StarFive 赛昉科技

JH7110 ISP Developing and Porting Guide

Version: 1.0

Date: 2022/12/30

Doc ID: JH7110-PGEN-003

Legal Statements

Important legal notice before reading this documentation.

PROPRIETARY NOTICE

Information in this document is provided "as is," with all faults. Contents may be periodically updated or revised due to product development. Shanghai StarFive Technology Co., Ltd. (hereinafter "StarFive") reserves the right to make changes without further notice to any products herein.

StarFive expressly disclaims all warranties, representations, and conditions of any kind, whether express or implied, including, but not limited to, the implied warranties or conditions of merchantability, fitness for a particular purpose, and non-infringement.

StarFive does not assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation indirect, incidental, special, exemplary, or consequential damages.

Contact Us

Address: Room 502, Building 2, No. 61 Shengxia Rd., China (Shanghai) Pilot Free Trade Zone, Shanghai, 201203, China

Website: http://www.starfivetech.com

Email:

- Sales: sales@starfivetech.com
- Support: support@starfivetech.com

Preface

About this guide and technical support information.

About this document

This document mainly provides the SDK developers and porting administrators with the programing basics and debugging know-how for the ISP and camera module of the StarFive next generation SoC platform - JH7110.

Revision History

Table 0-1 Revision History

Version	Released	Revision
1.0	2022/12/30	First official release.

Notes and notices

The following notes and notices might appear in this guide:

- **Tip:**
  Suggests how to apply the information in a topic or step.
- **Note:**
  Explains a special case or expands on an important point.
- **Important:**
  Points out critical information concerning a topic or step.
- **CAUTION:**
  Indicates that an action or step can cause loss of data, security problems, or performance issues.
- **Warning:**
  Indicates that an action or step can result in physical harm or cause damage to hardware.

Contents

List of Tables

List of Figures

1. Introduction

The document provides basic information about the *Image Signal Processing (ISP)* block in the StarFive SoC JH7110 and the associated ISP drivers developed under Linux OS. Driver architecture and related APIs are explained with examples and details.

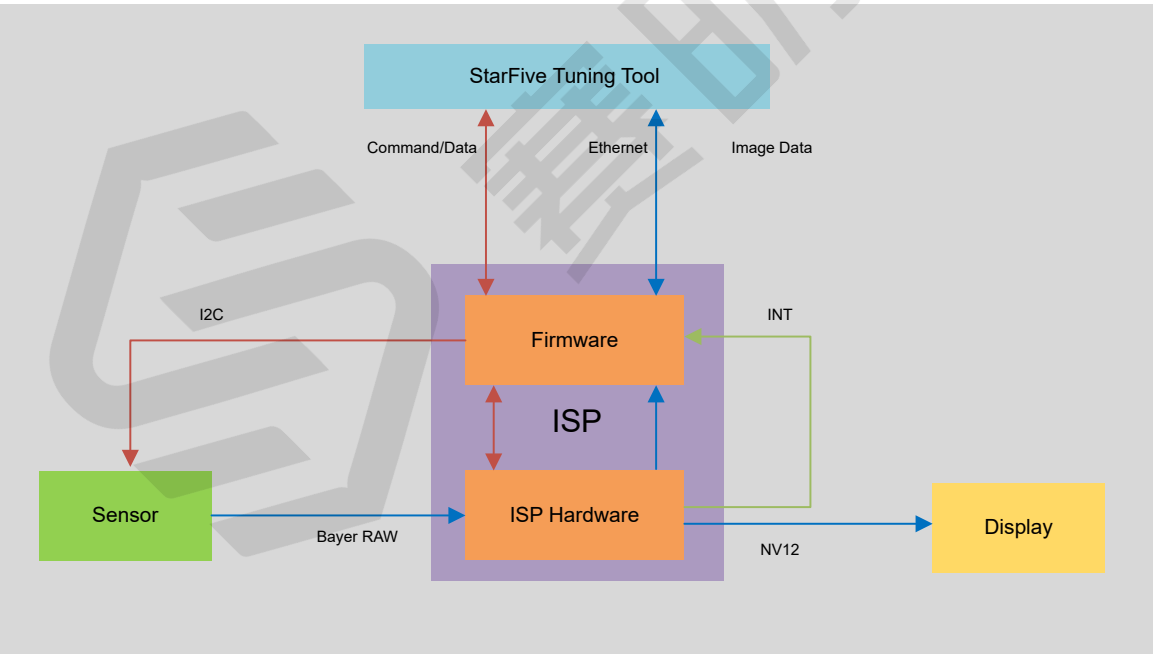
The *Image Signal Processor (ISP)* processes digital images by using a series of digital image processing algorithms. The ISP supports the following functions: 3A algorithm, black level correction, lens shading correction, gamma correction, local tone mapping, edge enhancement, noise reduction, and color enhancement. The ISP consists of the logic and firmware running on the logic.

1.1. Function Introduction

The ISP processor provides algorithms and real-time image information feedback. Since some parameters of the algorithms are initially uncertain, the algorithms must work with adaptive control methods in firmware. The adaptive control method obtains the image information, recalculates related parameter values such as the exposure time and gain, and estimates the parameter values required by the next frame to control the sensor, and ISP logic. This ensures that the image quality is automatically adjusted.

The following diagram shows the operating mode of the ISP. The light signals onto the photosensitive area of the sensor. After photoelectric conversion, the sensor transmits Bayer raw images to the ISP. The ISP processes the images based on related algorithms and outputs the images in the Bayer/YUV spatial domain to the firmware and display module. During this process, the ISP controls the sensor by using the firmware and implementing the functions including *Automatic Exposure (AE)*, and *Automatic White Balance (AWB)*. The firmware is driven by the interrupts of the ISP module. The StarFive Tuning Tools adjust the quality of the online images of the ISP over the Ethernet port.

Figure 1-1 ISP Operation Mode

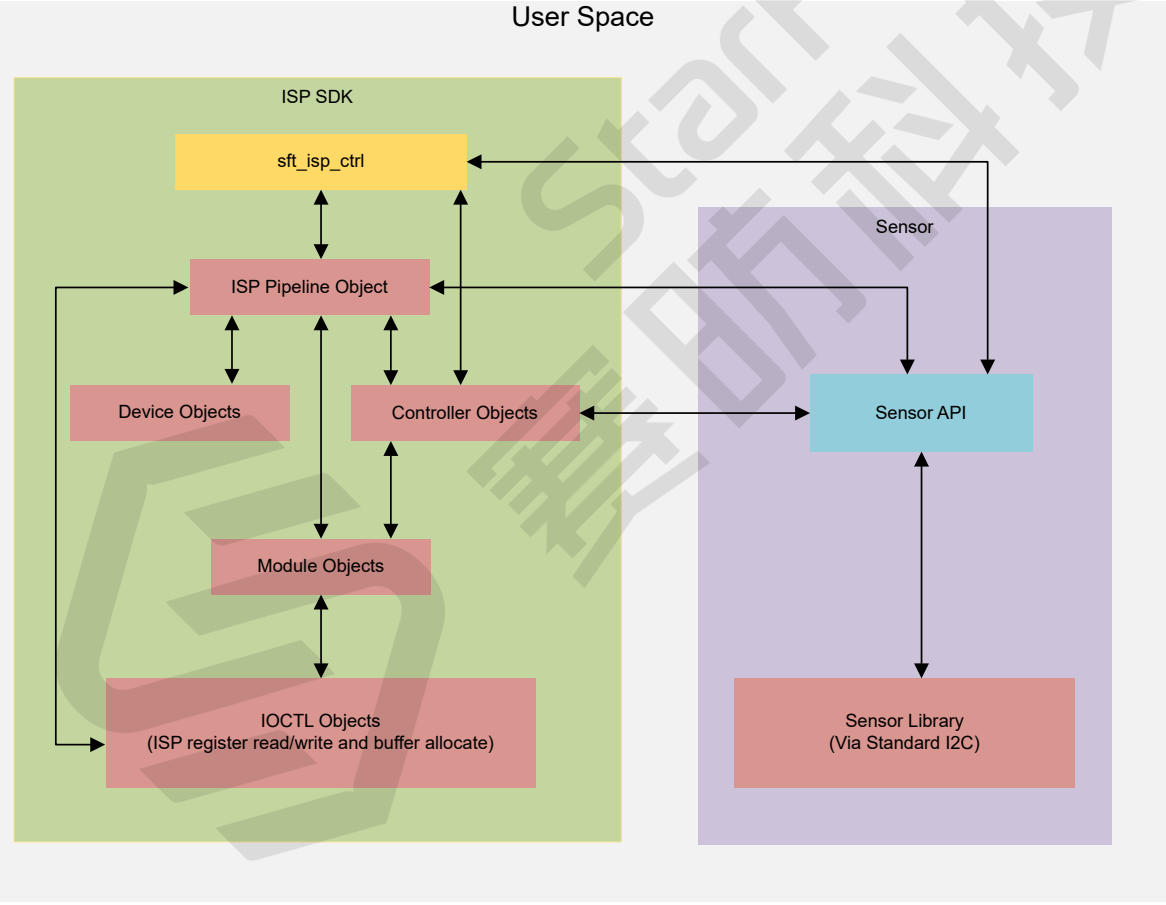


1.2. Architecture

In the OS system, the ISP firmware consists of the kernel driver and user mode ISP SDK. In this document, we only discuss the ISP SDK part. The ISP SDK consists of the IOCTL library, device library, control library, module library, sensor library, pipeline library, and ISP flow control application.

- The IOCTL library provides the ISP register read/write and buffer allocate/release functions to service the module unit.
- The device objects provide the basic device operations. For example, **open/close/ioctl/init/allocate buffer/release buffer**, etc.
- The control objects include the 3A algorithm and other basic algorithms for adaptive control. For example, AE/AWB/SAT, etc. These adaptive controls are based on the ISO and color temperature.
- The module objects provide the parameter conversion and the ISP register program.
- The sensor library provides the exposure time and sensor gain calculation and program.
- The pipeline objects provides the device/control/module management.
- The "**stf_isp_ctrl**" application monitors the video device stream on/off and establishes/tears down the pipeline for the ISP adaptive control.

Figure 1-2 ISP SDK Architecture



1.3. StarFive ISP Control Workflow

The ISP SDK is a high-level API and it encapsulates the functionality for interacting with the ISP, and provides a convenient method to handle and control ISP and sensor behaviors, such as:

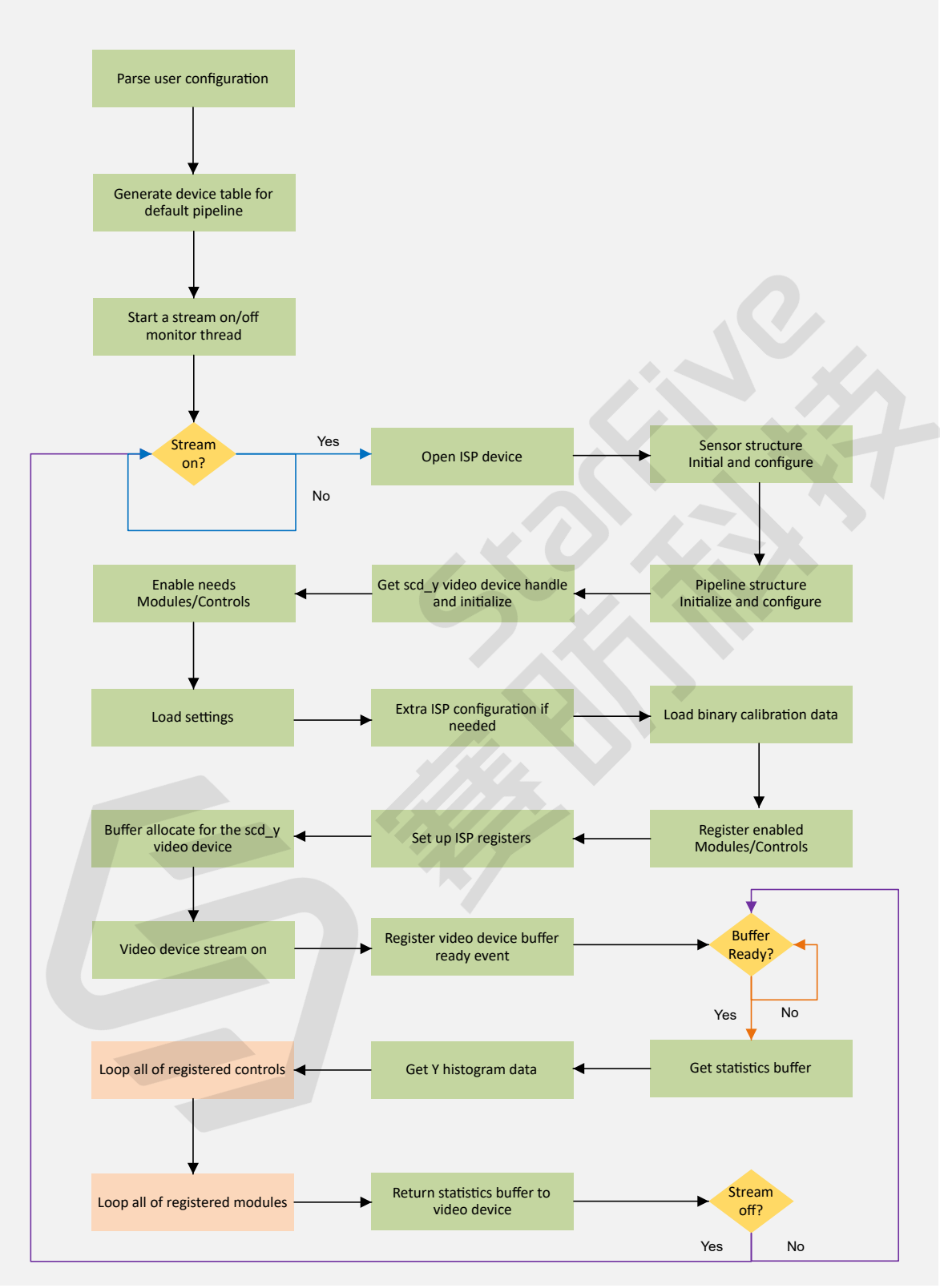
- Video device management.
- Buffer allocations.
- Image shot management.
- Streaming control.
- IOCTL command.
- Module configuration.
- Adaptive control.
- Sensor control, etc.

Besides the SDK, StarFive also provides the application of **stf_isp_ctrl**, namely the StarFive ISP Control. The application invokes the ISP SDK API to make the ISP hardware keep processing every stream automatically. It uses the SDK library to operate the ISP mainly in the following steps:

1. Fetch the statistical collection data.
2. Pass the statistics data to registered controls for the adaptive control.
3. Loop registered modules to convert the high-level parameters to ISP register values.
4. Send these register values to kernel driver through the IOCTL command.
5. Program the ISP.
6. Return the statistics buffer to video device after completing the registers update.

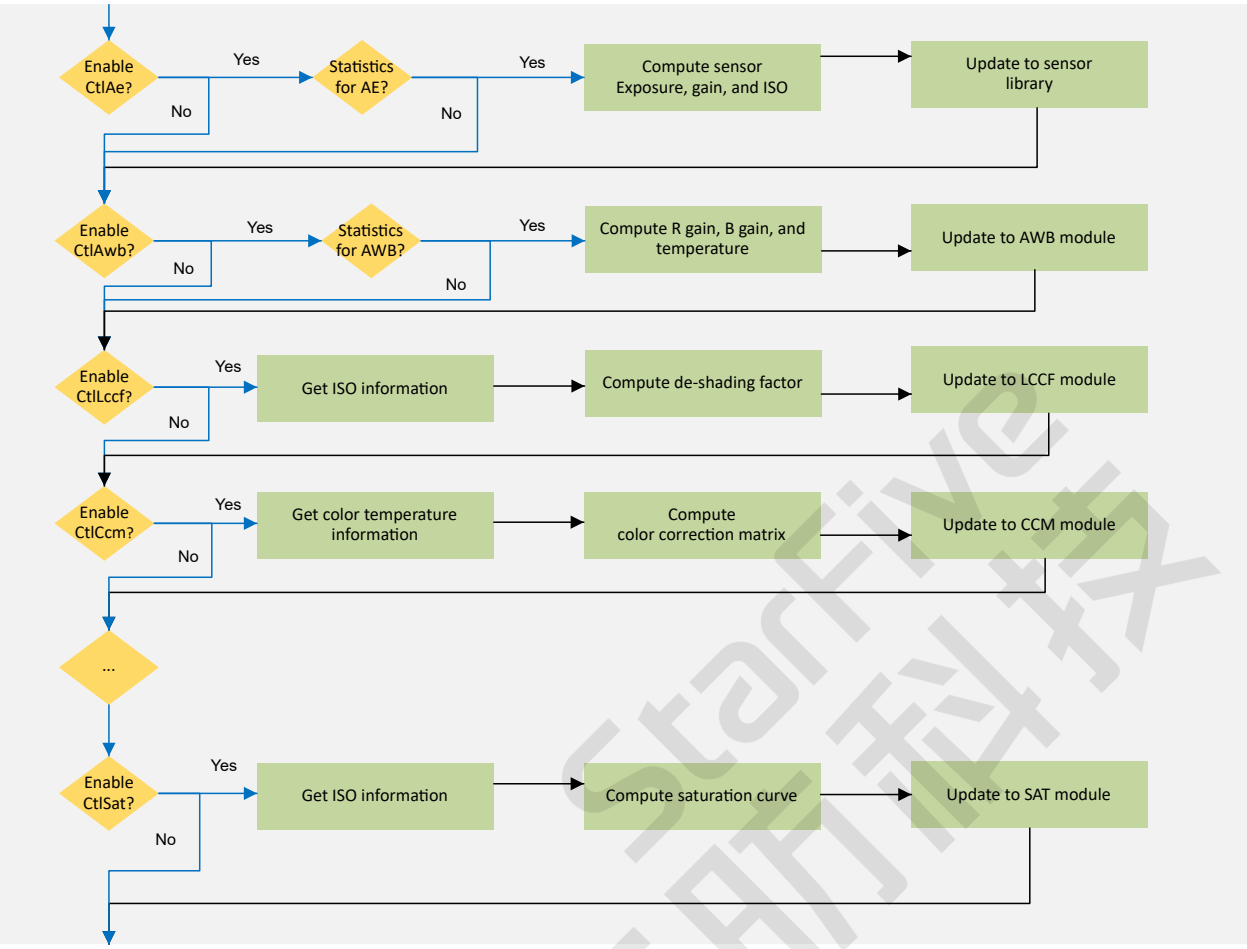
The following diagram shows the standard StarFive ISP control workflow.

