift register and the transmit FIFO have been emptied.

parameter:

- `dev`: UART Device Instance

returned value:

- `JX\_TRUE`: The sending FIFO is empty.
- `JX\_FALSE`: Data is still being sent.

instance:

```
jx_uart_puts(uart, data, len);
while (!jx_uart_tx_empty(uart)) {
    /* Wait for transmission completion */
}
```

DMA interface (must be enabled; use the DEVICE_DMA macro)

uart_dma_config_t — DMA configuration structure

```
typedef struct {
    jx_u8_t    type;          /* Transmission direction: UART_DMA_TX / UART_DMA_RX */
    jx_u8_t    channel_prio; /* DMA channel priority */
    block_cfg_t *block; /* Transmission Data Block Configuration */
    `jx_handle_t` callback; /* Transmission Complete Callback */
    jx_void_t  *para;        /* Callback Parameters */
    jx_u32_t    intr_prio; /* Interrupt priority */
    jx_u32_t    core;        /* Target CPU ID (valid for SMP systems) */
    jx_bool_t    aon_enable; /* AON domain enable */
} uart_dma_config_t;
```

jx_uart_ahb_dma_init — Initialize AHB DMA

function:

Acquire an AHB DMA handle and initialize the configuration for UART transmission and reception using AHB DMA.

parameter:

- `dev`: UART Device Instance
- `cfg`: `uart\_dma\_config\_t` type – used to configure the AHB DMA channel.

Return value: DMA handle; `JX\_NULL` indicates failure.

```
instance:
uart_dma_config_t uart_dma_cfg = {
    .type = UART_DMA_TX,
    .callback = uart_dma_irq_handle,
    .block = &tx_data_block,
};
console_dma_handle = jx_uart_ahb_dma_init(uart, &uart_dma_cfg);
jx_dma_start(console_dma_handle); /* Start data transfer */
```

jx_uart_axi_dma_init — Initialize AXI DMA

function:

Acquire an AXI DMA handle and initialize the configuration for UART transmission and reception using AXI DMA.

parameter:

- 'dev': UART Device Instance
- 'cfg': 'uart_dma_config_t' type – used to configure

the AXI DMA channel. Return value: DMA handle;

"JX_NULL" indicates failure.

3. Matters Need Attention

1. **Pin Multiplexing:** Before using the UART, configure the corresponding pin for UART functionality via the 'jx_pin_function_set()'; refer to the pin mapping table in the 'board/*/soc/pin_list.h' (e.g., 'PIN_0__UART0_TX', 'PIN_1__UART0_RX').
2. **波特率精度:** The driver automatically calculates the division factor based on the clock source provided by CRG; therefore, there may be an error for non-standard baud rates.
3. **FIFO Clear:** After initialization, it is recommended to call the 'jx_uart_fifo_clear()' function to clear any residual data in the FIFO.
4. **Send Completion Check:** After batch transmission, if you need to confirm that the data has been fully sent, use 'jx_uart_tx_empty()' to wait; the 'jx_uart_puts()' response only indicates that the data has been loaded into the FIFO.
5. **DMA Transceiver:** For UART communication using DMA, it is recommended to use AHB DMA.
6. **Serial port log configuration:** When using the UART for the serial port logging functionality, you must edit the project's 'conf' file to specify the UART instance for the logging feature. For detailed instructions, please refer to the Project Configuration Guide.

7.1.2 GPIO

The X100 chip's GPIO driver is implemented using Synopsys DesignWare IP and supports direction configuration, level reading/writing, and interrupt management for a total of 61 + 132 GPIO pins (D0–D1). The header file is located at `platform/device/include/gpio.h`.

1. Pin Numbering

GPIO pins are enumerated using the `gpio\_list\_t` and are defined in `platform/include/bsp\_json\_cfg/gpio\_user\_define.h`:

Enum Prefix	Starting number	Region	Explain
D0_GPIO_0 ~ D0_GPIO_60	0 ~ 60	D0	D0-side GPIO
D1_GPIO_0 ~ D1_GPIO_16	64 ~ 80	D1 AON	D1 Always-On Domain
D1_GPIO_17 ~ D1_GPIO_131	96 ~ 210	D1 PERI	D1 Peripheral Interface Domain

Valid range: 0 ~ GPIO_NUM_MAX-1; if an invalid value is passed, the interface returns JX_INVALID.

2. Macro Definition

directional control

Macro	Price	Explain
GPIO_DIRECTION_INPUT	0	Input pattern
GPIO_DIRECTION_OUTPUT	1	Output Mode

Level Definition

Macro	Price	Explain
GPIO_LEV_LOW	0	Low level
GPIO_LEV_HIGH	1	High level

Interrupt Trigger Mode

Macro	Price	Explain
GPIO_IRQ_MODE_RISING_EDGE	0	Rising-edge trigger

GPIO_IRQ_MODE_FALLING_EDGE	1	Rising-edge trigger
GPIO_IRQ_MODE_LOW_LEVEL	2	Low-level triggering
GPIO_IRQ_MODE_HIGH_LEVEL	3	High-level triggering
GPIO_IRQ_MODE_RISING_FALLING	4	Edge triggering

Pin status (for `jx\_gpio\_pins\_write`/`jx\_gpio\_pins\_read`)

Macro	Price	Explain
GPIO_STATUS_INPUT_LOW	0	Low-level input
GPIO_STATUS_INPUT_HIGH	1	High-level input
GPIO_STATUS_OUTPUT_LOW	2	Output low level
GPIO_STATUS_OUTPUT_HIGH	3	High-level output

3. API Interface

jx_gpio_direction_set — Set pin direction

function:

Set a specified GPIO pin to input or output mode.

parameter:

- `pin\_num`: GPIO pin number (e.g., `D0\_GPIO\_0`, `D1\_GPIO\_16`)
- `direction`: uint32 type; uses `GPIO\_DIRECTION\_INPUT` or `GPIO\_DIRECTION\_OUTPUT`; when the value is greater than 1, it is considered `GPIO\_DIRECTION\_OUTPUT`.

returned value:

- `JX\_OK` (0): Success
- `JX\_INVALID`: `pin\_num` >= `GPIO\_NUM\_MAX`

instance:

```
/* Set D0_GPIO_5 to output */
```

```
jx_gpio_direction_set(D0_GPIO_5, GPIO_DIRECTION_OUTPUT);
```

```
/* Set D1_GPIO_3 as an input */
```

```
jx_gpio_direction_set(D1_GPIO_3, GPIO_DIRECTION_INPUT);
```

jx_gpio_pin_write — Set the output level for a single pin

Function: Sets the output level of a specified GPIO pin. This interface automatically sets the pin to output mode, eliminating the need to call `jx\_gpio\_direction\_set` in advance.

parameter

- `pin\_num`: GPIO pin number
- `level`: uint32 type; uses `GPIO\_LEV\_LOW` or `GPIO\_LEV\_HIGH`; when the value is greater than 1, it is considered `GPIO\_LEV\_HIGH`.

returned value:

- `JX\_OK` (0): Success
- `JX\_INVALID`: `pin\_num >= GPIO\_NUM\_MAX`

instance:

```
/* Drive high D0_GPIO_10 */
jx_gpio_pin_write(D0_GPIO_10, GPIO_LEV_HIGH);

/* Pull down D1_GPIO_16 */
jx_gpio_pin_write(D1_GPIO_16, GPIO_LEV_LOW);
```

jx_gpio_pin_read — Read single-pin level

function:

Read the logic level status of a specified GPIO pin. When the pin is configured as an output, it returns the value from the output register; when configured as an input, it returns the actual logic level of the input pin.

parameter:

- `pin\_num`: GPIO pin number

returned value:

- `0`: Low-level input
- `1`: High-level input
- `2`: output low level
- `3`: High-level output

> Note: The assertion (JX_ASSERT) is triggered when `pin\_num >= GPIO\_NUM\_MAX`.

instance:

```
jx_gpio_direction_set(D0_GPIO_5, GPIO_DIRECTION_INPUT);
jx_u32_t level = jx_gpio_pin_read(D0_GPIO_5);
if (level == GPIO_LEV_HIGH) {
    /* Pin is at a high logic level */
}
```



```
}

```

jx_gpio_pin_irq_register — Register a pin interrupt

function:

Register an interrupt callback function for a specified GPIO pin and configure its trigger mode.
The interrupt is automatically enabled upon registration.

parameter:

- `pin\_num`: GPIO pin number
- `irq\_mode`: Interrupt Trigger Mode; refer to Section 3.3 for values.
- `func`: Interrupt callback function; prototype: `void (\*callback)(void \*param)`
- `args`: User parameters passed to the callback function

returned value:

- `JX\_OK`: Success
- `JX\_INVALID`: Invalid `func == NULL`, `pin\_num >= GPIO\_NUM\_MAX`, or IRQ value

instance:

```
static void my_gpio_isr(void *arg)
{

```

```
    /* Handle GPIO interrupt */

```

```
    APP_LOGW("GPIO interrupt triggered on pin, arg=%p", arg);

```

```
}

```

```
/* Register D0_GPIO_8 rising-edge interrupt */

```

```
jx_gpio_pin_irq_register(D0_GPIO_8, GPIO_IRQ_MODE_RISING_EDGE,
                        (jx_handle_t)my_gpio_isr, NULL);

```

jx_gpio_pin_irq_enable — Enable pin interrupt

function:

Enables interrupts for the specified GPIO pin. This is typically used after re-enabling the pin following a call to `jx\_gpio\_pin\_irq\_disable`.

parameter:

- `pin\_num`: GPIO pin number

returned value:

- `JX\_OK`: Success
- `JX\_INVALID`: `pin\_num >= GPIO\_NUM\_MAX`

jx_gpio_pin_irq_disable — Disable pin interrupt

function:

Disables interrupts for specified GPIO pins. The interrupt callback remains registered; it is simply no longer triggered.

parameter:

- `pin\_num`: GPIO pin number

returned value:

- `JX\_OK`: Success

- `JX\_INVALID`: `pin\_num >= GPIO\_NUM\_MAX`

instance:

```
/* Temporarily disable interrupt */
```

```
jx_gpio_pin_irq_disable(D0_GPIO_8);
```

```
/* Execute critical section operation... */
```

```
/* Re-enable interrupt */
```

```
jx_gpio_pin_irq_enable(D0_GPIO_8);
```

4. Matters Need Attention

1. **Pin Multiplexing:** Before using the GPIO functionality, you must configure the pin to GPIO mode via the `jx\_pin\_function\_set()`; refer to the pin mapping section in the `board/\*/soc/pin\_list.h` (for example, the `PIN\_FUNCTION\_1` for "PIN_0_GPIO0" indicates that it is set to GPIO mode).
2. **Pull-up/Pull-down Configuration:** In GPIO input mode, you can configure a pin as pull-up or pull-down using the `jx\_iomux\_pull\_set()` setting to prevent input floating.
3. **`jx\_gpio\_pin\_write` Automatically set as output:** When calling this interface, no prior direction configuration is required; it will automatically configure the pin in output mode.
4. **`jx\_gpio\_pin\_read` Behavioral differences:** In output mode, the value from the output register is read (rather than the actual pin level); in input mode, the actual pin level is read.
5. **Interrupt registration can be performed on the fly:** After calling `jx\_gpio\_pin\_irq\_register`, the interrupt is automatically enabled; there is no need to call `jx\_gpio\_pin\_irq\_enable` again.
6. **Interrupt callback during IRQ context execution:** Avoid time-consuming operations within the callback function; do not call blocking interfaces.
7. **D0/D1 GPIO Interrupt Physical Isolation:** The GPIO interrupt for D0 must be registered with C0/C1; the GPIO interrupt for D1 must be registered with C2/C3.

7. 1. 3 IOMUX

The IOMUX (Pin Multiplexing Controller) is responsible for selecting the functions and configuring the electrical characteristics of all external pins on the X100 chip, including pin multiplexing functionality, input enable, pull-up/pull-down control, drive strength, Schmitt trigger, and slew rate control. Before peripherals such as GPIO, UART, I2C, SPI, or PWM can be used with a pin, the IOMUX must first configure the pin for the corresponding function. The header file is located in the `platform/device/include/iomux.h` directory; the pin function mapping table and the enumeration of electrical characteristics are defined in the `board/\$DX/soc/pin\_list.h` (`\$DX` file (as `x100\_d0` or `x100\_d1`)).

1. Pin Numbering

The IOMUX interface uses `gpio\_list\_t` (defined in `platform/include/bsp\_json\_cfg/gpio\_user\_define.h`) as the pin indexing scheme, sharing the same numbering system as the GPIO.

The driver automatically routes requests based on the assigned address range to the corresponding IOMUX controller; the caller does not need to concern themselves with the controller's base address.

2. Data Type

pin_function_list_t-Pin Function Number

```
typedef enum {
    PIN_FUNCTION_0,
    PIN_FUNCTION_1,
    PIN_FUNCTION_2,
    PIN_FUNCTION_3,
    PIN_FUNCTION_4,
    PIN_FUNCTION_5,
    PIN_FUNCTION_6
} pin_function_list_t;
```

Each pin supports one of up to seven optional multiplexing functions. The specific function mapping for each pin is defined by macro definitions in the `pin\_list.h` file; the naming convention is `PIN\_<n>\_<FUNC>`, for example:

Macro	Price	Explain
PIN_0__UART0_TX	PIN_FUNCTION_0	PIN_0 is multiplexed as UART0_TX
PIN_0__GPIO0	PIN_FUNCTION_1	PIN_0 is multiplexed as GPIO0
PIN_2__UART1_TX	PIN_FUNCTION_0	PIN_2 is multiplexed as UART1_TX
PIN_2__I2C1_SCL	PIN_FUNCTION_1	PIN_2 is multiplexed as I2C1_SCL
PIN_2__GPIO2	PIN_FUNCTION_2	PIN_2 is multiplexed as GPIO2

To query all available functions for a specific pin, simply search for macros prefixed with `PIN\_<n>\_` in `board/\$DIE/soc/pin\_list.h`.

pull_select_t-Internal pull-up/pull-down configuration

```
typedef enum {
    PULL_SELECT_NO = 0,
    PULL_SELECT_DOWN = 4,
    PULL_SELECT_UP = 2,
    PULL_SELECT_STRONG_UP = 6,
} pull_select_t;
```

Enum value	Explain
PULL_SELECT_NO	No internal pull-up/pull-down functionality
PULL_SELECT_DOWN	Internal dropdown
PULL_SELECT_UP	Internal Pull-up
PULL_SELECT_STRONG_UP	Internal strong pull-up (supported only by D1)

driving_level_t-Driving intensity

```
typedef enum {
    DRIVING_SELECT_LV0 = 0,
    DRIVING_SELECT_LV1,
    ...
```

DRIVING_SELECT_LV15,} driving_level_t; The driving intensity ranges from 16 levels (LV0 to LV15), where a higher value indicates stronger driving capability. The specific current driving intensities are listed in the table below.

Current Drive Level	D0 (mA)			D1 (mA)		
	Min.	typical	Max.	Min.	typical	Max.
IOL Low Level Output Current @VOL						
0	2.1	3.3	4.6	0.8	1.0	1.5
1	4.3	6.5	9.2	1.1	1.5	2.2
2	6.3	9.7	13.7	2.3	2.9	4.3
3	8.4	13.0	18.3	3.4	4.4	6.5
4	10.5	16.2	22.8	4.5	5.8	8.6
5	12.6	19.4	27.3	5.7	7.3	10.8
6	14.7	22.6	31.8	6.8	8.8	13.0
7	16.8	25.8	36.4	7.9	10.2	15.1
8	18.8	28.9	40.8	9.0	11.7	17.2
9	20.9	32.2	45.4	10.2	13.1	19.4
10	23.0	35.3	49.9	11.3	14.6	21.6
11	25.1	38.6	54.4	12.4	16.1	23.7
12	27.1	41.7	58.8	13.5	17.5	25.8
13	29.2	44.9	63.3	14.7	18.9	28.0

14	31.2	48.0	67.8	15.8	20.4	30.1
15	33.3	51.2	72.3	16.9	21.9	32.3
IOH High Level Output Current @VOH						
0	1.9	3.2	4.6	0.7	1.1	1.6
1	3.8	6.4	9.2	1.1	1.6	2.3
2	5.7	9.6	13.8	2.1	3.2	4.7
3	7.5	12.7	18.3	3.2	4.8	7.0
4	9.5	15.9	22.9	4.2	6.4	9.3
5	11.3	19.1	27.5	5.3	8.0	11.7
6	13.2	22.2	32.0	6.3	9.6	13.9
7	15.1	25.4	36.5	7.4	11.2	16.2
8	17.0	28.6	41.1	8.4	12.8	18.5
9	18.9	31.7	45.6	9.4	14.4	20.8
10	20.8	34.9	50.1	10.5	16.0	23.1
11	22.6	38.0	54.7	11.5	17.6	25.4
12	24.6	41.2	59.2	12.6	19.1	27.6
13	26.4	44.3	63.7	13.6	20.7	29.9
14	28.3	47.5	68.2	14.6	22.3	32.1
15	30.2	50.6	72.8	15.7	23.9	34.4