Figure 1-3 StarFive ISP Control Workflow



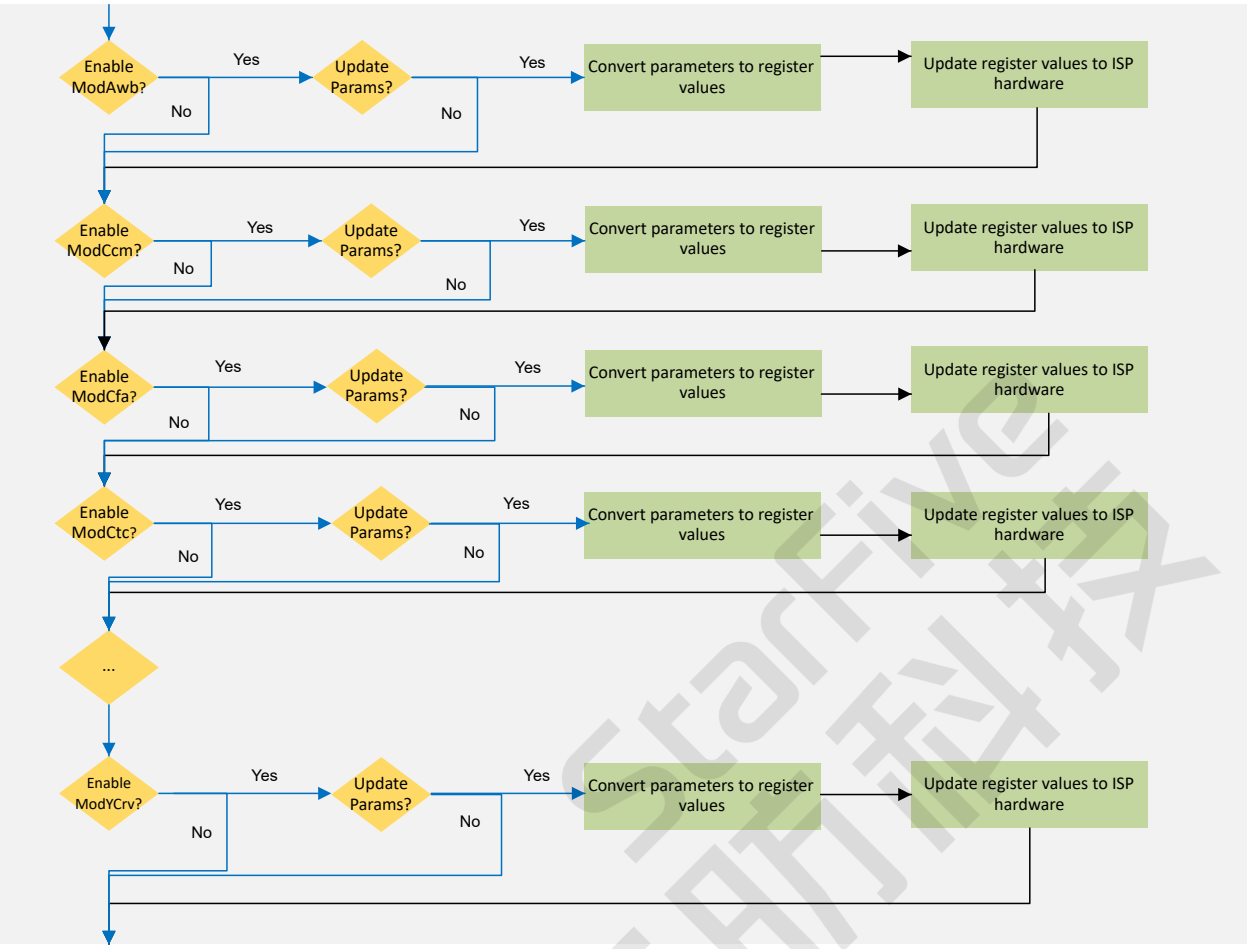
The following diagram shows the loop of all registered controls.

Figure 1-4 Loop of All Registered Controls



The following diagram shows the loop of all registered modules.

Figure 1-5 Loop of All Registered Modules

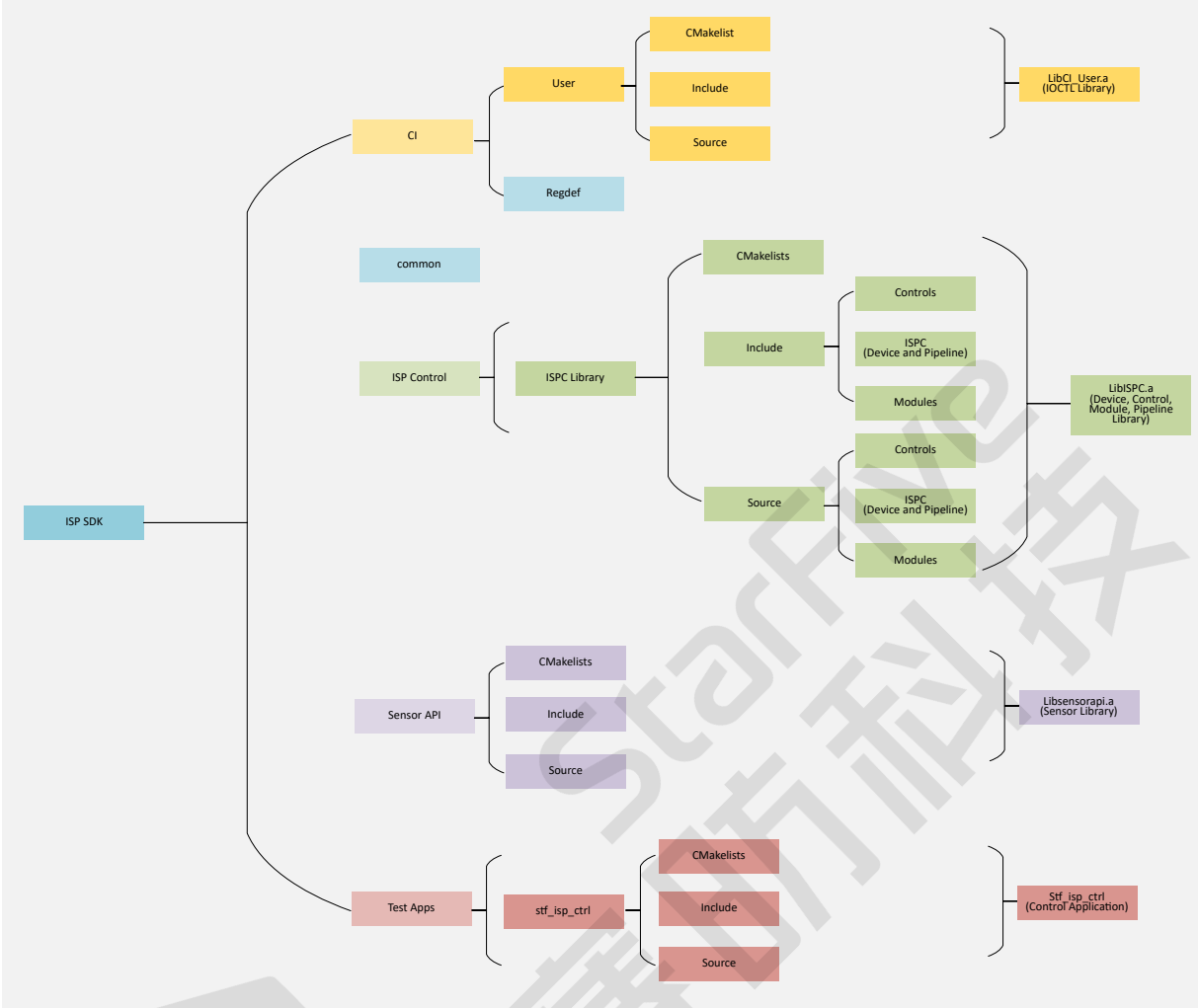


1.4. File Structure

The following diagram shows the file structure of the StarFive ISP SDK.

- The file `libCI_User.a` contains the IOCTL objects.
- The file `libISPC.a` contains the device, control, module, and pipeline objects.
- The file `libSensorapi.a` is the sensor library.
- The application **stf_isp_ctrl** is the ISP control application.

Figure 1-6 File Structure



2. Sensor Library - High Level API and Structure

In the ISP SDK, the processing image comes from the sensor. Therefore, it is very important for the sensor to provide a high quality image. The sensor API consists of many sensor functions, and the ISP SDK integrates these functions into a sensor structure to provide a simple method to operate the sensor. Here we will focus on this structure and necessary API functions.

2.1. Data Structure

The following topic shows the data structure of Sensor Library.

2.1.1. Sensor

The following code block shows the data structure for the sensor structure.

```
typedef struct _ST_SENSOR {
    SENSOR_HANDLE *pstSensorHandle;
    EN_SENSOR_STATE enState;
    STF_U16 u16Width;
    STF_U16 u16Height;
    STF_U8 u8Imager;
    STF_U16 u16VerticalTotal;
    enum MOSAICType enBayerFormat;
    STF_U8 u8SensorContexts;
    STF_U8 u8BitDepth;
    STF_DOUBLE dFrameRate;
    STF_DOUBLE dCurrentFPS;
    STF_U32 u32WellDepth;
    STF_DOUBLE dReadNoise;
    STF_DOUBLE dAperture;
    STF_U16 u16FocalLength;
    STF_U32 u32ProgrammedExposure;
    STF_DOUBLE dProgrammedGain;
    STF_U16 u16ProgrammedFocus;
    STF_U32 u32MinExposure;
    STF_U32 u32MaxExposure;
    STF_U16 u16MinFocus;
    STF_U16 u16MaxFocus;
    STF_DOUBLE dMinGain;
    STF_DOUBLE dMaxGain;
    STF_BOOL8 bFocusSupported;

    const STF_CHAR* (*StateName)(EN_SENSOR_STATE enState);
    STF_VOID (*GetSensorNames)(STF_U8 *pu8Count, STF_CHAR szSensorName[][64]);
    STF_S8 (*GetSensorId)(const char *SensorName);
    STF_RESULT (*Init)(ST_SENSOR *pstSensor, STF_U8 u8SensorId, STF_U8 u8Index);
    STF_RESULT (*Destroy)(ST_SENSOR *pstSensor);
    SENSOR_HANDLE *(*GetHandle)(ST_SENSOR *pstSensor);
    EN_SENSOR_STATE (*GetState)(ST_SENSOR *pstSensor);
    STF_RESULT (*GetInfo)(ST_SENSOR *pstSensor, SENSOR_INFO *pstSensorInfo);
    STF_S8 (*GetModeIdx)(ST_SENSOR *pstSensor, STF_U16 u16Width, STF_U16 u16Height);
    STF_RESULT (*GetMode)(ST_SENSOR *pstSensor, STF_U8 u8ModeIdx, SENSOR_MODE *pstModes);
    STF_RESULT (*SetMode)(ST_SENSOR *pstSensor, STF_U8 u8ModeIdx, STF_U8 u8Flipping);
    STF_RESULT (*GetInterfaceInfo)(ST_SENSOR *pstSensor, SENSOR_INTFC *pstInterface);
    STF_RESULT (*Configure)(ST_SENSOR *pstSensor, STF_U8 u8ModeIdx, STF_U8 u8Flipping);
    STF_RESULT (*Insert)(ST_SENSOR *pstSensor);
    STF_RESULT (*WaitProcessed)(ST_SENSOR *pstSensor);
    STF_RESULT (*Enable)(ST_SENSOR *pstSensor);
    STF_RESULT (*Disable)(ST_SENSOR *pstSensor);
    STF_RESULT (*Reset)(ST_SENSOR *pstSensor);
    STF_RESULT (*GetStatus)(ST_SENSOR *pstSensor, SENSOR_STATUS *pstStatus);
    STF_RESULT (*GetReg)(ST_SENSOR *pstSensor, STF_U16 u16RegAddr, STF_U16 *pu16RegValue);
    STF_RESULT (*SetReg)(ST_SENSOR *pstSensor, STF_U16 u16RegAddr, STF_U16 u16RegValue);
    STF_RESULT (*SetFlipMirror)(ST_SENSOR *pstSensor, STF_U8 u8Flag);
    STF_RESULT (*GetFixedFPS)(ST_SENSOR *pstSensor, STF_U16 *pu16FixedFps);
    STF_RESULT (*SetFPS)(ST_SENSOR *pstSensor, STF_DOUBLE dFps);
```

```
    STF_RESULT (*GetSnapShotResolution)(ST_SENSOR *pstSensor, ST_RES_LIST *pstResList);
    STF_RESULT (*SetResolution)(ST_SENSOR *pstSensor, STF_U16 u16ImgW, STF_U16 u16ImgH);
    STF_U32 (*GetMinExposure)(ST_SENSOR *pstSensor);
    STF_U32 (*GetMaxExposure)(ST_SENSOR *pstSensor);
    STF_RESULT (*GetExposureRange)(ST_SENSOR *pstSensor, STF_U32 *pu32MinExposure, STF_U32 *pu32MaxExposure);
    STF_U32 (*GetExposure)(ST_SENSOR *pstSensor);
    STF_RESULT (*SetExposure)(ST_SENSOR *pstSensor, STF_U32 u32Exposure);
    STF_DOUBLE (*GetMinGain)(ST_SENSOR *pstSensor);
    STF_RESULT (*SetMinGain)(ST_SENSOR *pstSensor, STF_DOUBLE dMinGain);
    STF_DOUBLE (*GetMaxGain)(ST_SENSOR *pstSensor);
    STF_RESULT (*SetMaxGain)(ST_SENSOR *pstSensor, STF_DOUBLE dMaxGain);
    STF_RESULT (*GetGainRange)(ST_SENSOR *pstSensor, STF_DOUBLE *pdMinGain, STF_DOUBLE *pdMaxGain);
    STF_RESULT (*SetIsoLimit)(ST_SENSOR *pstSensor, STF_U8 u8IsoLimit);
    STF_DOUBLE (*GetGain)(ST_SENSOR *pstSensor);
    STF_RESULT (*SetGain)(ST_SENSOR *pstSensor, STF_DOUBLE dGain);
    STF_RESULT (*SetExposureAndGain)(ST_SENSOR *pstSensor, STF_U32 u32Exposure, STF_DOUBLE dGain);
    STF_BOOL8 (*GetFocusSupported)(ST_SENSOR *pstSensor);
    STF_U16 (*GetMinFocus)(ST_SENSOR *pstSensor);
    STF_U16 (*GetMaxFocus)(ST_SENSOR *pstSensor);
    STF_U16 (*GetFocusDistance)(ST_SENSOR *pstSensor);
    STF_RESULT (*SetFocusDistance)(ST_SENSOR *pstSensor, STF_U16 u16FocusDistance);
} ST_SENSOR, *PST_SENSOR;
```

The following table describes the fields in the above code block.

Table 2-1 Sensor Field Description

Field	Description
pstSensorHandle	Sensor API handle
enState	Sensor running state
u16Width	Output image width
u16Height	Output image height
u8Imager[1]	Imager ID. No longer in use
u16VerticalTotal	Number of lines including blanking
enBayerFormat	Mosaic of the sensor
u8SensorContexts[1]	Sensor contexts. No longer in use
u8BitDepth	Bits per pixel
dFrameRate	Sensor default frame rate
dCurrentFPS	Frame rate programmed in the sensor
u32WellDepth[1]	Number of electrons can be held by the sensor wells before clipping. No longer in use
dReadNoise[1]	Standard deviation of noise when reading pixel value off a sensor in electrons. No longer in use
dAperture[1]	Aperture f/# of the attached lens. No longer in use
u16FocalLength[1]	Focal length of the lens in millimetres. No longer In use
u32ProgrammedExposure	Exposure time programmed in the sensor
dProgrammedGain	Gain programmed in the sensor
u16ProgrammedFocus[1]	Focus distance set in the sensor. No longer in use
u32MinExposure	Minimum exposure times
u32MaxExposure	Maximum exposure times
u16MinFocus[1]	Minimum focus distance. No longer in use

Table 2-1 Sensor Field Description (continued)

Field	Description
u16MaxFocus[1]	Maximum focus distance. No longer in use
dMinGain	Minimum exposure gain
dMaxGain	Maximum exposure gain
bFocusSupported[1]	Variable focus support status. No longer in use
***StateName**	Get sensor state name
***GetSensorNames**	Populate a list with the available sensors from Sensor API
***GetSensorId**	Get sensor ID from a name
***Init**	Sensor structure initialization
***Destroy**	Destroy a sensor through a given identifier
***GetHandle**	Access to sensors API handle
***GetState**	Get the sensor state
***GetInfo**	Get the sensor information
***GetModeIdx**	Get the sensor mode index
***GetMode**	Get some mode information (including if flipping is supported)
***SetMode**	Set the sensor mode
***GetInterfaceInfo**	Get the sensor connection data bus and I2C hardware information
***Configure**	Configure the sensor with a selected mode
***Insert**[1]	Access to the Sensor API insert function. No longer in use
***WaitProcessed**[1]	Access to the Sensor API **waitProcessed** function. No longer in use
***Enable**	Start transmission of the data from the sensor. In the V4L2 driver, only changed the sensor state
***Disable**	Stop transmission of the data from the sensor. In the V4L2 driver, only changed the sensor state
***Reset**	Reset the sensor. In the V4L2 driver, only changed the sensor state
***GetStatus**	Get status value in the sensor
***GetReg**	Get the sensor register value
***SetReg**	Program the value to the register of sensor
***GetFlipMirror**	Get flip and mirror value in the sensor
***SetFlipMirror**	Set flip and mirror value in the sensor
***GetFixedFPS**	Get the fixed FPS value in the sensor
***SetFPS**	Set the FPS value in the sensor
***GetSnapShotResolution**[1]	Get the snapshot resolution value in the sensor. No longer in use
***SetResolution**[1]	Set the resolution value to the sensor. No longer in use
***GetMinExposure**	Get minimum exposure time value programmable in the sensor

Table 2-1 Sensor Field Description (continued)

Field	Description
***GetMaxExposure**	Get maximum exposure time value programmable in the sensor
***GetExposureRange**	Get minimum and maximum exposure time value programmable in the sensor
***GetExposure**	Get the exposure time programmed in the sensor
***SetExposure**	Program the exposure time in the sensor
***GetMinGain**	Get minimum gain value programmable in the sensor
***SetMinGain**	Set the minimum gain value programmable in the sensor
***GetMaxGain**	Set the minimum gain value programmable in the sensor
***SetMaxGain**	Set the maximum gain value programmable in the sensor
***GetGainRange**	Get minimum and maximum gain value programmable in the sensor
***SetIsoLimit**	Set minimum and maximum gain value programmable in the sensor according to the ISO value
***GetGain**	Get the gain value from the sensor
***SetGain**	Set the gain value in the sensor
***SetExposureAndGain**	Program the exposure time and gain in the sensor
***GetFocusSupported**[1]	Get state of driver support for variable focus. No longer in use
***GetMinFocus**[1]	Get minimum focus distance. No longer in use
***GetMaxFocus**[1]	Get maximum focus distance. No longer in use
***GetFocusDistance**[1]	Get the current focus distance. No longer in use
***SetFocusDistance**[1]	Set focus distance in the sensor in millimetre. No longer in use

Note:

1. The field items are no longer in use.

2.2. Sensor API

The following topics show the application interfaces (API) of Sensor Library.

2.2.1. STFLIB_ISP_SENSOR_GetSensorId

The API is described in the following sections.

Description

Get sensor ID from sensor name.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_SENSOR_GetSensorId(
    const STF_CHAR *pszSensorName
    );
```

Parameter

The API has the following parameters.

Table 2-2 STFLIB_ISP_SENSOR_GetSensorId Parameter Description

Parameter	Description	Input/Output
pszSensorName	Sensor name	Input

Return

The API has the following return values.

Table 2-3 STFLIB_ISP_SENSOR_GetSensorId Return Description

Return	Description
STF_S8	Return positive identifier or negative number if sensor cannot be found

Requirement

The files `libSensorapi.a` and `libISPC.a` are required for using this API.

2.2.2. STFLIB_ISP_SENSOR_SturctInitialize

The API is described in the following sections.

Description

Initialize a ST_SENSOR structure.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_SENSOR_SturctInitialize(
    ST_SENSOR *pstSensor,
    STF_U8 u8SensorId,
    STF_U8 u8Index
    );
```

Parameter

The API has the following parameters.

Table 2-4 STFLIB_ISP_SENSOR_SturctInitialize Parameter Description

Parameter	Description	Input/Output
pstSensor	Sensor handle	Input/Output
u8SensorId	Sensor ID	Input
u8Index	Sensor setting index. No longer in use	Input

Return

The API has the following return values.

Table 2-5 STFLIB_ISP_SENSOR_SturctInitialize Return Description

Return	Description
0	Success

Table 2-5 STFLIB_ISP_SENSOR_SturctInitialize Return Description (continued)

Return	Description
Other values	Failure

Requirement

The files `libSensorapi.a` and `libISPC.a` are required for using this API.

2.2.3. STFLIB_ISP_SENSOR_SturctUninitialize

The API is described in the following sections.

Description

De-initialize a ST_SENSOR structure.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_SENSOR_SturctUninitialize(
    ST_SENSOR *pstSensor,
    );
```

Parameter

The API has the following parameter.

Table 2-6 STFLIB_ISP_SENSOR_SturctUninitialize Parameter Description

Parameter	Description	Input/Output
pstSensor	Sensor handle	Input/Output

Return

The API has the following return values.

Table 2-7 STFLIB_ISP_SENSOR_SturctUninitialize Return Description

Return	Description
0	Success
Other values	Failure

Requirement

The files `libSensorapi.a` and `libISPC.a` are required for using this API.

3. Sensor Library - Adding a New Sensor

Currently the ISP SDK supports two sensors. One is the IMX219 MIPI sensor, and the other is the SC2235 DVP sensor. Both of them are using a “SENSOR_FUNCS” structure to implement the sensor feature and providing this handle for adaptive control. This chapter will discuss how to add a new sensor support to sensor library. Customer can follow this chapter to add a new sensor support if needed.

3.1. Procedure for Adding a New Sensor

Follow the procedure below to add a new sensor.

1. Declare a private sensor structure that packed the “**SENSOR_FUNCTS**” structure and necessary variables.

   For the MIPI sensor, you can refer to the files of `imx219mipi.c` and `imx219mipi.h`. Related to the DVP sensor, you can refer to the files of `sc2235dvp.c` and `sc2235dvp.h`.

2. Implement member functions based on the structure of “**SENSOR_FUNCS**”.

   The following member functions are required.

   - GetModeIdx
   - GetMode
   - GetState
   - GetInterfaceInfo
   - Enable
   - Disable
   - Destroy
   - GetInfo
   - GetRegister
   - SetRegister
   - GetGainRange
   - GetCurrentGain
   - SetGain
   - GetExposureRange
   - GetExposure
   - SetExposure
   - SetExposureAndGain

3. Implement a creation function for the private sensor structure initialization.
4. Add the sensor creation function into the “**InitialiseSensors[]**” structure.
5. Add the sensor name into the “**Sensors[]**” structure.
6. Add the file names with ".c" and ".h" extensions into the file `CMakeLists.txt`.
7. Re-build the ISP SDK to test the sensor functions.

3.2. Data Structure

The following topics show the data structure of Sensor Library - Adding a New Sensor.

3.2.1. Sensor Status

The following code block shows the data structure for the sensor status structure.

```
typedef struct _SENSOR_STATUS {
    STF_U8 u8CurrentMode;
    STF_U8 u8Flipping;
    STF_DOUBLE dCurrentFps;
    SENSOR_STATE enState;
} SENSOR_STATUS, *PSENSOR_STATUS;
```

The following table describes the fields in the above code block.

Table 3-1 Sensor Status Field Description

Field	Description
u8CurrentMode	Mode ID that the sensor is currently running in, this is an index to the list of modes returned by calling the **GetSensorMode** function
u8Flipping	Select flip and mirror information
dCurrentFps	Currently sensor output frame rate
enState	Sensor running state

3.2.2. Sensor Mode

The following code block shows the data structure for the sensor mode structure.

```
typedef struct _SENSOR_MODE {
    STF_U8 u8BitDepth;
    STF_U16 u16Width;
    STF_U16 u16Height;
    STF_DOUBLE dFrameRate;
    STF_DOUBLE dPixelRate;
    STF_U16 u16HorizontalTotal;
    STF_U16 u16VerticalTotal;
    STF_U8 u8SupportFlipping;
    STF_U32 u32ExposureMin;
    STF_U32 u32ExposureMax;
    STF_DOUBLE dExposureMin;
    STF_U8 u8MipiLanes;
} SENSOR_MODE, *PSENSOR_MODE;
```

The following table describes the fields in the above code block.

Table 3-2 DRM Buffer Field Description

Field	Description
u8BitDepth	The bit depth in the sensor mode
u16Width	Width of output in pixels
u16Height	Height of output in pixels
dFrameRate	Frame rate in Hz
dPixelRate	The pixel rate in the sensor mode
u16HorizontalTotal	Horizontal total size (including blanking) in pixels
u16VerticalTotal	Vertical total size (including blanking) in pixels
u8SupportFlipping	Support flipping when enabling

Table 3-2 DRM Buffer Field Description (continued)

Field	Description
u32ExposureMin	Minimum exposure time (integer type)
u32ExposureMax	Maximum exposure time
dExposureMin	Minimum exposure time (double type)
u8MipiLanes	MIPI lanes for MIPI interface sensor

3.2.3. Sensor Interface

The following code block shows the data structure for the sensor interface structure.

```
typedef struct _SENSOR_INTFC {
    STF_U8 u8DataInterface;
    STF_U8 u8I2cChannel;
    STF_U8 u8I2cSlaveAddrbit;
    STF_U8 u8I2cRegAddrBit;
    STF_U8 u8I2cRegDataBit;
    STF_U16 u16I2cSlaveAddr;
    STF_S16 s16PowerGpioId;
    STF_U16 u16PowerGpioInitLevel;
    STF_S16 s16ResetGpioId;
    STF_U16 u16ResetGpioInitLevel;
    STF_U16 u16ResetIntervalTime;
    STF_U8 u8SensorPolarity;
} SENSOR_INTFC, *PSENSOR_INTFC;
```

The following table describes the fields in the above code block.

Table 3-3 DRM Buffer Field Description

Field	Description
u8DataInterface	Indicate which data interface will be used. 0: DVP, 1: MIPI
u8I2cChannel	Indicate which I2C channel will be used. Range: 0 - 3
u8I2cSlaveAddrbit[1]	Indicate the slave address bit. 0: 7 bits, 1: 10 bits. Not used in the V4L2 driver
u8I2cRegAddrBit[1]	Indicate the sensor register address bit. 0: 8 bits, 1: 16 bits. Not used in the V4L2 driver
u8I2cRegDataBit[1]	Indicate the sensor register data bit. 0: 8 bits, 1: 16 bits. Not used in the V4L2 driver
u16I2cSlaveAddr	Indicate the slave address of sensor.
s16PowerGpioId[1]	Power control pin. Not used in the V4L2 driver
u16PowerGpioInitLevel[1]	Power control pin initialize value. Not used in the V4L2 driver
s16ResetGpioId[1]	Reset pin. Not used in the V4L2 driver
u16ResetGpioInitLevel[1]	Reset pin initialize value. Not used in the V4L2 driver
u16ResetIntervalTime[1]	Reset pin hold timing. Not used in the V4L2 driver
u8SensorPolarity	Sensor polarity status

Note:

1. The field items are for proprietary driver only.

3.2.4. Sensor Information

The following code block shows the data structure for the sensor information structure.

```
typedef struct _SENSOR_INFO {
    enum MOSAICType enBayerOriginal;
    enum MOSAICType enBayerEnabled;
    char pszSensorName[SENSOR_INFO_NAME_MAX];
    char pszSensorVersion[SENSOR_INFO_VERSION_MAX];
    STF_DOUBLE dNumber;
    STF_U16 u16FocalLength;
    STF_U32 u32WellDepth;
    STF_DOUBLE dReadNoise;
    STF_U8 u8Imager;
    STF_BOOL8 bBackFacing;
    SENSOR_STATUS stStatus;
    SENSOR_MODE stMode;
    STF_U32 u32ModeCount;
    EN_EXPO_GAIN_METHOD enExposureGainMethod;
} SENSOR_INFO, *PSENSOR_INFO;
```

The following table describes the fields in the above code block.

Table 3-4 Sensor Information Field Description

Field	Description
enBayerOriginal	Original CFA filter layout (before flipping)
enBayerEnabled	Enabled CFA filter layout (including flipping)
pszSensorName	Text name for the sensor for easy identification
pszSensorVersion	To be used where multiple versions of the sensor exists
dNumber[1]	Aperture of the attached lens. No longer in use
u16FocalLength[1]	Focal length of the lens. No longer in use
u32WellDepth[1]	Number of electrons can be held by the sensor wells before clipping. No longer in use
dReadNoise[1]	Standard deviation of noise when reading pixel value off a sensor in electrons. No longer in use
u8Imager[1]	Imager ID. No longer in use
bBackFacing[1]	Is this a back or front of facing camera? No longer in use
stStatus	Sensor status
stMode	Sensor mode information
u32ModeCount	Indicate how many modes are supported in this sensor
enExposureGainMethod	Indicate how to program the exposure time and gain

Note:

1. The field items are no longer in use.

3.2.5. Sensor Function

The following code block shows the data structure for the sensor function structure.

```
typedef struct _SENSOR_FUNCS {
    STF_S8 (*GetModeIdx)(SENSOR_HANDLE *pstHandle, STF_U16 u16Width, STF_U16 u16Height);
```

```
    STF_RESULT (*GetMode)(SENSOR_HANDLE *pstHandle, STF_U8 u8ModeIdx, SENSOR_MODE *pstModes);
    STF_RESULT (*GetState)(SENSOR_HANDLE *pstHandle, SENSOR_STATUS *pstStatus);
    STF_RESULT (*GetInterfaceInfo)(SENSOR_HANDLE *pstHandle, SENSOR_INTFC *pstInterface);
    STF_RESULT (*SetMode)(SENSOR_HANDLE *pstHandle, STF_U8 u8ModeIdx, STF_U8 u8Flipping);
    STF_RESULT (*Enable)(SENSOR_HANDLE *pstHandle);
    STF_RESULT (*Disable)(SENSOR_HANDLE *pstHandle);
    STF_RESULT (*Destroy)(SENSOR_HANDLE *pstHandle);
    STF_RESULT (*GetInfo)(SENSOR_HANDLE *pstHandle, SENSOR_INFO *pstInfo);
    STF_RESULT (*GetRegister)(SENSOR_HANDLE *pstHandle, STF_U16 u16RegAddr, STF_U16 *pu16RegVal, STF_U8
 u8Context);
    STF_RESULT (*SetRegister)(SENSOR_HANDLE *pstHandle, STF_U16 u16RegAddr, STF_U16 u16RegVal, STF_U8
 u8Context);
    STF_RESULT (*GetGainRange)(SENSOR_HANDLE *pstHandle, STF_DOUBLE *pdMin, STF_DOUBLE *pdMax, STF_U8
 *pu8Contexts );
    STF_RESULT (*GetCurrentGain)(SENSOR_HANDLE *pstHandle, STF_DOUBLE *pdCurrentGain, STF_U8 u8Context);
    STF_RESULT (*SetGain)(SENSOR_HANDLE *pstHandle, STF_DOUBLE dGain, STF_U8 u8Context);
    STF_RESULT (*GetExposureRange)(SENSOR_HANDLE *pstHandle, STF_U32 *pu32Min, STF_U32 *pu32Max, STF_U8
 *pu8Contexts);
    STF_RESULT (*GetExposure)(SENSOR_HANDLE *pstHandle, STF_U32 *pu32Exposure, STF_U8 u8Context);
    STF_RESULT (*SetExposure)(SENSOR_HANDLE *pstHandle, STF_U32 u32Exposure, STF_U8 u8Context);
    STF_RESULT (*GetFocusRange)(SENSOR_HANDLE *pstHandle, STF_U16 *pu16Min, STF_U16 *pu16Max);
    STF_RESULT (*GetCurrentFocus)( SENSOR_HANDLE *pstHandle, STF_U16 *pu16CurrentFocus);
    STF_RESULT (*SetFocus)(SENSOR_HANDLE *pstHandle, STF_U16 u16Focus);
    STF_RESULT (*ConfigureFlash)(SENSOR_HANDLE *pstHandle, STF_BOOL8 bAlwaysOn, STF_S16 s16FrameDelay, STF_S16
 s16Frames, STF_U16 u16FlashPulseWidth);
    STF_RESULT (*Insert)(SENSOR_HANDLE *pstHandle);
    STF_RESULT (*WaitProcessed)(SENSOR_HANDLE *pstHandle);
    STF_RESULT (*SetFlipMirror)(SENSOR_HANDLE *pstHandle, STF_U8 u8Flag);
    STF_RESULT (*GetFixedFPS)(SENSOR_HANDLE *pstHandle, STF_U16 *py16Fixed);
    STF_RESULT (*SetFPS)(SENSOR_HANDLE *pstHandle, STF_DOUBLE dFps);
    STF_RESULT (*SetExposureAndGain)(SENSOR_HANDLE *pstHandle, STF_U32 u32Exposure, STF_DOUBLE dGain, STF_U8
 u8Context);
    STF_RESULT (*SetResolution)(SENSOR_HANDLE *pstHandle,  STF_U16 u16ImgW, STF_U16 U16ImgH);
    STF_RESULT (*GetSnapRes)(SENSOR_HANDLE *pstHandle, ST_RES_LIST *pstReslist);
    STF_RESULT (*Reset)(SENSOR_HANDLE *pstHandle);
} SENSOR_FUNCS, *PSENSOR_FUNCS;
```

The following table describes the fields in the above code block.

Table 3-5 Sensor Function Field Description

Field	Description
***GetModeIdx**	Get the sensor mode index
***GetMode**	Fill in a structure detailing the modes supported by the sensor
***GetState**	Get the current state of the sensor
***GetInterfaceInfo**	Get the sensor's connection data bus and I2C hardware information
***SetMode**	Set the sensor mode
***Enable**	Enable the sensor
***Disable**	Disable the sensor
***Destroy**	Release and de-initialize the sensor
***GetInfo**	Get information about the sensor
***GetRegister**	Read the sensor register value
***SetRegister**	Set the sensor register value
***GetGainRange**	Get the range of gain settings that the sensor can support
***GetCurrentGain**	Get the current gain setting from sensor
***SetGain**	Set the current gain setting to the sensor

Table 3-5 Sensor Function Field Description (continued)

Field	Description
***GetExposureRange**	Get the range of exposures the sensor can use
***GetExposure**	Query the sensor for the current exposure period
***SetExposure**	Set the sensor to the requested exposure period
***GetFocusRange**[1]	Get the range of allowed focus positions. No longer in use
***GetCurrentFocus**[1]	Query the current focus position. No longer in use
***SetFocus**[1]	Set the sensor to focus at the selected position. No longer in use
***ConfigureFlash**[1]	Set the sensor to the requested exposure period. No longer in use
***Insert**[1]	Used to control insertion point of the sensor. No longer in use
***WaitProcessed**[1]	Used to wait for an inserted frame to have been fully processed and sent to the ISP. No longer in use
***SetFlipMirror**	Set flip and mirror value in the sensor
***GetFixedFPS**	Get the fixed FPS value in the sensor
***SetFPS**	Set the FPS value in the sensor
***SetExposureAndGain**	Program the exposure time and gain in the sensor
***SetResolution**[1]	Set the resolution value to the sensor. No longer in use
***GetSnapRes**[1]	Get supported resolution list from the sensor library
***Reset**	Reset the sensor.

Note:

1. The field items are no longer in use.

3.3. Member Functions - To be Implemented

The following topics show the member functions of Sensor Library - Adding a New Sensor.

3.3.1. GetModeIdx

The function is described in the following sections.

Description

Get the sensor mode index.

Syntax

The following code block shows the syntax of the function.