3. API Interface

jx_pin_function_set-Configure pin multiplexing functionality

function:

Configure the multiplexing function for a specified pin. This operation only configures the Function Selection Register; it does not modify the PAD electrical characteristics.

parameter:

- `idx`: Pin number (e.g., `D0\_GPIO\_2`, `D1\_GPIO\_76`)
- `func\_num`: Reused function number; typically used to pass the return value of a macro (e.g., `PIN\_2\_I2C1\_SCL`) from `pin\_list.h`: `JX\_OK` (0)

instance:

```
/* Configure PIN_2 as I2C1_SCL (Function 1) */
```

```
jx_pin_function_set(D0_GPIO_2, PIN_2_I2C1_SCL);
```

```
/* Configure D1_GPIO_76 for PWM functionality (Function 1) */
```

```
jx_pin_function_set(D1_GPIO_76, PIN_FUNCTION_1);
```

jx_iomux_direction_set-Set input enable

function:

The separate pin-mapped Input Buffer Enable (IE bit) does not affect other configurations.

parameter:

- `idx`: Pin Numbering
- `input`: `JX\_TRUE` Enables input buffer; `JX\_FALSE`

Disables return value: `JX\_OK`(0)

instance:

```
/* Enable input buffering for D0_GPIO_5 */  
jx_iomux_direction_set(D0_GPIO_5, JX_TRUE);
```

jx_iomux_pull_set-Configure internal pull-up/pull-down functionality

function:

Setting the pull-up/pull-down state for a single pin does not affect other configurations.

parameter:

- `idx`: Pin Numbering
- `pull`: Up/Down mode; refer to Section 3.2 for details.

Return value: `JX\_OK`(0)

instance:

```
/* D0_GPIO_5 Pull-up */  
jx_iomux_pull_set(D0_GPIO_5, PULL_SELECT_UP);  
  
/* D1_GPIO_3 (Downward) */  
jx_iomux_pull_set(D1_GPIO_3, PULL_SELECT_DOWN);
```

jx_iomux_driving_level_set-Set driving intensity

function:

Setting the drive strength for a single pin does not affect other configurations.

parameter:

- `idx`: Pin Numbering
- `lv`: Drive intensity level (`DRIVING\_SELECT\_LV0` ~ `DRIVING_SELECT_LV

15`) – Return value: `JX_OK`(0)

jx_iomux_schmitt_trigger_set-Set Schmitt trigger

function:

Enabling or disabling a Schmitt trigger for a single pin does not affect other configurations.

parameter:

- `idx`: Pin Numbering
- `enable`: `JX\_TRUE` – Enabled; `JX\_FALSE` –

Disabled; Return Value: `JX\_OK` (0)

jx_iomux_slew_rate_control_set-Set slew rate control (optional)

function:

Individual enablement or disabling of pin slew rate control.

Compilation occurs only when the `SLEW\_RATE\_CONTROL` macro is defined; currently, only Pin D1 is supported.

parameter:

- `idx`: Pin Numbering
- `enable`: `JX\_TRUE` enabled, `JX\_FALSE` dis-

abled; Return value: `JX\_OK` (0)

4. Matters Need Attention

1. **Configure the peripheral functions before using the peripheral device:** Before initializing peripherals such as UART, I2C, SPI, or PWM, you must first call the `jx\_pin\_function\_set()` function to configure the corresponding pins for peripheral use; otherwise, the signals will not be output correctly.

2. **Functional pin mapping varies depending on the Die:** The pin functional mapping for D0 and D1 differs; during configuration, refer to the corresponding `board/\$DX/soc/pin\_list.h` – these pins should not be used interchangeably.

3. **Strong pull-up is supported only on D1:** `PULL\_SELECT\_STRONG\_UP` This setting is valid only on the D1 pin; configuring this value on the D0 pin has no effect.

4. **GPIO Mode:** After configuring a pin for GPIO functionality, you must also use the GPIO driver interface (e.g., `jx\_gpio\_direction\_set`) to set the pin direction and level; refer to the GPIO driver documentation for details.

5. **Batch Initialization:** `jx\_pin\_init()` All pin initialization is completed during the board-level boot phase based on the BSP configuration; the application layer typically only manually calls the aforementioned interfaces when it needs to reuse pin functions at runtime.

7. 1. 4 DMA

1. Summary

`dma\_user` is a lightweight encapsulation layer driven by AXI DMA (located within `platform/components/dma\_user/`), which hides physical address translation and DMA channel lifecycle management from upper-layer applications. The caller simply provides the source address, destination address, and transmission size; the encapsulation layer handles the address translation (IPM) internally.

Virtual Address → Physical Address; Automatically Acquire an `auto\_free` DMA Channel; Initiate Transmission and Wait for Completion.

The Dynamically Allocated Channel Is Reserved for Internal Firmware Use.

- Encapsulation Layer Header File: `platform/include/dma\_user.h`
- Encapsulation layer implementation: `platform/components/dma\_user/dma\_user.c`
- Underlying DMA driver interface and type: `platform/device/include/dma.h`

Address Agreement

Address Range	Meaning	Process mode
32-bit address ($\leq 0xFFFFFFFF$)	Local IPM Virtual Address	Convert from `virt_to_phys()` to physical address
64-bit address ($> 0xFFFFFFFF$)	Physical address	Use directly

Channel Selection and Presets

The encapsulation layer obtains automatically released channels via the underlying `jx\_dma\_init\_user()` (a restricted version of `jx\_dma\_init` that skips the `auto\_free == 0` channel; the firmware itself uses `jx\_dma\_init` to access all channels).

The underlying `dma\_channel\_get()`, when operating on the `AXI\_DMA\_SINGLE\_MODE` branch, will prioritize the use of dedicated single-block channels; if no available single-block channels are available, it will automatically fallback to multi-block channels and operate in single-block register mode (without LLI configuration). The multi-block channels constitute a superset of the single-block channels; when LLI is not configured, this mode degenerates into a standard single-transmission operation. As a result, clusters such as c1/c3—which do not have dedicated single-block channels—can still perform single-block data transfer, all while remaining transparent to the upper layers.

- Single-block handling (`jx\_dma\_user\_copy`): The `AXI\_M2M\_32BIT\_SINGLE` is fixed in advance; the driving layer automatically switches back to the multi-block channel.
- Multiple-block transfer (`jx\_dma\_user\_copy\_blocks` / `\_async\_init\_blocks`): The `AXI\_M2M\_32BIT\_MULTI` is fixed and predefined; multiple-block channels must be used (single-block channels do not support LLI); no rollback is allowed.

Fixed 32-bit width with source/destination address auto-incrementing. The encapsulation layer internally manages the `cfg.core = 1` (Interrupt Binding Core0) and `cfg.prio`; the caller does not need to concern themselves with these details.

Transmission size and segmentation

- Single-call `size` upper limit: 4 GB (in `0xFFFFFFFF`; limited by `block\_cfg\_t.size` set to `jx\_u32\_t`).
- Transfers exceeding `AXI\_DMA\_SIZE\_MAX` (4*4 MB) are automatically segmented by the underlying AXI driver, with this process being transparent to the caller.
- Maximum number of blocks to handle: `DMA\_MULTI\_BLOCK\_SIZE` (default: 10).

2.