```
GetModeIdx(
    SENSOR_HANDLE *pstHandle,
    STF_U16 u16Width,
    STF_U16 u16Height
    );
```

Parameter

The function has the following parameters.

Table 3-6 GetModeIdx Parameter Description

Parameter	Description	Input/Output
pstHandle	Sensor handle	Input
u16Width	Image width	Input
u16Height	Image height	Input

Return

The function has the following return values.

Table 3-7 GetModeIdx Return Description

Return	Description
STF_S8	Sensor mode index

Requirement

None

3.3.2. GetMode

The function is described in the following sections.

Description

Fill in a structure detailing the modes supported by the sensor.

Syntax

The following code block shows the syntax of the function.

```
GetMode(
    SENSOR_HANDLE *pstHandle,
    STF_U8 u8ModeIdx,
    SENSOR_MODE *pstModes
    );
```

Parameter

The function has the following parameters.

Table 3-8 GetMode Parameter Description

Parameter	Description	Input/Output
pstHandle	Sensor handle	Input
u8ModeIdx	Sensor mode index	Input
pstModes	Sensor mode information	Output

Return

The function has the following return values.

Table 3-9 GetMode Return Description

Return	Description
0	Success
Other values	Failure

Requirement

None

3.3.3. GetState

The function is described in the following sections.

Description

Get the current state of the sensor.

Syntax

The following code block shows the syntax of the function.

```
GetState(
    SENSOR_HANDLE *pstHandle,
    SENSOR_STATUS *pstStatus
    );
```

Parameter

The function has the following parameters.

Table 3-10 GetState Parameter Description

Parameter	Description	Input/Output
pstHandle	Sensor handle	Input
pstStatus	Sensor state	Output

Return

The function has the following return values.

Table 3-11 GetState Return Description

Return	Description
0	Success
Other values	Failure

Requirement

None

3.3.4. GetInterfaceInfo

The function is described in the following sections.

Description

Get the sensor's connection data bus and I2C hardware information.

Syntax

The following code block shows the syntax of the function.

```
GetInterfaceInfo(
    SENSOR_HANDLE *pstHandle,
    SENSOR_INTFC *pstInterface
    );
```

Parameter

The function has the following parameters.

Table 3-12 GetInterfaceInfo Parameter Description

Parameter	Description	Input/Output
pstHandle	Sensor handle	Input
pstInterface	Sensor interface handle	Output

Return

The function has the following return values.

Table 3-13 GetInterfaceInfo Return Description

Return	Description
0	Success
Other values	Failure

Requirement

None

3.3.5. SetMode

The function is described in the following sections.

Description

Set the sensor mode.

Syntax

The following code block shows the syntax of the function.

```
SetMode(
    SENSOR_HANDLE *pstHandle,
    STF_U8 u8ModeIdx,
    STF_U8 u8Flipping
    );
```

Parameter

The function has the following parameters.

Table 3-14 SetMode Parameter Description

Parameter	Description	Input/Output
pstHandle	Sensor handle	Input
u8ModeIdx	Sensor mode index	Input
u8Flipping	Flip and mirror information	Input

Return

The function has the following return values.

Table 3-15 SetMode Return Description

Return	Description
0	Success
Other values	Failure

Requirement

None

3.3.6. Enable

The function is described in the following sections.

Description

Enable the sensor.

Syntax

The following code block shows the syntax of the function.

```
Enable(
    SENSOR_HANDLE *pstHandle
    );
```

Parameter

The function has the following parameters.

Table 3-16 Enable Parameter Description

Parameter	Description	Input/Output
pstHandle	Sensor handle	Input

Return

The function has the following return values.

Table 3-17 Enable Return Description

Return	Description
0	Success
Other values	Failure

Requirement

None

3.3.7. Disable

The function is described in the following sections.

Description

Disable the sensor.

Syntax

The following code block shows the syntax of the function.

```
Disable(
    SENSOR_HANDLE *pstHandle
    );
```

Parameter

The function has the following parameter.

Table 3-18 Disable Parameter Description

Parameter	Description	Input/Output
pstHandle	Sensor handle	Input

Return

The function has the following return values.

Table 3-19 Disable Return Description

Return	Description
0	Success
Other values	Failure

Requirement

None

3.3.8. Destroy

The function is described in the following sections.

Description

Release and de-initialize the sensor.

Syntax

The following code block shows the syntax of the function.

```
Destroy(
    SENSOR_HANDLE *pstHandle
    );
```

Parameter

The function has the following parameters.

Table 3-20 Destroy Parameter Description

Parameter	Description	Input/Output
pstHandle	Sensor handle	Input

Return

The function has the following return values.

Table 3-21 Destroy Return Description

Return	Description
0	Success
Other values	Failure

Requirement

None

3.3.9. GetInfo

The function is described in the following sections.

Description

Release and de-initialize the sensor.

Syntax

The following code block shows the syntax of the function.

```
GetInfo(
    SENSOR_HANDLE *pstHandle,
    SENSOR_INFO *pstInfo
    );
```

Parameter

The function has the following parameters.

Table 3-22 GetInfo Parameter Description

Parameter	Description	Input/Output
pstHandle	Sensor handle	Input
pstInfo	Sensor information handle	Output

Return

The function has the following return values.

Table 3-23 GetInfo Return Description

Return	Description
0	Success

Table 3-23 GetInfo Return Description (continued)

Return	Description
Other values	Failure

Requirement

None

3.3.10. GetRegister

The function is described in the following sections.

Description

Read the sensor register value.

Syntax

The following code block shows the syntax of the function.

```
GetRegister(
    SENSOR_HANDLE *pstHandle,
    STF_U16 u16RegAddr,
    STF_U16 *pu16RegVal,
    STF_U8 u8Context
    );
```

Parameter

The function has the following parameters.

Table 3-24 GetRegister Parameter Description

Parameter	Description	Input/Output
pstHandle	Sensor handle	Input
u16RegAddr	Register address	Input
pu16RegVal	Read back value from the sensor	Output
u8Context	Sensor contexts. No longer in use	Input

Return

The function has the following return values.

Table 3-25 GetRegister Return Description

Return	Description
0	Success
Other values	Failure

Requirement

None

3.3.11. SetRegister

The function is described in the following sections.

Description

Set the sensor register value.

Syntax

The following code block shows the syntax of the function.

```
SetRegister(
    SENSOR_HANDLE *pstHandle,
    STF_U16 u16RegAddr,
    STF_U16 u16RegVal,
    STF_U8 u8Context
    );
```

Parameter

The function has the following parameters.

Table 3-26 SetRegister Parameter Description

Parameter	Description	Input/Output
pstHandle	Sensor handle	Input
u16RegAddr	Register address	Input
u16RegVal	Register value	Input
u8Context	Sensor contexts. No longer in use	Input

Return

The function has the following return values.

Table 3-27 SetRegister Return Description

Return	Description
0	Success
Other values	Failure

Requirement

None

3.3.12. GetGainRange

The function is described in the following sections.

Description

Fill in a structure detailing the modes supported by the sensor.

Syntax

The following code block shows the syntax of the function.

```
GetMode(
    SENSOR_HANDLE *pstHandle,
    STF_U8 u8ModeIdx,
    SENSOR_MODE *pstModes
```

```
    );
```

Parameter

The function has the following parameters.

Table 3-28 GetMode Parameter Description

Parameter	Description	Input/Output
pstHandle	Sensor handle	Input
u8ModeIdx	Sensor mode index	Input
pstModes	Sensor mode information	Output

Return

The function has the following return values.

Table 3-29 GetMode Return Description

Return	Description
0	Success
Other values	Failure

Requirement

None

3.3.13. GetCurrentGain

The function is described in the following sections.

Description

Read the sensor register value.

Syntax

The following code block shows the syntax of the function.

```
GetCurrentGain(
    SENSOR_HANDLE *pstHandle,
    STF_DOUBLE *pdCurrentGain,
    STF_U8 u8Context
    );
```

Parameter

The function has the following parameters.

Table 3-30 GetCurrentGain Parameter Description

Parameter	Description	Input/Output
pstHandle	Sensor handle	Input
pdCurrentGain	Current gain value	Output
u8Context	Sensor contexts. No longer in use	Input

Return

The function has the following return values.

Table 3-31 GetCurrentGain Return Description

Return	Description
0	Success
Other values	Failure

Requirement

None

3.3.14. SetGain

The function is described in the following sections.

Description

Set the current gain to the sensor.

Syntax

The following code block shows the syntax of the function.

```
SetGain(
    SENSOR_HANDLE *pstHandle,
    STF_DOUBLE dGain,
    STF_U8 u8Context
    );
```

Parameter

The function has the following parameters.

Table 3-32 SetGain Parameter Description

Parameter	Description	Input/Output
pstHandle	Sensor handle	Input
dGain	Gain value	Input
u8Context	Sensor context. No longer in use	Input

Return

The function has the following return values.

Table 3-33 SetGain Return Description

Return	Description
0	Success
Other values	Failure

Requirement

None

3.3.15. GetExposureRange

The function is described in the following sections.

Description

Get the range of exposures the sensor can use.

Syntax

The following code block shows the syntax of the function.

```
GetExposureRange(
    SENSOR_HANDLE *pstHandle,
    STF_U32 *pu32Min,
    STF_U32 *pu32Max,
    STF_U8 *pu8Contexts
    );
```

Parameter

The function has the following parameters.

Table 3-34 GetExposureRange Parameter Description

Parameter	Description	Input/Output
pstHandle	Sensor handle	Input
pu32Min	Minimum exposure time value	Output
pu32Max	Maximum exposure time	Output
u8Context	Sensor context. No longer in use	Input

Return

The function has the following return values.

Table 3-35 GetExposureRange Return Description

Return	Description
0	Success
Other values	Failure

Requirement

None

3.3.16. GetExposure

The function is described in the following sections.

Description

Query the sensor for the current exposure period.

Syntax

The following code block shows the syntax of the function.

```
GetExposure(
    SENSOR_HANDLE *pstHandle,
    STF_U32 *pu32Exposure,
    STF_U8 u8Context
    );
```

Parameter

The function has the following parameters.

Table 3-36 GetExposure Parameter Description

Parameter	Description	Input/Output
pstHandle	Sensor handle	Input
pu32Exposure	Exposure time value	Output
u8Context	Sensor context. No longer in use	Input

Return

The function has the following return values.

Table 3-37 GetExposure Return Description

Return	Description
0	Success
Other values	Failure

Requirement

None

3.3.17. SetExposure

The function is described in the following sections.

Description

Set the sensor to the requested exposure period.

Syntax

The following code block shows the syntax of the function.

```
SetExposure(
    SENSOR_HANDLE *pstHandle,
    STF_U32 u32Exposure,
    STF_U8 u8Context
    );
```

Parameter

The function has the following parameters.

Table 3-38 SetExposure Parameter Description

Parameter	Description	Input/Output
pstHandle	Sensor handle	Input
u32Exposure	Exposure time value	Output

Table 3-38 SetExposure Parameter Description (continued)

Parameter	Description	Input/Output
u8Context	Sensor context. No longer in use	Input

Return

The function has the following return values.

Table 3-39 SetExposure Return Description

Return	Description
0	Success
Other values	Failure

Requirement

None

3.3.18. SetFlipMirror

The function is described in the following sections.

Description

Set flip and mirror value in the sensor.

Syntax

The following code block shows the syntax of the function.

```
SetFlipMirror(
    SENSOR_HANDLE *pstHandle,
    STF_U8 u8Flag
    );
```

Parameter

The function has the following parameters.

Table 3-40 SetFlipMirror Parameter Description

Parameter	Description	Input/Output
pstHandle	Sensor handle	Input
u8Flag	Flip and mirror status	Input

Return

The function has the following return values.

Table 3-41 SetFlipMirror Return Description

Return	Description
0	Success
Other values	Failure

Requirement

None

3.3.19. GetFixedFPS

The function is described in the following sections.

Description

Get the fixed FPS value in the sensor.

Syntax

The following code block shows the syntax of the function.

```
GetFixedFPS(
    SENSOR_HANDLE *pstHandle,
    STF_U16 *pu16FixedFps
    );
```

Parameter

The function has the following parameters.

Table 3-42 GetFixedFPS Parameter Description

Parameter	Description	Input/Output
pstHandle	Sensor handle	Input
pu16FixedFps	Fixed FPS value	Output

Return

The function has the following return values.

Table 3-43 GetFixedFPS Return Description

Return	Description
0	Success
Other values	Failure

Requirement

None

3.3.20. SetFPS

The function is described in the following sections.

Description

Set the FPS value in the sensor.

Syntax

The following code block shows the syntax of the function.

```
SetFPS(
    SENSOR_HANDLE *pstHandle,
    STF_DOUBLE dFps
```

```
    );
```

Parameter

The function has the following parameters.

Table 3-44 SetFPS Parameter Description

Parameter	Description	Input/Output
pstHandle	Sensor handle	Input
dFps	FPS value	Input

Return

The function has the following return values.

Table 3-45 SetFPS Return Description

Return	Description
0	Success
Other values	Failure

Requirement

None

3.3.21. SetExposureAndGain

The function is described in the following sections.

Description

Program the exposure time and gain in the sensor.

Syntax

The following code block shows the syntax of the function.

```
SetExposureAndGain(
    SENSOR_HANDLE *pstHandle,
    STF_U32 u32Exposure,
    STF_DOUBLE dGain,
    STF_U8 u8Context
    );
```

Parameter

The function has the following parameters.

Table 3-46 SetExposureAndGain Parameter Description

Parameter	Description	Input/Output
pstHandle	Sensor handle	Input
u32Exposure	Exposure time value	Input
dGain	Gain value	Input
u8Context	Sensor context. No longer in use	Input

Return

The function has the following return values.

Table 3-47 SetExposureAndGain Return Description

Return	Description
0	Success
Other values	Failure

Requirement

None

3.3.22. Reset

The function is described in the following sections.

Description

Reset the sensor.

Syntax

The following code block shows the syntax of the function.

```
Reset(
    SENSOR_HANDLE *pstHandle
    );
```

Parameter

The function has the following parameter.

Table 3-48 Reset Parameter Description

Parameter	Description	Input/Output
pstHandle	Sensor handle	Input

Return

The function has the following return values.

Table 3-49 Reset Return Description

Return	Description
0	Success
Other values	Failure

Requirement

None

3.4. API - To be Implemented

The following topics show the application interfaces (API) of Sensor Library - Porting a New Sensor.

3.4.1. Sensor_Create

The API is described in the following sections.

Description

Sensor structure create and initialization.

Syntax

The following code block shows the syntax of the API.

```
Sensor_Create(
    SENSOR_HANDLE **ppstHandle,
    STF_U8 u8Index
    );
```

Parameter

The API has the following parameters.

Table 3-50 Sensor_Create Parameter Description

Parameter	Description	Input/Output
ppstHandle	Sensor handle	Input/Output
u8Index	Sensor setting index. No longer in use	Input

Return

The API has the following return values.

Table 3-51 Sensor_Create Return Description

Return	Description
0	Success
Other values	Failure

Requirement

None

4. ISPC Library - Device

This section explains the device library API and data structures. The device library integrates necessary function in device structure and provides the user with a simple method to operate the media sub-device, video device, FB device, PP device, and DRM device. Customers can either use this library directly or through the pipeline to operate the device.

4.1. Data Structure

The following topics show the data structure of ISPC Library - Device.

4.1.1. Pre-process Video Device Mode

The following code block shows the data structure for the pre-process video device mode structure.

```
typedef struct _ST_PP_VDO_MODE {
    EN_PXL_FMT enPxlFmt;
    STF_UINT uiHeight;
    STF_UINT uiWidth;
    STF_UINT uiAddr;
} ST_PP_VDO_MODE, *PST_PP_VDO_MODE;
```

The following table describes the fields in the above code block.

Table 4-1 Pre-process Video Device Mode Field Description

Field	Description
enPxlFmt	Pixel format
uiHeight	Image Height
uiWidth	Image width
uiAddr	Buffer address

4.1.2. Pre-process Mode

The following code block shows the data structure for the pre-process mode structure.

```
typedef struct _ST_PP_MODE {
    STF_U8 u8Id;
    bool bBusOut;
    bool bFifoOut;
    bool bInitialized;
    ST_PP_VDO_MODE stPpVdoModeSrc;
    ST_PP_VDO_MODE stPpVdoModeDst;
} ST_PP_MODE, *PST_PP_MODE;
```

The following table describes the fields in the above code block.

Table 4-2 Pre-process Mode Field Description

Field	Description
u8Id	Pre-process ID
bBusOut	Indicate the pre-process output image data to DDR
bFifoOut	Indicate the pre-process output image data to LCDC
bInitialized	Indicate the pre-process is initialized

Table 4-2 Pre-process Mode Field Description (continued)

Field	Description
stPpVdoModeSrc	Store the video device mode information of the source of the pre-process
stPpVdoModeDst	Store the video device mode information of the destination of the pre-process

4.1.3. DRM Buffer

The following code block shows the data structure for the DRM buffer structure.

```
typedef struct _ST_DRM_BUF {
    STF_U32 u32Pitch;
    STF_U32 u32Size;
    STF_U32 u32FbId;
    STF_INT nDmaBufFd;  // Used for DMA_BUF
    STF_INT nBufObjHandle;
    STF_U8 *pu8Buf;
} ST_DRM_BUF, *PST_DRM_BUF;
```

The following table describes the fields in the above code block.

Table 4-3 DRM Buffer Field Description

Field	Description
u32Pitch	The pitch of dumb buffer
u32Size	Dumb buffer size
u32FbId	The index of frame buffer
nDmaBufFd	The file descriptor for the DMA buffer
nBufObjHandle	The object handle of the dumb buffer
pu8Buf	The dumb buffer pointer

4.1.4. DRM Device

The following code block shows the data structure for the DRM device structure.