7.1.5 I2C

1. Data Type

I2C operating mode

```
enum { I2C_MODE_SLAVE = 0,  
       I2C_MODE_MASTER};
```

I2C peripheral address bit width

```
enum { I2C_DEV_ADDR_7BITS = 0,  
       I2C_DEV_ADDR_10BITS};
```

I2C peripheral register address bit width

```
enum { I2C_REG_ADDR_8BITS=0,  
       I2C_REG_ADDR_16BITS,  
       I2C_REG_ADDR_32BITS};
```

I2C peripheral register data bit width

```
enum{ I2C_REG_DATA_8BITS = 0,  
      I2C_REG_DATA_16BITS = 0x10,  
      I2C_REG_DATA_32BITS = 0x20};
```

i2c index of matrix

```
enum {  
    X100_D1_I2C0_INDEX = 0,  
    X100_D1_I2C1_INDEX,  
    X100_D1_I2C2_INDEX,  
    X100_D1_I2C3_INDEX,  
    X100_D1_I2C4_INDEX,  
    X100_D1_I2C5_INDEX,  
    X100_D1_I2C6_INDEX,  
    X100_D1_I2C7_INDEX,
```

```
X100_D1_I2C8_INDEX,  
X100_D1_I2C9_INDEX,  
X100_D1_I2C10_INDEX,  
X100_D1_I2C11_INDEX,  
X100_D1_I2C12_INDEX,  
X100_D1_I2C13_INDEX,  
X100_D1_I2C14_INDEX,  
X100_D1_I2C15_INDEX,  
X100_D0_I2C0_INDEX,  
X100_D0_I2C1_INDEX  
};
```

I2C initialization parameters

```
typedef struct {  
    jx_u8_t  mode;           /**i2c mode*/  
    jx_u8_t  addr_bit;       /**i2c addr bit*/  
    jx_u16_t  addr;          /**set target addr as controller, set address as slave*/  
    jx_u32_t  freq;          /**i2c clock frequency*/  
} i2c_config_t;
```

I2C interrupt initialization parameters

```
typedef struct {  
    jx_u8_t prio;            /**i2c irq prio*/  
    jx_u8_t cpu_mask;        /**irq bind cpu*/  
    jx_u8_t tx_threshold;    /**irq tx threshold*/  
    jx_u8_t rx_threshold;    /**irq rx threshold*/  
    jx_u8_t tx_cnt_per_irq;  /**tx count per irq*/  
    jx_u8_t reserved[3];  
} i2c_irq_config_t;
```

i2cdma initialization parameters

```
typedef struct {
```

```
jx_u8_t  type;                /**transfer type*/
jx_u8_t  channel_prio;        /**dma prio*/
jx_u8_t  threshold;          /**tx/rx dma threshold*/
jx_u8_t  cpu_mask;
jx_u8_t  irq_prio;
jx_bool_t aon_dma_en;
jx_u8_t  reserved[2];
} i2c_dma_config_t;
```

2. I²C Interface

I2C Initialization Interface

jx_err_t jx_i2c_init(jx_u8_t i2c_id, i2c_config_t *cfg);

Parameter description:

i2c_id: i2c index of matrix

cfg: I2C configuration parameters

I2C Data Transmission Interface

jx_err_t jx_i2c_transmit(jx_u8_t i2c_id, jx_u16_t device_addr, jx_u8_t *tx_data, jx_u16_t size, jx_u32_t timeout);

parameter declaration:

i2c_id: i2c index of matrix

device_addr: Device address

tx_data: Data transmission address

Size: Data transmission size

Timeout: Timeout duration (ms)

I2C Data Reception Interface

jx_err_t jx_i2c_receive(jx_u8_t i2c_id, jx_u8_t *rx_data, jx_u16_t size, jx_u32_t timeout);

parameter declaration:

i2c_id: i2c index of matrix

rx_data: Received data address

Size: Data reception size

Timeout: Timeout duration (ms)

I2C Peripheral Register Write Interface

jx_err_t jx_i2c_write(jx_u8_t i2c_id, jx_u16_t device_addr, jx_u32_t mem_addr, jx_u8_t addr_data_width, jx_void_t *tx_data, jx_u16_t size, jx_u32_t timeout);

parameter declaration:

i2c_id: i2c index of matrix
device_addr: Device address
mem_addr: Memory address
addr_data_width: Register address and data bit width
tx_data: Data transmission address
Size: Data transmission size
Timeout: Timeout duration (ms)

I2C Peripheral Register Read Interface

jx_err_t jx_i2c_read(jx_u8_t i2c_id, jx_u16_t device_addr, jx_u32_t mem_addr, jx_u8_t addr_data_width, jx_void_t *rx_data, jx_u16_t size, jx_u32_t timeout);

parameter declaration:

i2c_id: i2c index of matrix
device_addr: Device address
mem_addr: Memory address
addr_data_width: Register address and data bit width
rx_data: Received data address
Size: Data reception size
Timeout: Timeout duration (ms)

I2C AHB/AXI DMA initialization

jx_err_t jx_i2c_ahb_dma_init(jx_u8_t i2c_id, i2c_dma_config_t *cfg);

jx_err_t jx_i2c_axi_dma_init(jx_u8_t i2c_id, i2c_dma_config_t *cfg);

Parameter Description:

i2c_id: i2c index of matrix
cfg: I2C DMA configuration parameters

I2C Data Transmission Interface via DMA

jx_err_t jx_i2c_transmit_dma(jx_u8_t i2c_id, jx_u16_t device_addr, jx_u8_t *tx_data, jx_u16_t size, jx_u32_t timeout);

parameter declaration:

i2c_id: i2c index of matrix
device_addr: Device address
tx_data: Data transmission address

Size: Data transmission size

Timeout: Timeout duration (ms)

I2C data reception interface via DMA

jx_err_t jx_i2c_receive_dma(jx_u8_t i2c_id, jx_u8_t *rx_data, jx_u16_t size, jx_u32_t timeout);

parameter declaration:

i2c_id: i2c index of matrix

rx_data: Received data address

Size: Data reception size

Timeout: Timeout duration (ms)

I2C via DMA register write interface

jx_err_t jx_i2c_write_dma(jx_u8_t i2c_id, jx_u16_t device_addr, jx_u32_t mem_addr, jx_u8_t addr_data_width, jx_void_t *tx_data, jx_u16_t size, jx_u32_t timeout);

parameter declaration:

i2c_id: i2c index of matrix

device_addr: Device address

mem_addr: Memory address

addr_data_width: Register address and data bit width

tx_data: Data transmission address

Size: Data transmission size

Timeout: Timeout duration (ms)

I2C via DMA register read interface

jx_err_t jx_i2c_read_dma(jx_u8_t i2c_id, jx_u16_t device_addr, jx_u32_t mem_addr, jx_u8_t addr_data_width, jx_void_t *rx_data, jx_u16_t size, jx_u32_t timeout);

parameter declaration:

i2c_id: i2c index of matrix

device_addr: Device address

mem_addr: Memory address

addr_data_width: Register address and data bit width

rx_data: Received data address

Size: Data reception size

Timeout: Timeout duration (ms)

I2C interrupt initialization

jx_err_t jx_i2c_irq_init(jx_u8_t i2c_id, i2c_irq_config_t *cfg);

parameter declaration:

i2c_id: i2c index of matrix

cfg: I2C interrupt configuration parameters

I2C via Interrupt-Driven Data Transmission Interface

jx_err_t jx_i2c_transmit_irq(jx_u8_t i2c_id, jx_u16_t device_addr, jx_u8_t *tx_data, jx_u16_t size, jx_u32_t timeout);

parameter declaration:

i2c_id: i2c index of matrix

device_addr: Device address

tx_data: Data transmission address

Size: Data transmission size

Timeout: Timeout duration (ms)

I2C receives data via interrupts.

jx_err_t jx_i2c_receive_irq(jx_u8_t i2c_id, jx_u8_t *rx_data, jx_u16_t size, jx_u32_t timeout);

parameter declaration:

i2c_id: i2c index of matrix

rx_data: Received data address

Size: Data reception size

Timeout: Timeout duration (ms)

I2C: Writing a register via an interrupt

jx_err_t jx_i2c_write_irq(jx_u8_t i2c_id, jx_u16_t device_addr, jx_u32_t mem_addr, jx_u8_t addr_data_width, jx_void_t *tx_data, jx_u16_t size, jx_u32_t timeout);

parameter declaration:

i2c_id: i2c index of matrix

device_addr: Device address

mem_addr: Memory address

addr_data_width: Register address and data bit width

tx_data: Data transmission address

Size: Data transmission size

Timeout: Timeout duration (ms)

I2C: Read register via interrupt

jx_err_t jx_i2c_read_irq(jx_u8_t i2c_id, jx_u16_t device_addr, jx_u32_t mem_addr, jx_u8_t addr_data_width, jx_void_t *rx_data, jx_u16_t size, jx_u32_t timeout); Parameter

Description:

i2c_id: i2c index of matrix

device_addr: Device address
mem_addr: Memory address
addr_data_width: Register address and data bit width
rx_data: Received data address
Size: Data reception size
Timeout: Timeout duration (ms)

illustrate:

```
jx_u8_t g_i2c_id = 0;
jx_u8_t g_tar_addr = 0;
jx_u16_t g_reg_addr = 0;
i2c_config_t g_i2c_conf = {0};
i2c_irq_config_t g_i2c_irq_conf = {0};
i2c_dma_config_t g_i2c_dma_tx_conf = {0};
i2c_dma_config_t g_i2c_dma_rx_conf = {0};
jx_u8_t g_send_buf[8] = {0};
jx_u8_t g_recv_buf[8] = {0};

g_i2c_id = 2;
g_i2c_conf.mode = I2C_MODE_MASTER;
g_i2c_conf.freq = 100000;
jx_i2c_init(g_i2c_id, &g_i2c_conf);

s_tar_addr = 0x28;
s_reg_addr = 0x4;
s_send_buf[0] = 0xa;
jx_i2c_write(g_i2c_id, g_tar_addr, g_reg_addr,
I2C_REG_ADDR_8BITS|I2C_REG_DATA_8BITS, g_send_buf, 1, 1000);
jx_i2c_read(g_i2c_id, g_tar_addr, g_reg_addr,
I2C_REG_ADDR_8BITS|I2C_REG_DATA_8BITS, g_recv_buf, 1, 1000);

g_i2c_irq_conf.tx_threshold = 1;
g_i2c_irq_conf.rx_threshold = 1;
g_i2c_irq_conf.tx_cnt_per_irq = 1;
jx_i2c_irq_init(g_i2c_id, &g_i2c_irq_conf);
jx_i2c_write_irq(g_i2c_id, g_tar_addr, s_reg_addr,
I2C_REG_ADDR_8BITS|I2C_REG_DATA_8BITS, g_send_buf, 1, 1000);
jx_i2c_read_irq(g_i2c_id, g_tar_addr, s_reg_addr,
I2C_REG_ADDR_8BITS|I2C_REG_DATA_8BITS, g_recv_buf, 1, 1000);
```

```
g_i2c_dma_tx_conf.type = DMA_MTP_FC_DMA;  
g_i2c_dma_tx_conf.threshold = 1;  
jx_i2c_ahb_dma_init(g_i2c_id, &s_i2c_dma_tx_conf);
```

```
g_i2c_dma_rx_conf.type = DMA_PTM_FC_DMA;  
g_i2c_dma_rx_conf.threshold = 1;  
jx_i2c_ahb_dma_init(g_i2c_id, &s_i2c_dma_rx_conf);
```

```
jx_i2c_write_dma(g_i2c_id, s_tar_addr, s_reg_addr,  
I2C_REG_ADDR_8BITS|I2C_REG_DATA_8BITS, g_send_buf, 1, 1000);  
jx_u8_t *rx_buf = jx_malloc_align(32, 32);  
jx_i2c_read_dma(g_i2c_id, s_tar_addr, s_reg_addr,  
I2C_REG_ADDR_8BITS|I2C_REG_DATA_8BITS, rx_buf, 1, 1000);
```

7.1.6 SPI

1. Data Type

SPI data frame bit width

```
enum {  
    SPI_DATABITS_8BIT = 7,  
    SPI_DATABITS_16BIT = 15,  
    SPI_DATABITS_32BIT = 31  
};
```

spiwork pattern

```
enum { SPI_CPHA_LOW = 0,  
        SPI_CPHA_HIGH};  
enum { SPI_CPOL_LOW = 0,  
        SPI_CPOL_HIGH};
```

spi index of matrix

```
enum {  
    X100_D1_SPI0_INDEX = 0,  
    X100_D1_SPI1_INDEX,  
    X100_D1_SPI2_INDEX,  
    X100_D1_SPI3_INDEX,
```

```
X100_D1_SPI4_INDEX,  
X100_D1_SPI5_INDEX,  
X100_D1_SPI6_INDEX,  
X100_D0_SPI0_INDEX,  
X100_D0_SPI1_INDEX,  
X100_D0_SPI2_INDEX  
};
```

SPI initialization configuration parameters

```
typedef struct {  
    jx_u8_t  clk_phase;           /**spi clock phase*/  
    jx_u8_t  clk_polarity;       /**spi clock polarity*/  
    jx_u8_t  frame_bits;        /**spi data bit length*/  
    jx_u8_t  reserved;          /**spi rx delay*/  
    jx_u32_t clk_freq;          /**spi clock frequency*/  
} spi_config_t;
```

SPI interrupt initialization parameters

```
typedef struct {  
    jx_u8_t prio;                /**spi irq prio*/  
    jx_u8_t cpu_mask;           /**spi irq bind cpu*/  
    jx_u8_t tx_threshold;       /**irq tx threshold*/  
    jx_u8_t rx_threshold;       /**irq rx threshold*/  
    jx_u8_t tx_cnt_per_irq;     /**tx count per irq*/  
    jx_u8_t reserved[3];  
} spi_irq_config_t;
```

SPI DMA initialization parameters

```
typedef struct {  
    jx_u8_t  type;               /**transfer type*/  
    jx_u8_t  width;             /**data width*/  
    jx_u8_t  channel_prio;      /**dma channel prio*/  
    jx_u8_t  threshold;        /**tx/rx dma threshold*/  
    jx_u8_t  irq_prio;  
    jx_u8_t  cpu_mask;  
    jx_bool_t aon_dma_en;  
    jx_u8_t  reserved;  
} spi_dma_config_t;
```


2. SPI Interface

spiinitialise

jx_err_t jx_spi_init(jx_u8_t spi_id, spi_config_t *cfg);

parameter declaration:

spi_id: spi index of matrix

cfg: SPI configuration parameters

spisend data

jx_err_t jx_spi_transmit(jx_u8_t spi_id, jx_void_t *tx_data, jx_u16_t size, jx_u32_t timeout);

parameter declaration:

spi_id: spi index of matrix

tx_data: Data transmission address

Size: Data transmission size

Timeout: Timeout duration (ms)

SPI data reception

jx_err_t jx_spi_recieve(jx_u8_t spi_id, jx_void_t *rx_data, jx_u16_t size, jx_u32_t timeout);

parameter declaration:

spi_id: spi index of matrix

rx_data: Received data address

Size: Data reception size

Timeout: Timeout duration (ms)

SPI full-duplex data transmission and reception

jx_err_t jx_spi_transmit_receive(jx_u8_t spi_id, jx_void_t *tx_data, jx_void_t *rx_data, jx_u16_t size, jx_u32_t timeout);

parameter declaration:

spi_id: spi index of matrix

tx_data: Data transmission address

rx_data: Received data address

Size: Data transfer size

Timeout: Timeout duration (ms)

SPI write peripheral register data

```
jx_err_t jx_spi_write(jx_u8_t spi_id, jx_void_t *cmd_buf, jx_u8_t cmd_size, jx_void_t  
*tx_data, jx_u16_t tx_size, jx_u32_t timeout);
```

parameter declaration:

spi_id: spi index of matrix

cmd_buf: Command address

cmd_size: Command size

tx_data: Data transmission address

Size: Data transmission size

Timeout: Timeout duration (ms)

SPI reads peripheral register data

```
jx_err_t jx_spi_read(jx_u8_t spi_id, jx_void_t *cmd_buf, jx_u8_t cmd_size, jx_void_t  
*rx_data, jx_u16_t rx_size, jx_u32_t timeout);
```

parameter declaration:

spi_id: spi index of matrix

cmd_buf: Command address

cmd_size: Command size

rx_data: Received data address

Size: Data transmission size

Timeout: Timeout duration (ms)

SPI Interrupt Initialization

```
jx_err_t jx_spi_irq_init(jx_u8_t spi_id, spi_irq_config_t *cfg);
```

parameter declaration:

spi_id: spi index of matrix

cfg: SPI interrupt configuration parameter

SPI sends data via interrupts.

```
jx_err_t jx_spi_transmit_irq(jx_u8_t spi_id, jx_void_t *tx_data, jx_u16_t size, jx_u32_t  
timeout);
```

parameter declaration:

spi_id: spi index of matrix

tx_data: Data transmission address

Size: Data transmission size

Timeout: Timeout duration (ms)

SPI receives data via interrupts.

```
jx_err_t jx_spi_receive_irq(jx_u8_t spi_id, jx_void_t *rx_data, jx_u16_t size, jx_u32_t  
timeout);
```

parameter declaration:

spi_id: spi index of matrix

rx_data: Received data address

Size: Data reception size

Timeout: Timeout duration (ms)

SPI enables full-duplex data transmission and reception via interrupts.**jx_err_t jx_spi_transmit_receive_irq(jx_u8_t spi_id, jx_void_t *tx_data, jx_void_t *rx_data, jx_u16_t size, jx_u32_t timeout);****parameter declaration:**

spi_id: spi index of matrix

tx_data: Data transmission address

rx_data: Received data address

Size: Data transfer size

Timeout: Timeout duration (ms)