```
typedef struct _ST_DRM_DEV {
    STF_U32 u32ConnId;
    STF_U32 u32EncId;
    STF_U32 u32CrtId;
    STF_U32 u32Width;
    STF_U32 u32Height;
    STF_U32 u32Pitch;
    drmModeModeInfo stDrmModeInfo;
    drmModeCrtc *pstDrmCrtc;
    STF_U32 u32DrmFormat;
    ST_DRM_BUF stDrmBuf[PIPELINE_IMG_BUF_MAX];
    ST_DRM_DEV *pstDrmDevNext;
} ST_DRM_DEV, *PST_DRM_DEV;
```

The following table describes the fields in the above code block.

Table 4-4 DRM Device Field Description

Field	Description
u32ConnId	The connector id of DRM device
u32EncId	Store encoder id

Table 4-4 DRM Device Field Description (continued)

Field	Description
u32CrtId	Store selected CRT id
u32Width	Store the width of DRM device
u32Height	Store the height of DRM device
u32Pitch	Store the pitch information of buffer
stDrmModeInfo	Store the mode information of DRM device
pstDrmCrtc	Store the context information of CRT device
u32DrmFormat	Store the format information of DRM device
stDrmBuf	Store the buffer information of DRM device
pstDrmDevNext	Point to next DRM device

4.1.5. DMA Buffer Information

The following code block shows the data structure for the DMA buffer information structure.

```
typedef struct _ST_DMA_BUF_INFO {
    STF_INT nDmaBufFd;
    STF_VOID *pvBuffer;
} ST_DMA_BUF_INFO, *PST_DMA_BUF_INFO;
```

The following table describes the fields in the above code block.

Table 4-5 DMA Buffer Information Field Description

Field	Description
nDmaBufFd	DMA buffer file descriptor
pvBuffer	Point to user space buffer

4.1.6. Video Device Buffer

The following code block shows the data structure for the video device buffer structure.

```
typedef struct _ST_VDO_BUF {
    STF_U32 u32Index;
    STF_INT nDmaBufFd;
    STF_U32 u32Length;
    STF_VOID *pvBuffer;
} ST_VDO_BUF, *PST_VDO_BUF;
```

The following table describes the fields in the above code block.

Table 4-6 Video Device Buffer Field Description

Field	Description
u32Index	Buffer index
nDmaBufFd	DMA buffer file descriptor
u32Length	DMA buffer size
pvBuffer	Point to user space buffer

4.1.7. Video Memory

The following code block shows the data structure for the video memory structure.

```
typedef union _ST_VDO_MEM {
    STF_U8 u8BufIdx;
    STF_INT nDmaBufFd;
    STF_U32 u32Length;
    STF_VOID *pvBuffer;
    STF_U32 u32BytesUsed;
    struct timeval stTimeStamp;
    STF_BOOL8 bIsScDumpForAe;
    CI_MEM_PARAM *pstMem1;
    CI_MEM_PARAM *pstMem2;
    sCell_T stCell;
} ST_VDO_MEM, *PST_VDO_MEM;
```

The following table describes the fields in the above code block.

Table 4-7 Video Memory Field Description

Field	Description
u8BufIdx	Buffer index
nDmaBufFd	DMA buffer file descriptor
u32Length	DMA buffer size
pvBuffer	Point to user space buffer
u32BytesUsed	Indicate how many bytes be used to store image data
stTimeStamp	Store the timestamp information
bIsScDumpForAe	Indicate that this buffer is ready for AE or AWB
pstMem1	Point to user space buffer segment 1
pstMem2	Point to user space buffer segment 2
stCell	Link list data structure

4.1.8. Video Memory Table

The following code block shows the data structure for the video memory table structure.

```
typedef struct _ST_VDO_MEM_TBL {
    STF_U32 u32Count;
    ST_VDO_MEM *pstVdoMem;
} ST_VDO_MEM_TBL, *PST_VDO_MEM_TBL;
```

The following table describes the fields in the above code block.

Table 4-8 Video Memory Table Field Description

Field	Description
u32Count	Indicate the amount of video memory in this table
pstVdoMem	Video memory array

4.1.9. Item Information Table

The following code block shows the data structure for the item information table structure.

```
typedef struct _ST_ITEM_INFO {
    STF_U8 u8Cnt;
    STF_U8 u8Idx;
    STF_CHAR szInfo[ITEM_INFO_MAX][128];
} ST_ITEM_INFO , *PST_ITEM_INFO;
```

The following table describes the fields in the above code block.

Table 4-9 Item Information Table Field Description

Field	Description
u8Cnt	Indicate the amount of items in this table
u8Idx	Not used
szInfo	Storage array of char type string information

4.1.10. Video Buffer Information

The following code block shows the data structure for the video buffer information structure.

```
typedef struct _ST_VDO_BUF_INFO {
    STF_U8 u8BufIdx;
    STF_BOOL8 bIsScDumpForAe;
    STF_U32 u32BytesUsed;
    struct timeval stTimeStamp;
} ST_VDO_BUF_INFO, *PST_VDO_BUF_INFO;
```

The following table describes the fields in the above code block.

Table 4-10 Video Buffer Information Field Description

Field	Description
u8BufIdx	Buffer index
bIsScDumpForAe	Indicate that this buffer is ready for AE or AWB
u32BytesUsed	Indicate the number of bytes used to store image data
stTimeStamp	Store the timestamp information

4.1.11. Frame Buffer Device Parameter

The following code block shows the data structure for the frame buffer device parameter structure.

```
typedef struct _ST_FB_DEV_PARAM {
    STF_U16 u16Width;
    STF_U16 u16Height;
    STF_U32 u32Bpp;
    STF_U32 u32ScreenSize;
    STF_U32 u32PixelFormat;
    STF_VOID *pvBuffer;
    struct fb_var_screeninfo stFbVarScreenInfo;
    struct fb_fix_screeninfo stFbFixScreenInfo;
} ST_FB_DEV_PARAM, *PST_FB_DEV_PARAM;
```

The following table describes the fields in the above code block.

Table 4-11 Frame Buffer Device Parameter Field Description

Field	Description
u16Width	The width of frame buffer

Table 4-11 Frame Buffer Device Parameter Field Description (continued)

Field	Description
u16Height	The height of frame buffer
u32Bpp	The bits per pixel
u32ScreenSize	The screen buffer size
u32PixelFormat	The pixel format
pvBuffer	Point to user space buffer
stFbVarScreenInfo	Store variable screen information
stFbFixScreenInfo	Store fixed screen information

4.1.12. Pre-process Device Parameter

The following code block shows the data structure for the pre-process device parameter structure.

```
typedef struct _ST_PP_DEV_PARAM {
    STF_U32 u32PixelFormat;
} ST_PP_DEV_PARAM, *PST_PP_DEV_PARAM;
```

The following table describes the fields in the above code block.

Table 4-12 Pre-process Device Parameter Field Description

Field	Description
u32PixelFormat	The pixel format

4.1.13. DRM Device Parameter

The following code block shows the data structure for the DRM device parameter structure.

```
typedef struct _ST_DRM_DEV_PARAM {
    STF_U16 u16Width;
    STF_U16 u16Height;
    STF_U32 u32PixelFormat;
    STF_U32 u32BufCount;
    STF_INT nConnectorId;
    ST_DRM_DEV *pstDrmDevHead;
} ST_DRM_DEV_PARAM, *PST_DRM_DEV_PARAM;
```

The following table describes the fields in the above code block.

Table 4-13 DRM Device Parameter Field Description

Field	Description
u16Width	The width of DRM buffer
u16Height	The height of DRM buffer
u32PixelFormat	The pixel format
u32BufCount	Indicate the amount of DRM buffers
nConnectorId	The connector id of DRM device
pstDrmDevHead	Point to the head of the DRM device list

4.1.14. Video Device Parameter

The following code block shows the data structure for the video device parameter structure.

```
typedef struct _ST_VDEV_PARAM {
    STF_BOOL8 bForceGenDev;
    STF_BOOL8 bIsVideoOut;
    STF_BOOL8 bStreamOn;
    STF_BOOL8 bIsDmaBufAllocFromDrmDev;
    STF_U16 u16Width;
    STF_U16 u16Height;
    STF_U32 u32PixelFormat;
    STF_BOOL8 bCrop;
    ST_RECT_2 stCropRect;
    STF_U8 u8Fps;
    ST_VDO_MEM_TBL stVdoMemTbl;
    pthread_mutex_t stBufLock;
    pthread_mutex_t stListLock;
    sLinkedList_T stAvailableQueue;
    sLinkedList_T stCompletedQueue;
} ST_VDEV_PARAM, *PST_VDEV_PARAM;
```

The following table describes the fields in the above code block.

Table 4-14 Video Device Parameter Field Description

Field	Description
bForceGenDev	Indicate that this video device will initialize as generic device
bIsVideoOut	Indicate that this video device is an output or capture device
bStreamOn	Indicate that this video device is stream on or off
bIsDmaBufAllocFromDrmDev	Indicate that the video buffer is allocated by DRM or video device
u16Width	The width of video buffer
u16Height	The height of video buffer
u32PixelFormat	The pixel format
bCrop	Indicate whether the capture window is crop
stCropRect	Store the crop area information
u8Fps	Indicate frames per second
stVdoMemTbl	Video memory table
stBufLock	For the video buffer operation protection
stListLock	For the link list operation protection
stAvailableQueue	Queue of available video buffers for video device
stCompletedQueue	Queue of completed video buffers for post processing

4.1.15. Basic Device

The following code block shows the data structure for the basic device structure.

```
typedef struct _ST_CI_DEVICE {
    CI_CONNECTION *pstCiConnection;
    STF_U8 u8DeviceId;
    STF_U8 u8DeviceType;
    STF_CHAR szDeviceName[DEV_NAME_LEN_MAX];
    STF_S32 s32CiConnections;
```

```
    STF_BOOL8 bConnected;
    EN_MEM_TYPE enMemType;
    STF_VOID *pvDevParam;

    STF_RESULT (*Destroy)(ST_CI_DEVICE *pstDevice);
    STF_RESULT (*Connection)(ST_CI_DEVICE *pstDevice);
    STF_RESULT (*Disconnect)(ST_CI_DEVICE *pstDevice);
    STF_BOOL8 (*IsConnected)(ST_CI_DEVICE *pstDevice);
    STF_S32 (*GetConnections)(ST_CI_DEVICE *pstDevice);
    CI_CONNECTION *(*GetConnection)(ST_CI_DEVICE *pstDevice);
    int (*GetFileHandle)(ST_CI_DEVICE *pstDevice);
    STF_RESULT (*SubscribeEvent)(ST_CI_DEVICE *pstDevice, STF_S32 s32EventType);
    STF_RESULT (*UnsubscribeEvent)(ST_CI_DEVICE *pstDevice, STF_S32 s32EventType);
    STF_RESULT (*DequeueEvent)(ST_CI_DEVICE *pstDevice, struct v4l2_event *pstEvent);
    STF_RESULT (*InitDevice)(ST_CI_DEVICE *pstDevice, STF_U32 u32PixelFormat);
    STF_U32 (*GetPixelFormat)(ST_CI_DEVICE *pstDevice);
    STF_RESULT (*SetPixelFormat)(ST_CI_DEVICE *pstDevice, STF_U32 u32PixelFormat);
    struct v4l2_rect (*QueryCropWin)(ST_CI_DEVICE *pstDevice);
    ST_RECT (*QueryCropWin_2)(ST_CI_DEVICE *pstDevice);
    struct v4l2_rect (*GetCropWin)(ST_CI_DEVICE *pstDevice);
    ST_RECT (*GetCropWin_2)(ST_CI_DEVICE *pstDevice);
    STF_RESULT (*SetCropWin)(ST_CI_DEVICE *pstDevice, struct v4l2_rect *pstRect);
    STF_RESULT (*SetCropWin_2)(ST_CI_DEVICE *pstDevice, ST_RECT *pstRect);
    STF_RESULT (*EnumFormat)(ST_CI_DEVICE *pstDevice);
    STF_RESULT (*GetFormat)(ST_CI_DEVICE *pstDevice, struct v4l2_format *pstFormat);
    STF_RESULT (*SetFormat)(ST_CI_DEVICE *pstDevice, STF_U16 u16Width, STF_U16 u16Height, STF_U32
u32PixelFormat);
    STF_RESULT (*TryFormat)(ST_CI_DEVICE *pstDevice, STF_U32 u32PixelFormat);
    STF_RESULT (*GetParam)(ST_CI_DEVICE *pstDevice, STF_U32 *pu32Numerator, STF_U32 *pu32Denominator);
    STF_RESULT (*SetParam)(ST_CI_DEVICE *pstDevice, STF_U32 u32Numerator, STF_U32 u32Denominator);
    STF_RESULT (*SetParam_2)(ST_CI_DEVICE *pstDevice, STF_U32 u32FPS);
    struct v4l2_rect (*GetSelection)(ST_CI_DEVICE *pstDevice);
    ST_RECT (*GetSelection_2)(ST_CI_DEVICE *pstDevice);
    STF_RESULT (*SetSelection)(ST_CI_DEVICE *pstDevice, struct v4l2_rect *pstRect);
    STF_RESULT (*SetSelection_2)(ST_CI_DEVICE *pstDevice, ST_RECT *pstRect);
    STF_RESULT (*SetStreamOn)(ST_CI_DEVICE *pstDevice, STF_BOOL8 bOn);
    STF_BOOL8 (*IsStreamOn)(ST_CI_DEVICE *pstDevice);
    STF_RESULT (*AllocateBuf)(ST_CI_DEVICE *pstDevice, STF_U32 u32Count, STF_BOOL8 bMmap, STF_BOOL8 bExport,
ST_DMA_BUF_INFO *pstDmaBufInfo);
    STF_RESULT (*AllocateAndEnqueueBuf)(ST_CI_DEVICE *pstDevice, STF_U32 u32AllocateCount, STF_U32
u32EnqueueCount, STF_BOOL8 bMmap, STF_BOOL8 bExport, ST_DMA_BUF_INFO *pstDmaBufInfo);
    ST_VDO_MEM *(*GetBuffer)(ST_CI_DEVICE *pstDevice, STF_U32 u32Index);
    ST_VDO_BUF (*GetBufInfo)(ST_CI_DEVICE *pstDevice, STF_U32 u32Index);
    STF_RESULT (*EnqueueBuf)(ST_CI_DEVICE *pstDevice, STF_U8 u8BufIdx);
    STF_RESULT (*EnqueueBuf_2)(ST_CI_DEVICE *pstDevice, STF_U32 u32Count, ST_VDO_BUF_INFO
*pstVideoBufferInfo);
    STF_RESULT (*EnqueueBufFromAvailableQueue)(ST_CI_DEVICE *pstDevice, STF_S32 s32Count);
    STF_RESULT (*EnqueueBufFromCompletedQueue)(ST_CI_DEVICE *pstDevice, STF_S32 s32Count);
    STF_RESULT (*DequeueBuf)(ST_CI_DEVICE *pstDevice, ST_VDO_MEM **ppstVdoMem);
    STF_RESULT (*DequeueBuf_2)(ST_CI_DEVICE *pstDevice, STF_S32 *ps32Count, ST_VDO_BUF_INFO
*pstVideoBufferInfo);
    STF_RESULT (*DequeueBufToCompletedQueue)(ST_CI_DEVICE *pstDevice, STF_S32 s32Count);
    STF_RESULT (*SwitchBufFromCompletedToAvailableQueue)(ST_CI_DEVICE *pstDevice, STF_S32 s32RemainCount);
    STF_RESULT (*PushBufToQueue)(ST_CI_DEVICE *pstDevice, sLinkedList_T *pstListQueue, ST_VDO_MEM *pstVdoMem,
STF_BOOL8 bIsPushToFront);
    ST_VDO_MEM *(*PopBufFromQueue)(ST_CI_DEVICE *pstDevice, sLinkedList_T *pstListQueue, STF_BOOL8
bIsPopFromBack);
    STF_RESULT (*ReleaseBuf)(ST_CI_DEVICE *pstDevice);
    STF_RESULT (*GenerateLinksInfo)(ST_CI_DEVICE *pstDevice, EN_SNSR_IF enSensorInterface, ST_ITEM_INFO
*pstLinksInfo);
    STF_RESULT (*ProcessLinks)(ST_CI_DEVICE *pstDevice, ST_ITEM_INFO *pstLinksInfo);
    STF_RESULT (*ProcessLinks_2)(ST_CI_DEVICE *pstDevice, EN_SNSR_IF enSensorInterface);
} ST_CI_DEVICE, *PST_CI_DEVICE;
```

The following table describes the fields in the above code block.

Table 4-15 Basic Device Field Description

Field	Description
pstCiConnection	The connection handle

Table 4-15 Basic Device Field Description (continued)

Field	Description
u8DeviceId	The device ID
u8DeviceType	The device type (FB, PP, DRM, Video, V4L2 sub-device, etc.)
szDeviceName	The device name
s32CiConnections	Indicate the number of applications using this connection handle
bConnected	Indicate whether the connection handle is open
enMemType	Indicate which video buffer type (**EN_MEM_TYPE_MMAP** or **EN_MEM_-TYPE_DMA**) will be allocated
pvDevParam	Point to the device parameters structure
***Destroy**	Destroy this device
***Connection**	Connect to the device driver
***Disconnect**	Disconnect form the device driver
***IsConnected**	Check the connection status
***GetConnections**	Get the number of connections
***GetConnection**	Get the connection handle
***GetFileHandle**	Get the file descriptor from the connection handle
***SubscribeEvent**[1]	Subscribe an event to the video device
***UnsubscribeEvent**[1]	Unsubscribe an event to the video device
***DequeueEvent**[1]	Dequeue an event from the video device
***InitDevice**	Device initialization
***GetPixelFormat**	Get the pixel format from the device
***SetPixelFormat**	Set the pixel format to the device
***QueryCropWin**[1]	Query the cropping window from the video device
***QueryCropWin_2**[1]	Query the cropping window from the video device
***GetCropWin**[1]	Get the cropping window from the video device
***GetCropWin_2**[1]	Get the cropping window from the video device
***SetCropWin**[1]	Set the cropping window into the video device
***SetCropWin_2**[1]	Set the cropping window into the video device
***EnumFormat**[1]	Enumerate supported formats from the video device
***GetFormat**[1]	Get the data format from the video device
***SetFormat**[1]	Set the data format into the video device
***TryFormat**[1]	Test if the video device supports this data format
***GetParam**[1]	Get streaming parameters from the video device
***SetParam**[1]	Set streaming parameters into the video device
***SetParam_2**[1]	Set streaming parameters into the video device

Table 4-15 Basic Device Field Description (continued)

Field	Description
***GetSelection**[1]	Get selection rectangle from the video device
***GetSelection_2**[1]	Get selection rectangle from the video device
***SetSelection**[1]	Set selection rectangle into the video device
***SetSelection_2**[1]	Set selection rectangle into the video device
***SetStreamOn**[1]	Start/Stop video device streaming I/O
***IsStreamOn**[1]	Check the streaming I/O status from the video device
***AllocateBuf**[2]	Allocate buffers for the device
***AllocateAndEnqueueBuf**[1]	Allocate and enqueue buffers into the video device
***GetBuffer**[1]	Get the video buffer from the video device
***GetBufInfo**[1]	Get the video buffer information from the video device
***EnqueueBuf**[1]	Enqueue buffer into the video device
***EnqueueBuf_2**[1]	Enqueue buffers into the video device
***EnqueueBufFromAvailableQueue**[1]	Enqueue buffers from available queue to the video device
***EnqueueBufFromCompletedQueue**[1]	Enqueue buffers from completed queue to the video device
***DequeueBuf**[1]	Dequeue buffers from the video device
***DequeueBuf_2**[1]	Dequeue buffers from the video device
***DequeueBufToCompletedQueue**[1]	Dequeue buffers from the video device to completed queue
***SwitchBufFromCompletedToAvailableQueue**[1]	Switch buffers from the completed queue to available queue
***PushBufToQueue**[1]	Push buffer into the queue
***PopBufFromQueue**[1]	Pop buffer from the queue
***ReleaseBuf**[2]	Release buffers from the device
***GenerateLinksInfo**[1]	Generate the media link information
***ProcessLinks**[1]	Process the media link settings
***ProcessLinks_2**[1]	Process the media link settings

Note:

1. The field items are for video device only.
2. The field items are for DRM and video device only.

4.1.16. Video Device Table

The following code block shows the data structure for the video device table structure.