SPI: Write register via interrupt**jx_err_t jx_spi_write_irq(jx_u8_t spi_id, jx_void_t *cmd_buf, jx_u16_t cmd_size, jx_void_t *tx_data, jx_u16_t tx_size, jx_u32_t timeout);****parameter declaration:**

spi_id: spi index of matrix

cmd_buf: Command address

cmd_size: Command size

tx_data: Data transmission address

Size: Data transmission size

Timeout: Timeout duration (ms)

SPI reads the register via an interrupt.**jx_err_t jx_spi_read_irq(jx_u8_t spi_id, jx_void_t *cmd_buf, jx_u8_t cmd_size, jx_void_t *rx_data, jx_u16_t rx_size, jx_u32_t timeout);****parameter declaration:**

spi_id: spi index of matrix

cmd_buf: Command address

cmd_size: Command size

rx_data: Received data address

Size: Data transmission size

Timeout: Timeout duration (ms)

SPI AHB/AXI DMA initialization**jx_err_t jx_spi_ahb_dma_init(jx_u8_t spi_id, spi_dma_config_t *cfg);**

```
jx_err_t jx_spi_axi_dma_init(jx_u8_t spi_id, spi_dma_config_t *cfg);
```

Parameter description:

spi_id: spi index of matrix

cfg: SPI DMA configuration parameters

SPI sends data via DMA.

```
jx_err_t jx_spi_transmit_dma(jx_u8_t spi_id, jx_void_t *tx_data, jx_u16_t size,  
jx_u32_t timeout);
```

parameter declaration:

spi_id: spi index of matrix

tx_data: Data transmission address

Size: Data transmission size

Timeout: Timeout duration (ms)

SPI receives data via DMA.

```
jx_err_t jx_spi_receive_dma(jx_u8_t spi_id, jx_void_t *rx_data, jx_u16_t size, jx_u32_t  
timeout);
```

parameter declaration:

spi_id: spi index of matrix

rx_data: Received data address

Size: Data reception size

Timeout: Timeout duration (ms)

SPI performs full-duplex data transmission and reception via DMA.

```
jx_err_t jx_spi_transmit_receive_dma(jx_u8_t spi_id, jx_void_t *tx_data, jx_void_t  
*rx_data, jx_u16_t size, jx_u32_t timeout);
```

parameter declaration:

spi_id: spi index of matrix

tx_data: Data transmission address

rx_data: Received data address

Size: Data transfer size

Timeout: Timeout duration (ms)

SPI via DMA register write

```
jx_err_t jx_spi_write_dma(jx_u8_t spi_id, jx_void_t *cmd_buf, jx_u16_t cmd_size,  
jx_void_t *tx_data, jx_u16_t tx_size, jx_u32_t timeout);
```

parameter declaration:

spi_id: spi index of matrix

cmd_buf: Command address

cmd_size: Command size
tx_data: Data transmission address
Size: Data transmission size
Timeout: Timeout duration (ms)

SPI reads the register via an interrupt.

```
jx_err_t jx_spi_read_dma(jx_u8_t spi_id, jx_void_t *cmd_buf, jx_u16_t cmd_size,  
jx_void_t *rx_data, jx_u16_t rx_size, jx_u32_t timeout);
```

parameter declaration:

spi_id: spi index of matrix
cmd_buf: Command address
cmd_size: Command size
rx_data: Received data address
Size: Data transmission size
Timeout: Timeout duration (ms)

illustrate:

```
jx_u8_t g_spi_id = 0;  
jx_u8_t g_cmd_buf[8] = {0};  
jx_u8_t g_send_buf[8] = {0};  
jx_u8_t g_recv_buf[8] = {0};  
spi_config_t g_spi_conf = {0};  
spi_irq_config_t g_spi_irq_conf = {0};  
spi_dma_config_t g_spi_dma_tx_conf = {0};  
spi_dma_config_t g_spi_dma_rx_conf = {0};  
  
g_spi_id = 3;  
g_spi_conf.clk_phase = SPI_CPHA_LOW;  
g_spi_conf.clk_polarity = SPI_CPOL_LOW;  
g_spi_conf.frame_bits = SPI_DATABITS_8BIT;  
g_spi_conf.clk_freq = 1000000;  
jx_spi_init(g_spi_id, &g_spi_conf);  
  
g_cmd_buf[0] = 0x60;  
g_send_buf[0] = 0xa;  
jx_spi_write(g_spi_id, g_cmd_buf, 1, g_send_buf, 1, 1000);  
g_cmd_buf[0] = 0xe0;  
jx_spi_read(g_spi_id, g_cmd_buf, 1, g_recv_buf, 1, 1000);  
  
g_spi_irq_conf.tx_threshold = 1;
```

```
g_spi_irq_conf.rx_threshold = 1;
g_spi_irq_conf.tx_cnt_per_irq = 4;
jx_spi_irq_init(s_spi_id, &s_spi_irq_conf);

g_cmd_buf[0] = 0x60;
g_send_buf[0] = 0x5;
jx_spi_write_irq(g_spi_id, g_cmd_buf, 1, g_send_buf, 1, 1000);
g_cmd_buf[0] = 0xe0;
jx_spi_read_irq(g_spi_id, g_cmd_buf, 1, g_recv_buf, 2, 1000);

g_spi_dma_tx_conf.type = DMA_MTP_FC_DMA;
g_spi_dma_tx_conf.width = DMA_WIDTH_8BIT;
g_spi_dma_tx_conf.threshold = 2;
jx_spi_ahb_dma_init(g_spi_id, &g_spi_dma_tx_conf);

g_spi_dma_rx_conf.type = DMA_PTM_FC_DMA;
g_spi_dma_rx_conf.width = DMA_WIDTH_8BIT;
g_spi_dma_rx_conf.threshold = 1;
jx_spi_ahb_dma_init(g_spi_id, &g_spi_dma_rx_conf);

g_cmd_buf[0] = 0x60;
g_send_buf[0] = 0xa;
jx_spi_write_dma(g_spi_id, g_cmd_buf, 1, g_send_buf, 1, 1000);

jx_u8_t *rx_buf = jx_malloc_align(32, 32);
g_cmd_buf[0] = 0xe0;
jx_spi_read_dma(g_spi_id, g_cmd_buf, 1, rx_buf, 1, 1000);
```

7.2 USB Drive (@Li Li)

7.3 Flash Drive Interface, File System Interface

Drive-Layer Interface (does Not Distinguish Between NOR Flash and NAND Flash); the System Performs Automatic Identification During Boot-Up.

```
int jx_flash_erase(uint32 offset, size_t len);
```


The Erase Interface operates on a per-block basis and accepts two parameters along with one return value. The `offset` parameter specifies the address at which data should be erased; for example, for a block size of 23 (256 MB), `offset = 23 \* 128 \* 1024`. If the specified address is not aligned, the program uses `ROUND\_DOWN` to calculate this address; the `len` parameter is also calculated using `ROUND\_UP`. Even if the input address is not exactly 128 KB, it will be rounded up to 128 KB. A return value less than 0 indicates a failure of the erase operation.

```
ssize_t jx_flash_read(uint32_t offset, void *buffer, size_t len);
```

The read interface operates on a per-page basis; for example, with a 256 MB NAND flash memory, each page is 2 KB in size. The `offset` parameter follows the same convention as described above; the `buffer` is the memory space used to store the data read from the NAND flash; `len` specifies the size of the data to be read (minimum value: 2 KB). If the operation succeeds, the function returns the size of the data read; if it fails, it returns 0.

```
ssize_t jx_flash_write(uint32_t offset, void *buffer, size_t len);
```

Write to a storage interface; the operation is performed per page. For example, using a 256MB NAND flash memory where each page is 2K bytes, the `offset` parameter follows the same convention as described above, and the `buffer` represents the data storage space – place the data to be written into this buffer, then write the data from the buffer to the NAND flash memory. The `len` parameter specifies the size of the data to be written (minimum value: 2K bytes); if the operation succeeds, the function returns the size of the data read; if it fails, it returns 0.

```
int jx_flash_register(void);
```

The `RegisterFunction` function and Flash initialization return a value; if this value is less than 0, it indicates that the initialization failed.

```
void jx_flash_unregister(void);
```

Unregister function.

illustrate:

```
jx_flash_register();

uint32_t offset = 23*128*1024;

len = 128*1024;

char *buffer= jx_zalloc(len);

if (buffer == NULL)

    return -JX_NOMEM;

int ret;

memset(buffer, 0xa5, len);

jx_flash_erase(offset, len);

jx_flash_write(offset,buffer,len);

memset(buffer, 0xff, len);

jx_flash_read(offset,buffer,len);
```

Write the 128 KB-long data sequence with value 0xA5 to the Flash memory; then set the buffer value to 0xFF; finally, read the data back into the buffer to verify its correctness.