```
typedef struct ST_VDEV_TBL {
    STF_U32 u32Count;
    ST_CI_DEVICE stVDev[EN_ISP_PORT_IDX_MAX + 1];
} ST_VDEV_TBL, *PST_VDEV_TBL;
```

The following table describes the fields in the above code block.

Table 4-16 Video Device Table Field Description

Field	Description
u32Count	Indicate the amount of devices in this table
stVDev	Device structure array

4.1.17. Video Device Pointer Table

The following code block shows the data structure for the video device pointer table structure.

```
typedef struct _ST_VDEV_PTR_TBL {
    STF_U32 u32Count;
    ST_CI_DEVICE *pstVDev[EN_ISP_PORT_IDX_MAX + 1];
} ST_VDEV_PTR_TBL, *PST_VDEV_PTR_TBL;
```

The following table describes the fields in the above code block.

Table 4-17 Video Device Pointer Table Field Description

Field	Description
u32Count	Indicate the amount of devices in this table
pstVDev	Device pointer structure array

4.2. API

The following topics show the application interfaces (API) of ISPC Library - Device.

4.2.1. STFLIB_ISP_DEVICE_FB_ConvertV4l2FormatToFbFormat

The API is described in the following sections.

Description

Convert the V4L2 pixel format to Frame buffer pixel format.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_DEVICE_FB_ConvertV4l2FormatToFbFormat(
    STF_U32 u32V4l2Format
    );
```

Parameter

The API has the following parameters.

Table 4-18 STFLIB_ISP_DEVICE_FB_ConvertV4l2FormatToFbFormat Parameter Description

Parameter	Description	Input/Output
u32V4l2Format	V4L2 pixel format	Input

Return

The API has the following return values.

Table 4-19 STFLIB_ISP_DEVICE_FB_ConvertV4l2FormatToFbFormat Return Description

Return	Description
STF_U32	Frame buffer pixel format

Requirement

The file `libISPC.a` is required for using this API.

4.2.2. STFLIB_ISP_DEVICE_DRM_ConvertV4l2FormatToDrmFormat

The API is described in the following sections.

Description

Convert the V4L2 pixel format to DRM device pixel format.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_DEVICE_DRM_ConvertV4l2FormatToDrmFormat(
    STF_U32 u32V4l2Format
    );
```

Parameter

The API has the following parameter.

Table 4-20 STFLIB_ISP_DEVICE_DRM_ConvertV4l2FormatToDrmFormat Parameter Description

Parameter	Description	Input/Output
u32V4l2Format	V4L2 pixel format	Input

Return

The API has the following return values.

Table 4-21 STFLIB_ISP_DEVICE_DRM_ConvertV4l2FormatToDrmFormat Return Description

Return	Description
STF_U32	DRM device pixel format

Requirement

The file `libISPC.a` is required for using this API.

4.2.3. STFLIB_ISP_DEVICE_StructInitialize

The API is described in the following sections.

Description

Device structure initialization function.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_DEVICE_StructInitialize(
    ST_CI_DEVICE *pstDevice,
    EN_DEV_ID enDevId,
    STF_BOOL8 bForceGenDev,
    EN_MEM_TYPE enMemType,
    STF_U16 u16Width,
    STF_U16 u16Height
    );
```

Parameter

The API has the following parameters.

Table 4-22 STFLIB_ISP_DEVICE_StructInitialize Parameter Description

Parameter	Description	Input/Output
pstDevice	Device handle	Input
enDevId	Device ID	Input
bForceGenDev	Indicate whether to open the video device using the “fopen” function	Input
enMemType	Video buffer type	Input
u16Width	Image width	Input
u16Height	Image height	Input

Return

The API has the following return values.

Table 4-23 STFLIB_ISP_DEVICE_StructInitialize Return Description

Return	Description
0	Success
Other values	Failure

Requirement

The file `libISPC.a` is required for using this API.

4.2.4. STFLIB_ISP_DEVICE_StructInitialize_2

The API is described in the following sections.

Description

Device structure initialization function.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_DEVICE_StructInitialize_2(
    ST_CI_DEVICE *pstDevice,
    EN_DEV_ID enDevId,
    STF_BOOL8 bForceGenDev,
    EN_MEM_TYPE enMemType,
    STF_U16 u16Width,
    STF_U16 u16Height,
    ST_RECT_2 stCropRect,
    STF_U8 u8Fps
```

```
    );
```

Parameter

The API has the following parameters.

Table 4-24 STFLIB_ISP_DEVICE_StructInitialize_2 Parameter Description

Parameter	Description	Input/Output
pstDevice	Device handle	Input
enDevId	Device ID	Input
bForceGenDev	Indicate whether to open the video device using the “fopen” function	Input
enMemType	Video buffer type	Input
u16Width	Image width	Input
u16Height	Image height	Input
stCropRect	Cropping window information	Input
u8Fps	Frame rate per second	Input

Return

The API has the following return values.

Table 4-25 STFLIB_ISP_DEVICE_StructInitialize_2 Return Description

Return	Description
0	Success
Other values	Failure

Requirement

The file `libISPC.a` is required for using this API.

4.2.5. STFLIB_ISP_DEVICE_StructInitializeWithDeviceName

The API is described in the following sections.

Description

Device structure initialization function with device name.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_DEVICE_StructInitializeWithDeviceName(
    ST_CI_DEVICE *pstDevice,
    STF_CHAR *pszDeviceName,
    STF_BOOL8 bForceGenDev,
    EN_MEM_TYPE enMemType,
    STF_U16 u16Width,
    STF_U16 u16Height
    );
```

Parameter

The API has the following parameters.

Table 4-26 STFLIB_ISP_DEVICE_StructInitializeWithDeviceName Parameter Description

Parameter	Description	Input/Output
pstDevice	Device handle	Input
pszDeviceName	Device name	Input
bForceGenDev	Indicate whether to open the video device using the “fopen” function	Input
enMemType	Video buffer type	Input
u16Width	Image width	Input
u16Height	Image height	Input

Return

The API has the following return values.

Table 4-27 STFLIB_ISP_DEVICE_StructInitializeWithDeviceName Return Description

Return	Description
0	Success
Other values	Failure

Requirement

The file `libISPC.a` is required for using this API.

4.2.6. STFLIB_ISP_DEVICE_StructInitializeWithDeviceName_2

The API is described in the following sections.

Description

Device structure initialization function with device name.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_DEVICE_StructInitializeWithDeviceName_2(
    ST_CI_DEVICE *pstDevice,
    STF_CHAR *pszDeviceName,
    STF_BOOL8 bForceGenDev,
    EN_MEM_TYPE enMemType,
    STF_U16 u16Width,
    STF_U16 u16Height
    ST_RECT_2 stCropRect,
    STF_U8 u8Fps
    );
```

Parameter

The API has the following parameters.

Table 4-28 STFLIB_ISP_DEVICE_StructInitializeWithDeviceName_2 Parameter Description

Parameter	Description	Input/Output
pstDevice	Device handle	Input

Table 4-28 STFLIB_ISP_DEVICE_StructInitializeWithDeviceName_2 Parameter Description (continued)

Parameter	Description	Input/Output
pszDeviceName	Device name	Input
bForceGenDev	Indicate whether to open the video device using the “fopen” function	Input
enMemType	Video buffer type	Input
u16Width	Image width	Input
u16Height	Image height	Input
stCropRect	Cropping window information	Input
u8Fps	Frame rate per second	Input

Return

The API has the following return values.

Table 4-29 STFLIB_ISP_DEVICE_StructInitializeWithDeviceName_2 Return Description

Return	Description
0	Success
Other values	Failure

Requirement

The file `libISPC.a` is required for using this API.

4.2.7. STFLIB_ISP_DEVICE_StructUninitialize

The API is described in the following sections.

Description

Device structure de-initialization function.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_DEVICE_StructUninitialize(
    ST_CI_DEVICE *pstDevice,
    );
```

Parameter

The API has the following parameter.

Table 4-30 STFLIB_ISP_DEVICE_StructUninitialize Parameter Description

Parameter	Description	Input/Output
pstDevice	Device handle	Input

Return

The API has the following return values.

Table 4-31 STFLIB_ISP_DEVICE_StructUninitialize Return Description

Return	Description
0	Success
Other values	Failure

Requirement

The file `libISPC.a` is required for using this API.

4.2.8. STFLIB_ISP_DEVICE_GenerateDeviceTable

The API is described in the following sections.

Description

Generate the media device table.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_DEVICE_GenerateDeviceTable(
    STF_CHAR *pszDriverName,
    STF_CHAR *pszModelName
    );
```

Parameter

The API has the following parameters.

Table 4-32 STFLIB_ISP_DEVICE_GenerateDeviceTable Parameter Description

Parameter	Description	Input/Output
pszDriverName	Driver name of media device	Input
pszModelName	Model name of media device	Input

Return

The API has the following return values.

Table 4-33 STFLIB_ISP_DEVICE_GenerateDeviceTable Return Description

Return	Description
0	Success
Other values	Failure

Requirement

The file `libISPC.a` is required for using this API.

5. ISPC Library - Pipeline

For the convenience of using the ISP SDK, here we implement a pipeline library. This library uses a data structure to integrate the device, sensor, ISP context, module, control, etc., and encapsulates many functions in this structure to operate the devices, modules and controls.

Customer can use this library to manage the video devices, allocate video buffers, manage captured images, control the ISP hardware, and handle the sensor behavior.

5.1. Data Structure

The following topics show the data structure of ISPC Library - Pipeline.

5.1.1. Version

The following code block shows the data structure for the version structure.

```
typedef union _ST_VERSION {
    STF_U32 u32Version;
    struct {
        STF_U8 u8BuildVersion;
        STF_U8 u8MinorVersion;
        STF_U16 u16MajorVersion;
    } field;
} ST_VERSION, *PST_VERSION;
```

The following table describes the fields in the above code block.

Table 5-1 Version Field Description

Field	Description
u32Version	32-bit version data
u8BuildVersion	Build version
u8MinorVersion	Minor version
u16MajorVersion	Major version

5.1.2. Header

The following code block shows the data structure for the header structure.

```
typedef struct _ST_HEADER {
    STF_U8 u8Id[4];
    STF_U32 u32DataSize;
    ST_VERSION stVersion;
} ST_HEADER, *PST_HEADER;
```

The following table describes the fields in the above code block.

Table 5-2 Header Field Description

Field	Description
u8Id	Used to identify the binary configuration file
u32DataSize	Indicate whole of binary setting file size. It is including the header, module/control information, module/control parameters and checksum
stVersion	Indicate the module/control parameters version

5.1.3. Module and Control Info

The following code block shows the data structure for the module control and information structure.

```
typedef struct _ST_MOD_CTL_INFO {
    STF_U8 u8Id;
    STF_U32 u32Offset;
    STF_U16 u16ParamSize;
} ST_MOD_CTL_INFO, *PST_MOD_CTL_INFO;
```

The following table describes the fields in the above code block.

Table 5-3 Module and Control Info Field Description

Field	Description
u8Id	Module/Control ID
u32Offset	location
u16ParamSize	Parameters size

5.1.4. Binary Parameter

The following code block shows the data structure for the module control and information structure.

```
typedef struct _ST_ISP_BIN_PARAM {
    ST_HEADER stHeader;
    STF_U8 u8Count;
    ST_MOD_CTL_INFO pstModCtlInfo[];
} ST_ISP_BIN_PARAM, *PST_ISP_BIN_PARAM;
```

The following table describes the fields in the above code block.

Table 5-4 Binary Parameter Field Description

Field	Description
stHeader	Header section
u8Count	Indicate how many modules/controls in this file
pstModCtlInfo	Module/Control location and size information

5.1.5. RAW dump table

The following code block shows the data structure for the RAW dump table structure.

```
typedef struct _ST_RAW_DUMP_TBL {
    STF_U8 u8Idx;
    STF_U8 u8TblCnt;
    STF_U8 u8CaptureCount;
    CI_BUFFTYPE enBufferType;
    CI_MEM_PARAM stDumpMem[CONTIGUOUS_CAPTURE_BUF_MAX];
} ST_RAW_DUMP_TBL, *PST_RAW_DUMP_TBL;
```

The following table describes the fields in the above code block.

Table 5-5 RAW Dump Table Field Description

Field	Description
u8Idx	Indicate current use buffer
u8TblCnt	Indicate total allocate buffer count

Table 5-5 RAW Dump Table Field Description (continued)

Field	Description
u8CaptureCount	Indicate how many images to capture
enBufferType	Indicate those buffers will be used for which image buffer
stDumpMem	RAW dump table

5.1.6. Shot Table

The following code block shows the data structure for the shot table structure.

```
typedef struct _ST_SHOT_TBL {
    STF_U8 u8TblCnt;
    ST_SHOT_INFO stShot[PIPELINE_IMG_BUF_MAX];
} ST_SHOT_TBL, *PST_SHOT_TBL;
```

The following table describes the fields in the above code block.

Table 5-6 Shot Table Field Description

Field	Description
u8TblCnt	Indicate how many shots are included in this table
stShot	Shot table

5.1.7. Shot Queue

The following code block shows the data structure for the shot queue structure.

```
typedef struct _ST_SHOT_QUEUE {
    STF_U8 u8TblCnt;
    STF_U8 u8TblIdx;
    STF_CHAR szName[16];
    ST_SHOT_INFO *pstShot[PIPELINE_IMG_BUF_MAX];
} ST_SHOT_QUEUE, *PST_SHOT_QUEUE;
```

The following table describes the fields in the above code block.

Table 5-7 Shot Queue Field Description

Field	Description
u8TblCnt	Indicate how many shots are included in this table
u8TblIdx	Indicate the next shot afterwards
szName	Queue name
pstShot	Shot pointer table

5.1.8. ISP Context

The following code block shows the data structure for the ISP context structure.

```
typedef struct _ST_ISP_CTX {
    STF_VOID *pPipeline;
    STF_U8 u8IspIdx;
    ST_SIZE stImgSize;
    STF_U8 *pu8BinParamBuf;
    ST_ISP_BIN_PARAM *pstIspBinParam;
    STF_BOOL8 m_bIqTuningDebugInfoEnable;
```

```
    ST_ISP_MOD *pstIspModCtx[EN_MODULE_ID_MAX];
    ST_ISP_CTL *pstIspCtlCtx[EN_CONTROL_ID_MAX];
    STF_BOOL8 m_bUserConfigIsp;
    ST_CI_DEVICE stVideoDevice[EN_ISP_PORT_ID_MAX];
} ST_ISP_CTX, *PST_ISP_CTX;
```

The following table describes the fields in the above code block.

Table 5-8 ISP Context Field Description

Field	Description
pPipeline	Pipeline handle
u8IspIdx	Indicated which ISP is used
stImgSize	Capture size
pu8BinParamBuf	Store binary parameters data
pstIspBinParam	Pointer to binary parameters header
bIqTuningDebugInfoEnable	Debug information output flag
pstIspModCtx	Module table
pstIspCtlCtx	Control table
bUserConfigIsp	Enable/Disable ISP initialization from V4L2 driver
stVideoDevice	Video device table

5.1.9. Pipeline

The following code block shows the data structure for the pipeline structure.