7.4 CSS DVFS Interface

7.4.1 Summary

The CSS (CSS Specific Computing Subsystem) on the X100 chip comprises the C0, C1, NPU, and 8 DSPs on D0, as well as the C2, C3, and 2 DSPs on D1. Each cluster supports independent frequency tuning and voltage regulation based on an arbitration mechanism (there are scenarios where multiple clusters share a single power domain, e.g., C0/C1, etc.).

The list of supported source clock frequencies and frequency division factors for each cluster is as follows:

cluster name	source clk	division ratio
C0/C1/D0_DSP0~7	26M/800M/1000M/1200M	1~16
C2/D1_DSP0~1	26M/600M/900M/1000M	1~16
C3	26M/400M	1~8

The recommended DVF tables for different clusters are as follows:

cluster name	recommend V-F value
C0/C1	0.75V-1200M/0.7V-1000M/0.65V-600M
D0_DSP0~7	0.75V-1000M/0.7V-800M/0.65V-00M
C2/D1_DSP0~1	0.8V-900M/0.7V-600M

As can be seen, although the design supports configuring a wide range of frequency values, the actual recommended DVFS values are limited to a few specific sets and have been rigorously converged through post-simulation analysis. If a customer wishes to configure other values, they should perform a self-test of voltage–frequency settings.

Additionally, as there is currently no demand for this functionality in practical applications, this document does not cover frequency or voltage modulation for the M3 and NPU.

7.4.2 Direction for Use

The user simply needs to include ``#include "css_perf_mgr.h" `` in the corresponding C file, and then call ``jx_css_perf_set`` or ``jx_css_perf_get``.

7.4.3 Data Types and API

```
typedef enum {
    CSS_PERF_DOMAIN_CPU_C0 = 0,          /* D0 C0 */
    CSS_PERF_DOMAIN_CPU_C1,              /* D0 C1 */
    CSS_PERF_DOMAIN_DSP0,                 /* D0 DSP
0 */
    CSS_PERF_DOMAIN_DSP1,                 /* D0 DSP
1 */
    CSS_PERF_DOMAIN_DSP2,                 /* D0 DSP
2 */
    CSS_PERF_DOMAIN_DSP3,                 /* D0 DSP
3 */
    CSS_PERF_DOMAIN_DSP4,                 /* D0 DSP
4 */
    CSS_PERF_DOMAIN_DSP5,                 /* D0 DSP
5 */
    CSS_PERF_DOMAIN_DSP6,                 /* D0 DSP
6 */
    CSS_PERF_DOMAIN_DSP7,                 /* D0 DSP
7 */
}
```

```
CSS_PERF_DOMAIN_D1_CPU,          /* D1 C2 */
CSS_PERF_DOMAIN_D1_DSP0,         /* D1 DSP 0 */
CSS_PERF_DOMAIN_D1_DSP1,         /* D1 DSP 1 */
CSS_PERF_DOMAIN_BUTT,
} jx_css_perf_domain_t---Points to indexes across different clusters
```

Interface for configuring voltage and frequency for a specified cluster

jx_css_perf_set(jx_css_perf_domain_t domain, jx_u32_t freq_mhz, jx_u32_t volt_mv)

parameter declaration:

Domain: Cluster Index, derived from jx_css_perf_domain_t enumeration

freq_mhz: Configured frequency value, unit: MHz

volt_mv: Configuration voltage value; unit: mV

Example code: Configure D0_DSP0 to operate at 0.7 V and 800 MHz.

```
jx_err_t err ;
err = jx_css_perf_set(CSS_PERF_DOMAIN_DSP0, 800, 700);
if (err == JX_OK) {
    return 0;
} else {
    return -1;
}
```

Interface for retrieving the current voltage and frequency of a specified cluster

jx_css_perf_get(jx_css_perf_domain_t domain, jx_u32_t *freq_mhz, jx_u32_t *volt_mv)

parameter declaration:

Domain: Cluster Index, derived from jx_css_perf_domain_t enumeration

freq_mhz: Retrieved frequency value, unit: MHz

volt_mv: Retrieved voltage value, unit: mV

Example code: Retrieve the current operating frequency and voltage of D1_C2.

```
jx_u32_t freq, volt;
err = jx_css_perf_get(CSS_PERF_DOMAIN_D1_CPU, &freq, &volt);
if (err == JX_OK) {
    return 0;
} else {
    return -1;
}
```

File System Interface

1. Call the `open` function, where `spinand1` is the partition name of the MTD device; after obtaining the device path, use it as the parameter for the `open` function. The second parameter of the `open` function is the attribute; the value provided below should be set to `rw` (readable and writable). A return value of 0 indicates success; a non-zero value indicates failure.

```
rt_device_t dev;  
  
dev = rt_device_find("spinand1");  
  
result = rt_device_open(dev, RT_DEVICE_OFLAG_RDWR);
```

Specifically, it calls `qspi\_mtd\_open` with the same parameters as above.

The `ioctl` interface function takes three parameters and returns a single value: it returns 0 on success and a non-zero value on failure. The first parameter is the `dev` obtained from the `rt\_device\_find` function; the second parameter is the `magic` value; the third parameter is an additional argument. The following example demonstrates two `ioctl` functions: the first function's parameters are retrieved from the driver layer, while the second function's parameters are passed from the application layer to the driver layer. The first function retrieves information about the NAND flash memory; the second function is a erasure function, with the minimum erasure unit being a block.

```
struct mtd_inifo_user info;  
  
result = rt_device_control(dev, MEMGETINFO, &info);
```

This `ioctl` function erases block 23; the block size is 128 KB (for a 256 MB NAND flash memory device), which can be retrieved using `info.eraser`.

```
uint32_t offset = (23*128*1024);  
  
int len = 128*1024;  
  
struct nand_erase_info einfo = {
```

```
.addr = ROUND_DOWN(offset, info.eraserize),  
.len =ROUND_UP(len, info.eraserize);  
.fail_addr = 0,  
};
```

```
result = rt_device_control(dev,ERASEBLOCK,&einfo);
```

The specific implementation calls the 'qspi_mtd_control' function; the parameters are the same as those mentioned above.

3. The 'write' function interface has four parameters and returns a single value; a return value of -1 (or 255) indicates failure, while a successful call returns the number of bytes written. The first 'dev' parameter is as described above; the second parameter specifies the offset of the address where data should be written; the third parameter is the data buffer – the data in this buffer is written to the address at the given offset; 'len' specifies the length of the single write operation (writing one page at a time); the exact length can be retrieved using 'info.writesize'.

```
result = rt_device_write(dev, offset, buffer,  
len);
```

The specific implementation calls the qspi_mtd_write function; the parameters are the same as above.

4. 'read' interface function: accepts four parameters and returns a single value; a return value of -1 (or 255) indicates failure, while a successful call returns the number of bytes written. This function reads data from the specified 'offset' address into the buffer.

```
result = rt_device_read(dev,offset,buffer, len);
```

The specific implementation calls the 'qspi_mtd_read' function; the parameters are the same as those mentioned above.

5. 'close' function.

```
rt_device_close(dev);
```


The specific implementation calls the `qspi\_mtd\_close` function; the parameters are the same as those mentioned above.

6. Test function example (256M NAND test case; specific parameters can be customized based on the actual NAND device; use `rt\_device\_control(dev, MEMGETINFO, &info)` to retrieve detailed information)

```
void test_mtd_fileoperations(void)
```

```
{
```

```
    rt_device_t dev;
```

```
    rt_err_t result;
```

```
    uint32_t offset = (23*128*1024);
```

```
    int len = 128*1024;
```

```
    struct mtd_info_user info;
```

```
    char *buffer = jx_zalloc(len);
```

```
    if(buffer == NULL)
```

```
        return -JX_NOMEM;
```

```
    memset(buffer,0x5a,len);
```

```
    dev= rt_device_find("spinand1");
```

```
    if(!dev)
```

```
        return -EIO;
```

```
    result = rt_device_control(dev, MEMGETINFO, &info);
```

```
    if(result != RT_EOK)
```

```
        return -EIO;
```

```
struct nand_erase_info einfo = {  
    .addr = ROUND_DOWN(offset, info.erasesize),  
    .len  =ROUND_UP(len, info.erasesize);  
    .fail_addr = 0,  
};  
result = rt_device_control(dev,ERASEBLOCK,&einfo);  
if(result != RT_EOK)  
    return -EIO;  
result = rt_device_write(dev, offset, buffer, len);  
if(result != RT_EOK)  
    return -EIO;  
  
memset(buffer,0,len);  
result = rt_device_read(dev, offset, buffer, len);  
if(result != RT_EOK)  
    return -EIO;  
  
rt_device_close(dev)  
}
```

8. DFT Tool

8.1 Heartbeat Kit (@Wen Yikai@Xu Qunqi@Wang Chunlong)

Each data description in the Heartbeat Pack interface.

8.2 SystemView (@Han Dahan)

On the PC, you can use the official SEGGER System View tool: [Verify and validate embedded system design | SEGGER SystemView](#). It is recommended to use version 4.10a or later.

8.2.1 Usage Mode

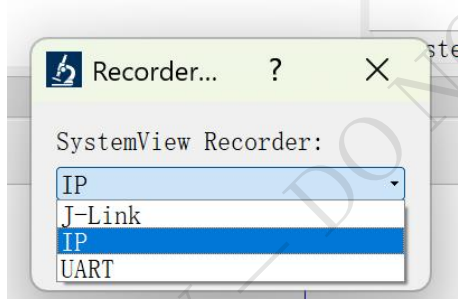
For information on using RT-Thread, please refer to the following URL; however, this example assumes a J-Link connection method, whereas the X100 uses an IP-based connection method – the approach is slightly different.

<https://www.rt-thread.org/document/site/#/rt-thread-version/rt-thread-standard/application-note/debug/systemview/an0009-systemview?id=systemview-%e9%85%8d%e7%bd%ae%e5%8f%8a%e4%bd%bf%e7%94%a8>

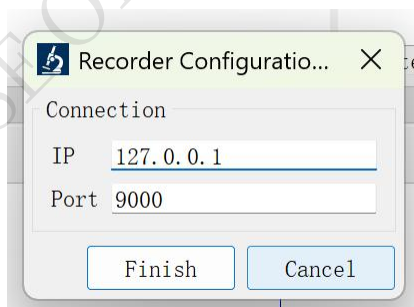
How to use the X100:

- 1、download software: [SEGGER – The Embedded Experts – Downloads – SystemView](#)
- 2、Connect the PC to the USB 3.0 interface on the X100 hardware platform using a USB 3.0 cable.
- 3、Launch the SystemView tool software. If you encounter an error indicating a missing Windows dynamic-link library, you need to install the Microsoft Visual C++ Redistributable Package. You can find the appropriate version to install at <https://learn.microsoft.com/zh-cn/cpp/windows/latest-supported-vc-redist?view=msvc-170>, then restart the SystemView tool.

- 4、 In the menu bar, click "Target → Recorder Configuration"; once the pop-up window appears, select the IP mode:

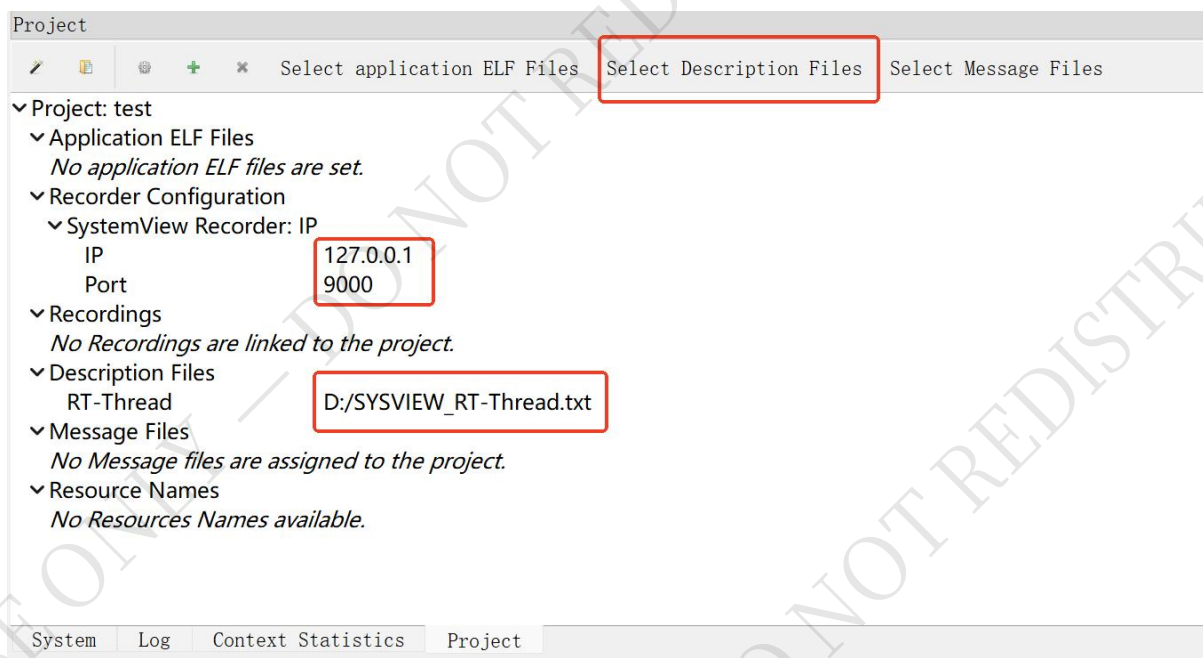


- 5、 Click "OK" to open the next "Connection" dialog box; select IP address 127.0.0.1, then choose one of the four port numbers (9000,9001,9002, or 9003) and enter it:



The port number corresponds to a specific CPU cluster (note: this refers to the CPU cluster, not individual CPU cores; each cluster can contain multiple cores).

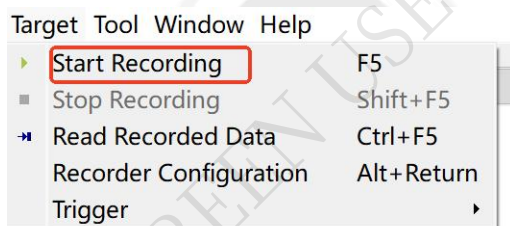
- 6、 After filling it out, the following will appear on the right-hand side of the Project page:



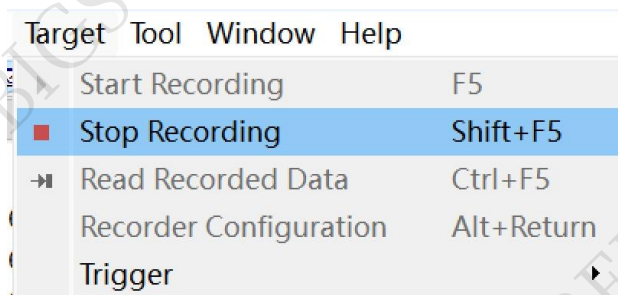
7、The file SYSVIEW_RT-Thread.txt shown above can be downloaded from the following URL:

https://github.com/RT-Thread-packages/SEGGER_SystemView/tree/master/SystemView_Description

8、After powering on the X100 hardware, select "Target → Start Recording" from the menu to begin recording SystemView event information:



9、Stop Recording: Click "Target → Stop Recording"



10、 Note: Recorded data is not automatically saved; you must manually click "File → Save Recording" within the tool software.

8.2.2 Implemented Features

1. Supports IP-based recording.
2. Supports continuous recording.
3. You can record events individually for a single cluster (C0, C1, or C2).
4. You can simultaneously record events from the C0, C1, and C2 clusters. To do this, launch multiple SystemView clients.

8.2.3 Known Constraints

1. Recording using J-link or UART mode is not supported.
2. Single-slot and post-mortem recording are not supported.
3. It is not possible to determine the origin of different CPU cores on each cluster.

8.2.4 Reference Material

SEGGER Official Website: [Verify and validate embedded system design | SEGGER SystemView](#)

Official Wiki: [SystemView-SEGGER Knowledge Base](#)

official document: [SEGGER SystemView User Guide](#)— Official Target Code (excluding the RT-Thread implementation): [SEGGERMicro/SystemView: SEGGER SystemView target sources, configurations, and RTOS patches](#). Porting target code to RT-Thread: [RT-Thread-packages/SEGGER SystemView: SEGGERSystemView](#)