```
typedef struct _ST_PIPELINE {
    STF_U8 u8IspIdx;
    ST_ISP_CTX stIspCtx;
    ST_SENSOR *pstSensor;
    CI_CONNECTION *pstCIConnection;
    STF_U16 u16MaxUoWidth;
    STF_U16 u16MaxUoHeight;
    STF_U16 u16MaxSs0Width;
    STF_U16 u16MaxSs0Height;
    STF_U16 u16MaxSs1Width;
    STF_U16 u16MaxSs1Height;
    STF_U16 u16MaxDumpWidth;
    STF_U16 u16MaxDumpHeight;
    STF_U16 u16MaxTiling_1_RdWidth;
    STF_U16 u16MaxTiling_1_RdHeight;
    STF_U16 u16MaxTiling_1_WrWidth;
    STF_U16 u16MaxTiling_1_WrHeight;
    EN_CTX_STATUS enCtxStatus;
    STF_BOOL8 bBufAllocated;
    ST_SHOT_INFO *pstAvailableShot;
    ST_SHOT_INFO *pstPendingShot;
    ST_SHOT_INFO *pstCompletedShot;
    ST_SHOT_INFO *pstOutputShot;
    ST_SHOT_INFO *pstRtspShot;
    ST_SHOT_INFO *pstTemporaryShot;
    ST_SHOT_TBL stImgShot;
    ST_SHOT_QUEUE stAvailableShotQueue;
    ST_SHOT_QUEUE stPendingShotQueue;
    ST_SHOT_QUEUE stCompletedShotQueue;
    ST_SHOT_QUEUE stOutputShotQueue;
    ST_SHOT_QUEUE stRtspShotQueue;
    ST_SHOT_QUEUE stTemporaryShotQueue;
    STF_BOOL8 bContiguousRawCaptureMode;
    ST_RAW_DUMP_TBL stRawDump;
    ST_VDEV_PTR_TBL stPipelineVdoDevTbl;
    EN_ISP_PORT enIspPort;
    STF_S8 s8PipelineModCnt;
    ST_ISP_MOD *pstPipelineMod[EN_MODULE_ID_MAX + 1];
    STF_S8 s8PipelineCtlCnt;
    ST_ISP_CTL *pstPipelineCtl[EN_CONTROL_ID_MAX + 1];
    STF_U64 u64FrameCount;

    CI_CONNECTION *(*GetConnection)(ST_PIPELINE *pstPipeline);
    ST_GLOBAL_SETUP (*GetGlobalSetup)(ST_PIPELINE *pstPipeline, STF_RESULT *pRet);
    ST_SENSOR *(*GetSensor)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*SetSensor)(ST_PIPELINE *pstPipeline, ST_SENSOR *pstSensor);
    ST_ISP_CTX *(*GetIspContext)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*BufAndShotInit)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*BufAllocate)(ST_PIPELINE *pstPipeline, STF_U8 u8BufCnt, STF_U8 u8BufType, STF_BOOL8 bClear, STF_BOOL8 bEnqueueBuf, ST_DMA_BUF_INFO
 *pstDmaBufInfo);
    STF_RESULT (*BufFree)(ST_PIPELINE *pstPipeline);
```

```
    STF_RESULT (*SetPipelineReady)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*BufGetDirectly)(ST_PIPELINE *pstPipeline, STF_U8 u8Idx, EN_PIPELINE_BUF_ID enBufId, CI_MEM_PARAM **ppstMemParam);
    STF_RESULT (*ShotGetDirectly)(ST_PIPELINE *pstPipeline, STF_U8 u8Idx, ST_SHOT_INFO **ppstShotInfo);
    STF_U32 (*ShotGetAllocatedCount)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*ShotAssignToCompleted)(ST_PIPELINE *pstPipeline, ST_SHOT_INFO *pstShotInfo);
    STF_RESULT (*ShotSwitchToCompletedFromVDev)(ST_PIPELINE *pstPipeline, ST_VDO_MEM **ppstUoVdoMem);
    STF_RESULT (*ShotDequeueBufToCompletedFromVDev)(ST_PIPELINE *pstPipeline, ST_VDO_MEM **ppstUoVdoMem);
    STF_RESULT (*ShotDequeueBufFromVDev)(ST_PIPELINE *pstPipeline, ST_SHOT_INFO *pstShotInfo, ST_VDO_MEM **ppstUoVdoMem);
    STF_RESULT (*ShotEnqueueBufToVDev)(ST_PIPELINE *pstPipeline, ST_SHOT_INFO *pstShotInfo);
    STF_RESULT (*ShotPush)(ST_PIPELINE *pstPipeline, ST_SHOT_QUEUE *pstShotQueue, ST_SHOT_INFO *pstShotInfo);
    STF_RESULT (*ShotPop)(ST_PIPELINE *pstPipeline, ST_SHOT_QUEUE *pstShotQueue, ST_SHOT_INFO **ppstShotInfo);
    STF_RESULT (*ShotPopLast)(ST_PIPELINE *pstPipeline, ST_SHOT_QUEUE *pstShotQueue, ST_SHOT_INFO **ppstShotInfo);
    STF_RESULT (*ShotSwitchToAvailable)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*ShotSwitchToAvailableSkipBackupShot)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*ShotSwitchToPending)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*ShotSwitchToCompleted)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*ShotSwitchToOutput)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*ShotSwitchToRtsp)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*ShotSwitchRtspToAvailable)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*ShotSwitchToTemporary)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*ShotSwitchTemporaryToAvailable)(ST_PIPELINE *pstPipeline);
    STF_U32 (*ShotGetAvailableCount)(ST_PIPELINE *pstPipeline);
    STF_U32 (*ShotGetPendingCount)(ST_PIPELINE *pstPipeline);
    STF_U32 (*ShotGetCompletedCount)(ST_PIPELINE *pstPipeline);
    STF_U32 (*ShotGetOutputCount)(ST_PIPELINE *pstPipeline);
    STF_U32 (*ShotGetRtspCount)(ST_PIPELINE *pstPipeline);
    STF_U32 (*ShotGetTemporaryCount)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*ShotGetShotInfo)(ST_PIPELINE *pstPipeline, ST_SHOT_INFO **ppstShotInfo);
    STF_RESULT (*ShotGetAvailableShot)(ST_PIPELINE *pstPipeline, ST_SHOT_INFO **ppstShotInfo);
    STF_RESULT (*ShotGetPendingShot)(ST_PIPELINE *pstPipeline, ST_SHOT_INFO **ppstShotInfo);
    STF_RESULT (*ShotGetCompletedShot)(ST_PIPELINE *pstPipeline, ST_SHOT_INFO **ppstShotInfo);
    STF_RESULT (*ShotGetOutputShot)(ST_PIPELINE *pstPipeline, ST_SHOT_INFO **ppstShotInfo);
    STF_RESULT (*ShotGetRtspShot)(ST_PIPELINE *pstPipeline, ST_SHOT_INFO **ppstShotInfo);
    STF_RESULT (*ShotGetTemporaryShot)(ST_PIPELINE *pstPipeline, ST_SHOT_INFO **ppstShotInfo);
    STF_RESULT (*ShotGetStatisticsBuf)(ST_PIPELINE *pstPipeline, EN_STAT_BUF_TYPE enStatBufType, STF_BOOL8 *pbIsScDumpForAe, STF_U32 *pu32ScDumpCount,
STF_VOID **ppvBuffer);
    STF_RESULT (*ShotSetOutputAndStatisticsBufInfo)(ST_PIPELINE *pstPipeline, STF_BOOL8 bIsScDumpFo-rAe);
    STF_RESULT (*ShotSetOutputAndStatisticsBuf)(ST_PIPELINE *pstPipeline, STF_BOOL8 bIsScDumpForAe);
    STF_BOOL8 (*ShotGetContiguousCaptureMode)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*ShotSetContiguousCaptureMode)(ST_PIPELINE *pstPipeline, STF_BOOL8 bContiguousRaw-CaptureMode, STF_U16 u16BufferType, STF_U8
u8CaptureCount);
    STF_U32 (*ShotGetContiguousCaptureBufferCount)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*ShotGetContiguousCaptureBuffer)(ST_PIPELINE *pstPipeline, STF_U8 u8Idx, CI_MEM_PARAM **ppstDumpMem);
    STF_RESULT (*InitVideoDevice)(ST_PIPELINE *pstPipeline, EN_ISP_PORT enIspPort, EN_SNSR_IF enSen-sorInterface, STF_U32 u32PixelFormat);
    ST_CI_DEVICE *(*GetVideoDevice)(ST_PIPELINE *pstPipeline, EN_ISP_PORT_ID enIspPortId);
    ST_CI_DEVICE *(*GetVideoDevice2)(ST_PIPELINE *pstPipeline, CI_BUFFTYPE enBufferType);
    STF_RESULT (*RegisterModule)(ST_PIPELINE *pstPipeline, ST_ISP_MOD *pstIspMod);
    STF_RESULT (*RegisterModuleById)(ST_PIPELINE *pstPipeline, EN_MODULE_ID enModuleId);
    STF_RESULT (*RegisterEnabledModules)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*RegisterAllOfModules)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*UnregisterModule)(ST_PIPELINE *pstPipeline, EN_MODULE_ID enModuleId);
    STF_RESULT (*UnregisterDisabledModules)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*UnregisterAllOfModules)(ST_PIPELINE *pstPipeline);
```

```
    ST_ISP_MOD *(*GetModule)(ST_PIPELINE *pstPipeline, EN_MODULE_ID enModuleId);
    ST_ISP_MOD *(*GetRegisteredModule)(ST_PIPELINE *pstPipeline, EN_MODULE_ID enModuleId);
    STF_CHAR *(*GetModuleName)(ST_PIPELINE *pstPipeline, EN_MODULE_ID enModuleId);
    STF_CHAR *(*GetRegisteredModuleName)(ST_PIPELINE *pstPipeline, EN_MODULE_ID enModuleId);
    STF_VOID *(*GetModuleRdmaBuf)(ST_PIPELINE *pstPipeline, EN_MODULE_ID enModuleId);
    STF_VOID *(*GetRegisteredModuleRdmaBuf)(ST_PIPELINE *pstPipeline, EN_MODULE_ID enModuleId);
    STF_VOID *(*GetModuleIspRdma)(ST_PIPELINE *pstPipeline, EN_MODULE_ID enModuleId);
    STF_VOID *(*GetRegisteredModuleIspRdma)(ST_PIPELINE *pstPipeline, EN_MODULE_ID enModuleId);
    STF_VOID *(*GetModuleRdma)(ST_PIPELINE *pstPipeline, EN_MODULE_ID enModuleId);
    STF_VOID *(*GetRegisteredModuleRdma)(ST_PIPELINE *pstPipeline, EN_MODULE_ID enModuleId);
    STF_VOID *(*GetModuleParam)(ST_PIPELINE *pstPipeline, EN_MODULE_ID enModuleId);
    STF_VOID *(*GetRegisteredModuleParam)(ST_PIPELINE *pstPipeline, EN_MODULE_ID enModuleId);
    STF_RESULT (*RegisterControl)(ST_PIPELINE *pstPipeline, ST_ISP_CTL *pstIspCtl);
    STF_RESULT (*RegisterControlById)(ST_PIPELINE *pstPipeline, EN_CONTROL_ID enControlId);
    STF_RESULT (*RegisterEnabledControls)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*RegisterAllOfControls)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*UnregisterControl)(ST_PIPELINE *pstPipeline, EN_CONTROL_ID enControlId);
    STF_RESULT (*UnregisterDisabledControls)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*UnregisterAllOfControls)(ST_PIPELINE *pstPipeline);
    ST_ISP_CTL *(*GetControl)(ST_PIPELINE *pstPipeline, EN_CONTROL_ID enControlId);
    ST_ISP_CTL *(*GetRegisteredControl)(ST_PIPELINE *pstPipeline, EN_CONTROL_ID enControlId);
    STF_CHAR *(*GetControlName)(ST_PIPELINE *pstPipeline, EN_CONTROL_ID enControlId);
    STF_CHAR *(*GetRegisteredControlName)(ST_PIPELINE *pstPipeline, EN_CONTROL_ID enControlId);
    STF_VOID *(*GetControlParam)(ST_PIPELINE *pstPipeline, EN_CONTROL_ID enControlId);
    STF_VOID *(*GetRegisteredControlParam)(ST_PIPELINE *pstPipeline, EN_CONTROL_ID enControlId);
    STF_RESULT (*RegisterEnabledModulesAndControls)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*RegisterAllOfModulesAndControls)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*UnregisterDisabledModulesAndControls)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*UnregisterAllOfModulesAndControls)(ST_PIPELINE *pstPipeline);
    STF_BOOL8 (*IsModuleEnable)(ST_PIPELINE *pstPipeline, EN_MODULE_ID enModuleId);
    STF_RESULT (*ModuleEnable)(ST_PIPELINE *pstPipeline, EN_MODULE_ID enModuleId, STF_BOOL8 bEna-ble);
    STF_BOOL8 (*IsModuleUpdate)(ST_PIPELINE *pstPipeline, EN_MODULE_ID enModuleId);
    STF_RESULT (*ModuleUpdate)(ST_PIPELINE *pstPipeline, EN_MODULE_ID enModuleId, STF_BOOL8 bUpdate);
    STF_BOOL8 (*IsControlEnable)(ST_PIPELINE *pstPipeline, EN_CONTROL_ID enControlId);
    STF_RESULT (*ControlEnable)(ST_PIPELINE *pstPipeline, EN_CONTROL_ID enControlId, STF_BOOL8 bEna-ble);
    STF_RESULT (*ClearModulesAndControls)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*PrintRegisteredModulesAndControls)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*InitAll)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*InitAllGlobals)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*InitAllModules)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*InitAllControls)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*InitPipelineModulesAndControls)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*SetNextRdma)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*UpdateAll)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*UpdateAllGlobals)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*UpdateAllModules)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*CalculateAllControls)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*UpdateModule)(ST_PIPELINE *pstPipeline, EN_MODULE_ID enModuleId);
    STF_RESULT (*CalculateControl)(ST_PIPELINE *pstPipeline, EN_CONTROL_ID enControlId);
    STF_RESULT (*UpdateAllRequested)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*UpdateAllRequestedModules)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*CalculateAllRequestedControls)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*SetupAll)(ST_PIPELINE *pstPipeline);
```

```
    STF_RESULT (*SetupAllGlobals)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*SetupAllModules)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*SetupModule)(ST_PIPELINE *pstPipeline, EN_MODULE_ID enModuleId);
    STF_RESULT (*SetupRequested)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*ModuleGetIqParam)(ST_PIPELINE *pstPipeline, EN_MODULE_ID enModuleId, STF_VOID *pParamBuf, STF_U16 *pu16ParamSize);
    STF_RESULT (*ModuleSetIqParam)(ST_PIPELINE *pstPipeline, EN_MODULE_ID enModuleId, STF_VOID *pParamBuf, STF_U16 u16ParamSize);
    STF_RESULT (*ControlGetIqParam)(ST_PIPELINE *pstPipeline, EN_CONTROL_ID enControlId, STF_VOID *pParamBuf, STF_U16 *pu16ParamSize);
    STF_RESULT (*ControlSetIqParam)(ST_PIPELINE *pstPipeline, EN_CONTROL_ID enControlId, STF_VOID *pParamBuf, STF_U16 u16ParamSize);
    STF_RESULT (*LoadBinParam)(ST_PIPELINE *pstPipeline, STF_CHAR *pszBinSettingFilename);
    STF_RESULT (*LoadBinClbrt)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*VerifyConfiguration)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*UpdateCiPipeline)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*ProgramPipeline)(ST_PIPELINE *pstPipeline, STF_BOOL8 bUpdateASAP);
    STF_RESULT (*AddShots)(ST_PIPELINE *pstPipeline, STF_U8 u8Num);
    STF_RESULT (*DeleteShots)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*AcquireShot)(ST_PIPELINE *pstPipeline, ST_SHOT *pstShot, STF_BOOL8 bBlocking);
    STF_RESULT (*ProgramSpecifiedShot)(ST_PIPELINE *pstPipeline, CI_BUFFID *pstBuffId);
    STF_RESULT (*ProgramShot)(ST_PIPELINE *pstPipeline);
    STF_VOID (*ProcessShot)(ST_PIPELINE *pstPipeline, ST_SHOT *pstShot, CI_SHOT *pstCIShotBuffer);
    STF_RESULT (*ReleaseShot)(ST_PIPELINE *pstPipeline, ST_SHOT *pstShot);
    STF_RESULT (*AllocateBuffer)(ST_PIPELINE *pstPipeline, CI_BUFFTYPE enBufferType, STF_U32 u32Size, STF_U32 *pu32BufferId);
    STF_RESULT (*ImportBuffer)(ST_PIPELINE *pstPipeline, CI_BUFFTYPE enBufferType, STF_U32 u32IonFd, STF_U32 u32Size, STF_U32 *pu32BufferId);
    STF_RESULT (*DeregisterBuffer)(ST_PIPELINE *pstPipeline, STF_U32 u32BufferID);
    ST_RES (*GetUoDimensions)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*SetUoDimensions)(ST_PIPELINE *pstPipeline, STF_U16 u16Width, STF_U16 u16Height);
    ST_RES (*GetSs0Dimensions)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*SetSs0Dimensions)(ST_PIPELINE *pstPipeline, STF_U16 u16Width, STF_U16 u16Height);
    ST_RES (*GetSs1Dimensions)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*SetSs1Dimensions)(ST_PIPELINE *pstPipeline, STF_U16 u16Width, STF_U16 u16Height);
    ST_RES (*GetDumpDimensions)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*SetDumpDimensions)(ST_PIPELINE *pstPipeline, STF_U16 u16Width, STF_U16 u16Height);
    ST_RES (*GetTiling_1_ReadDimensions)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*SetTiling_1_ReadDimensions)(ST_PIPELINE *pstPipeline, STF_U16 u16Width, STF_U16 u16Height);
    ST_RES (*GetTiling_1_WriteDimensions)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*SetTiling_1_WriteDimensions)(ST_PIPELINE *pstPipeline, STF_U16 u16Width, STF_U16 u16Height);
    STF_RESULT (*StartCapture)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*StopCapture)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*GetFirstAvailableBuffers)(ST_PIPELINE *pstPipeline, CI_BUFFID *pstBuffId);
    STF_RESULT (*Register)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*SetActiveCapture)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*ClearActiveCapture)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*ResetFrameCount)(ST_PIPELINE *pstPipeline);
    STF_U64 (*GetFrameCount)(ST_PIPELINE *pstPipeline);
    STF_RESULT (*SetPendingCount)(ST_PIPELINE *pstPipeline, STF_U32 u32PendingCount);
    STF_U32 (*IncPendingCount)(ST_PIPELINE *pstPipeline);
    STF_U32 (*GetCompletedCount)(ST_PIPELINE *pstPipeline, STF_BOOL8 bBlocking);
    STF_RESULT (*SetCompletedCount)(ST_PIPELINE *pstPipeline, STF_U32 u32CompletedCount);
    STF_RESULT (*GetPendingCompletedCount)(ST_PIPELINE *pstPipeline, STF_BOOL8 bBlocking, STF_BOOL8 *pbHwCapture, STF_U32 *pu32PendingCount, STF_U32
 *pu32CompletedCount);
} ST_PIPELINE, *PST_PIPELINE;
```

The following table describes the fields in the above code block.

Table 5-9 Pipeline Field Description

Field	Description
u8IspIdx	Indicated which ISP is used
stIspCtx	ISP context structure, this structure store the modules, controls, and videos handle
pstSensor	Sensor handle
pstCIConnection	Connection handle
u16MaxUoWidth[1]	Store the UO capture image width. Not used in the V4L2 driver
u16MaxUoHeight[1]	Store the UO capture image height. Not used in the V4L2 driver
u16MaxSs0Width[1]	Store the SS0 capture image width. Not used in the V4L2 driver
u16MaxSs0Height[1]	Store the SS0 capture image height. Not used in the V4L2 driver
u16MaxSs1Width[1]	Store the SS1 capture image width. Not used in the V4L2 driver
u16MaxSs1Height[1]	Store the SS1 capture image height. Not used in the V4L2 driver
u16MaxDumpWidth[1]	Store the DUMP capture image width. Not used in the V4L2 driver
u16MaxDumpHeight[1]	Store the DUMP capture image height. Not used in the V4L2 driver
u16MaxTiling_1_RdWidth[1]	Store the TIL_1_RD capture image width. Not used in the V4L2 driver
u16MaxTiling_1_RdHeight[1]	Store the TIL_1_RD capture image height. Not used in the V4L2 driver
u16MaxTiling_1_WrWidth[1]	Store the TIL_1_WR capture image width. Not used in the V4L2 driver
u16MaxTiling_1_WrHeight[1]	Store the TIL_1_WR capture image height. Not used in the V4L2 driver
enCtxStatus	Context status, used to indicate the pipeline status
bBufAllocated	Buffers allocated flag, used to indicate whether the video buffer is allocated
pstAvailableShot	Point to an available shot
pstPendingShot[1]	Point to a pending shot. Not used in the V4L2 driver
pstCompletedShot	Point to a completed shot
pstOutputShot	Point to a output shot
pstRtspShot[2]	Point to a RTSP shot. No longer in use
pstTemporaryShot[2]	Point to a temporary shot. No longer in use
stImgShot	Image shot table, store all of allocated image buffer
stAvailableShotQueue	Available shot queue
stPendingShotQueue[1]	Pending shot queue. Not used in the V4L2 driver
stCompletedShotQueue	Completed shot queue
stOutputShotQueue	Output shot queue
stRtspShotQueue[2]	RTSP shot queue. No longer in use
stTemporaryShotQueue[2]	Temporary shot queue. No longer in use
stRawDump[1]	Contiguous RAW image dump buffers table. Not used in the V4L2 driver
stPipelineVdoDevTbl	Video device table, which stores video devices registered to the pipeline

Table 5-9 Pipeline Field Description (continued)

Field	Description
enIspPort	Video device flag, which stores video devices registered to the pipeline
s8PipelineModCnt	The number of modules registered to the pipeline
pstPipelineMod	Module table, which stores modules registered to the pipeline
s8PipelineCtlCnt	The number of controls registered to the pipeline
pstPipelineCtl	Control table, which stores controls registered to the pipeline
u64FrameCount[1]	Frame count. Not used in the V4L2 driver
***GetConnection**	Get the ISP driver connection handle
***GetGlobalSetup**[1]	Get all of image buffer information. Not used in the V4L2 driver
***GetSensor**	Get the sensor handle
***SetSensor**	Set the sensor handle to the pipeline
***GetIspContext**	Get the ISP context handle
***BufAndShotInit**	Initialize all of buffer, shot pointers and queues
***BufAllocate**	Allocate video buffers for video device
***BufFree**	Free the allocated video buffers
***SetPipelineReady**	Change pipeline status to ready status
***BufGetDirectly**	Get a specific buffer from the video device
***ShotGetDirectly**	Get a specific shot from the image shot table
***ShotGetAllocatedCount**	Get count of allocated shot buffers
***ShotAssignToCompleted**	Assign a shot to completed shot pointer
***ShotSwitchToCompleted-FromVDev**	Pop buffer from the video devices and pack them to a shot, and then push this shot into the completed queue
***ShotDequeueBufToCom-pletedFromVDev**	Dequeue buffer from the video devices and pack them to a shot, and then push this shot into the completed queue
***ShotDequeueBufFromVDev**	Dequeue buffer from the video devices and pack them to a shot
***ShotEnqueueBufToVDev**	Enqueue buffer from shot to video devices
***ShotPush**	Push a shot into a queue
***ShotPop**	Pop a shot from a queue
***ShotPopLast**[2]	Pop the last shot from a queue. No longer in use
***ShotSwitchToAvailable**	Switch shot from output queue to available queue, and assign this shot to the output pointer
***ShotSwitchToAvailableSkip-BackupShot**[1]	Switch shot from output queue to available queue. Not used in the V4L2
***ShotSwitchToPending**[1]	Switch shot from available queue to pending queue, and assign this shot to the available pointer. Not used in the V4L2 driver
***ShotSwitchToCompleted**[1]	Switch shot from pending queue to completed queue, and assign this shot to the pending pointer. Not used in the V4L2 driver

Table 5-9 Pipeline Field Description (continued)

Field	Description
***ShotSwitchToOutput**[1]	Switch shot from completed queue to output queue, and assign this shot to the completed pointer. Not used in the V4L2 driver
***ShotSwitchToRtsp**[2]	Switch shot from output queue to RTSP queue, and a assign this shot to the RTSP pointer. No longer in use
***ShotSwitchRtspToAvailable**[2]	Switch shot from RTSP queue to Available queue. No longer in use
***ShotSwitchToTemporary**[2]	Switch shot from output queue to temporary queue, and assign this shot to the temporary pointer. No longer in use
***ShotSwitchTemporaryToAvailable**[2]	Switch shot from temporary queue to Available queue. No longer in use
***ShotGetAvailableCount**[1]	Get available shot count. Not used in the V4L2 driver
***ShotGetPendingCount**[1]	Get pending shot count. Not used in the V4L2 driver
***ShotGetCompletedCount**[1]	Get completed shot count. Not used in the V4L2 driver
***ShotGetOutputCount**[1]	Get output shot count. Not used in the V4L2 driver
***ShotGetRtspCount**[2]	Get RTSP shot count. No longer in use
***ShotGetTemporaryCount**[2]	Get temporary shot count. No longer in use
***ShotGetShotInfo**	Get a shot information
***ShotGetAvailableShot**[1]	Get an available shot. Not used in the V4L2 driver
***ShotGetPendingShot**[1]	Get a pending shot. Not used in the V4L2 driver
***ShotGetCompletedShot**[1]	Get a completed shot. Not used in the V4L2 driver
***ShotGetOutputShot**[1]	Get an output shot. Not used in the V4L2 driver
***ShotGetRtspShot**[2]	Get a RTSP shot. No longer in use
***ShotGetTemporaryShot**[2]	Get a temporary shot. No longer in use
***ShotGetStatisticsBuf**	Get a statistics buffer from completed shot
***ShotSetOutputAndStatisticsBufInfo**[1]	Set output and statistics buffer information to related modules. Not used in the V4L2 driver
***ShotSetOutputAndStatisticsBuf**[1]	Set output and statistics buffer to related modules. Not used in the V4L2 driver
***ShotGetContiguousCaptureMode**[1]	Get contiguous capture mode. Not used in the V4L2 driver
***ShotSetContiguousCaptureMode**[1]	Set contiguous capture mode. Not used in the V4L2 driver
***ShotGetContiguousCaptureBufferCount**[1]	Get contiguous capture buffer count. Not used in the V4L2 driver
***ShotGetContiguousCaptureBuffer**	Get a specific contiguous capture buffer. Not used in the V4L2 driver
***InitVideoDevice**	Video devices initialization
***GetVideoDevice**	Get a specific video device handle from the ISP context

Table 5-9 Pipeline Field Description (continued)

Field	Description
***GetVideoDevice2**	Get a specific video device handle from the ISP context
***RegisterModule**	Register a module in the pipeline
***RegisterModuleById**	Use module ID to register a module in the pipeline
***RegisterEnabledModules**	Register all enabled modules in the pipeline
***RegisterAllOfModules**	Register all the modules in the pipeline
***UnregisterModule**	Unregister a module from the pipeline
***UnregisterDisabledModules**	Unregister all disabled modules from the pipeline
***UnregisterAllOfModules**	Unregister all the modules from the pipeline
***GetModule**	Get a specific module handle form the ISP context
***GetRegisteredModule**	Get a specific registered module handle form the pipeline
***GetModuleName**	Get a specific module name form the ISP context
***GetRegisteredModuleName**	Get a specific registered module name form the pipeline
***GetModuleRdmaBuf**	Get a specific module Rdma buffer form the ISP context
***GetRegisteredModule-RdmaBuf**	Get a specific registered module Rdma buffer form the pipeline
***GetModuleIspRdma**	Get a specific module IspRdma form the ISP context
***GetRegisteredModuleIsp-Rdma**	Get a specific registered module IspRdma form the pipeline
***GetModuleRdma**	Get a specific module Rdma form the ISP context
***GetRegisteredModuleRdma**	Get a specific registered module Rdma form the pipeline
***GetModuleParam**	Get a specific module parameter form the ISP context
***GetRegisteredModule-Param**	Get a specific registered module parameter form the pipeline
***RegisterControl**	Register a control in the pipeline
***RegisterControlById**	Use control ID to register a control in the pipeline
***RegisterEnabledControls**	Register all enabled controls in the pipeline
***RegisterAllOfControls**	Register all of controls in the pipeline
***UnregisterControl**	Unregister a control from the pipeline
***UnregisterDisabledControls**	Unregister all disabled controls from the pipeline
***UnregisterAllOfControls**	Unregister all of controls from the pipeline
***GetControl**	Get a specific control handle form the ISP context
***GetRegisteredControl**	Get a specific registered control handle form the pipeline
***GetControlName**	Get a specific control name form the ISP context
***GetRegisteredControlName**	Get a specific registered control name form the pipeline
***GetControlParam**	Get a specific control parameter form the ISP context

Table 5-9 Pipeline Field Description (continued)

Field	Description
***GetRegisteredControlParam**	Get a specific registered control parameter form the pipeline
***RegisterEnabledModules-AndControls**	Register all enabled modules and controls in the pipeline
***RegisterAllOfModulesAnd-Controls**	Register all the modules and controls in the pipeline
***UnregisterDisabledModule-sAndControls**	Unregister all disabled modules and controls from the pipeline
***UnregisterAllOfModules-AndControls**	Unregister all the modules and controls from the pipeline
***IsModuleEnable**	Return a specific module enable status from the pipeline
***ModuleEnable**	Enable or disable a specific module from the pipeline
***IsModuleUpdate**	Return a specific module update status from the pipeline
***ModuleUpdate**	Enable or disable the update function of specific module from the pipeline
***IsControlEnable**	Return a specific control enable status from the pipeline
***ControlEnable**	Enable or disable a specific control from the pipeline
***ClearModulesAndControls**	Remove and destroy all the registered modules and controls in the pipeline
***PrintRegisteredModules-AndControls**	Print all modules and controls name that registered in the pipeline
***InitAll**	Initialize all modules and controls form the ISP context
***InitAllGlobals**	Initialize all global modules form the ISP context
***InitAllModules**	Initialize all modules form the ISP context
***InitAllControls**	Initialize all controls form the ISP context
***InitPipelineModulesAnd-Controls**	Initialize all modules and controls registered in the pipeline
***SetNextRdma**	Initialize all modules' next Rdma chain form the pipeline
***UpdateAll**	Update all modules in the ISP context including global
***UpdateAllGlobals**	Update only the global modules in the ISP context
***UpdateAllModules**	Update all modules but the global ones in the ISP context
***CalculateAllControls**	Calculate all controls in the ISP context.
***UpdateModule**	Update a specific module Param to Rdma in the ISP context
***CalculateControl**	Calculate a specific control Param to module Param in the ISP context
***UpdateAllRequested**	Update all modules and controls' Param to Rdma with the updated flag activated from the pipeline
***UpdateAllRequestedMod-ules**	Update all modules' Param to Rdma with the updated flag activated from the pipeline
***CalculateAllRequestedCon-trols**	Calculate all controls' Param to module Param with the enabled flag activated from the pipeline

Table 5-9 Pipeline Field Description (continued)

Field	Description
***SetupAll**	Setup all modules in the ISP context
***SetupAllGlobals**	Setup only the global modules in the ISP context
***SetupAllModules**	Setup all modules but the global ones in the ISP context
***SetupModule**	Setup a specific module in the ISP context
***SetupRequested**	Setup all modules with the updated flag activated from the pipeline
***ModuleGetIqParam**	Get a specific module parameter from the pipeline
***ModuleSetIqParam**	Set a specific module parameter from the pipeline
***ControlGetIqParam**	Get a specific control parameter from the pipeline
***ControlSetIqParam**	Set a specific control parameter from the pipeline
***LoadBinParam**	Process binary parameters file
***LoadBinClbrt**	Loop each module to load binary calibration file if needed
***VerifyConfiguration**[2]	Verify if the configuration is correct for the current HW version. No longer in use
***UpdateCiPipeline**[1]	Update CI_PIPELINE information. Not used in the V4L2 driver
***ProgramPipeline**	Program current configuration in the pipeline, and change the pipeline status to setup status
***AddShots**[1]	Add several Shots to the Pipeline. Not used in the V4L2 driver
***DeleteShots**[1]	Delete all allocated Shots of the Pipeline - buffers are unaffected. Not used in the V4L2 driver
***AcquireShot**[1]	Retrieve a captured shot from the pipeline. The Shot must be released later. Not used in the V4L2 driver
***ProgramSpecifiedShot**[1]	This function programs shot with provided buffer id. Not used in the V4L2 driver
***ProgramShot**[1]	Program a shot to be captured in the pipeline (takes 1st available Shot and 1st available Buffers). Not used in the V4L2 driver
***ProcessShot**[1]	Configure a captured shot from the pipeline. Not used in the V4L2 driver
***ReleaseShot**[1]	Release a previously captured and retrieved shot. Not used in the V4L2 driver
***AllocateBuffer**[1]	Allocate 1 buffer for the selected output (pipeline must be preconfigured). Not used in the V4L2 driver
***ImportBuffer**[1]	This function imports a buffer. Not used in the V4L2 driver
***DeregisterBuffer**[1]	Unregister an allocated or imported buffer (The pipeline must be stopped). Not used in the V4L2 driver
***GetUoDimensions**[1]	Get the UO output maximum dimensions. Not used in the V4L2 driver
***SetUoDimensions**[1]	Change the UO output maximum dimensions. Not used in the V4L2 driver
***GetSs0Dimensions**[1]	Get the SS0 output maximum dimensions. Not used in the V4L2 driver
***SetSs0Dimensions**[1]	Change the SS0 output maximum dimensions. Not used in the V4L2 driver
***GetSs1Dimensions**[1]	Get the SS1 output maximum dimensions. Not used in the V4L2 driver
***SetSs1Dimensions**[1]	Change the SS1 output maximum dimensions. Not used in the V4L2 driver
***GetDumpDimensions**[1]	Get the DUMP output maximum dimensions. Not used in the V4L2 driver

Table 5-9 Pipeline Field Description (continued)

Field	Description
***SetDumpDimensions**[1]	Change the DUMP output maximum dimensions. Not used in the V4L2 driver
***GetTiling_1_ReadDimensions**[1]	Get the TIL_1_RD output maximum dimensions. Not used in the V4L2 driver
***SetTiling_1_ReadDimensions**[1]	Change the TIL_1_RD output maximum dimensions. Not used in the V4L2 driver
***GetTiling_1_WriteDimensions**[1]	Get the TIL_1_WR output maximum dimensions. Not used in the V4L2 driver
***SetTiling_1_WriteDimensions**[1]	Change the TIL_1_WR output maximum dimensions. Not used in the V4L2 driver
***StartCapture**	Reserve the hardware for capturing
***StopCapture**	Stop the capture process releasing the HW
***GetFirstAvailableBuffers**[1]	Get the identifier of the 1st available buffers to run with the current configuration. Not used in the V4L2 driver
***Register**	Register the pipeline configuration into kernel driver, and change the pipeline status to setup status
***SetActiveCapture**	Set the active capture to kernel driver
***ClearActiveCapture**	Clear the active capture to kernel driver
***ResetFrameCount**[1]	Reset the ISP total frame count. Not used in the V4L2 driver
***GetFrameCount**[1]	Get the ISP total frame count. Not used in the V4L2 driver
***SetPendingCount**[1]	Set the ISP pending count. Not used in the V4L2 driver
***IncPendingCount**[1]	Increase the ISP pending count. Not used in the V4L2 driver
***GetCompletedCount**[1]	Get the ISP completed count. Not used in the V4L2 driver
***SetCompletedCount**[1]	Set the ISP completed count. Not used in the V4L2 driver
***GetPendingCompletedCount**[1]	Get the ISP pending and completed count. Not used in the V4L2 driver

Note:

1. The field items are for proprietary driver only.
2. The field items are no longer in use.

5.2. API

The following topics show the application interfaces (API) of ISPC Library - Pipeline.

5.2.1. STFLIB_ISP_GetVideoDevice

The API is described in the following sections.

Description

Get a specific video device handle from the ISP context.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_GetVideoDevice(
    STF_VOID *pIspCtx,
    EN_ISP_PORT_ID enIspPortId
    );
```

Parameter

The API has the following parameters.

Table 5-10 STFLIB_ISP_GetVideoDevice Parameter Description

Parameter	Description	Input/Output
pIspCtx	ISP context handle	Input
enIspPortId	Video device ID	Input

Return

The API has the following return values.

Table 5-11 STFLIB_ISP_GetVideoDevice Return Description

Return	Description
STF_S8	Return positive identifier or negative number if sensor cannot be found

Requirement

The file `libISPC.a` is required for using this API.

5.2.2. STFLIB_ISP_GetVideoDevice2

The API is described in the following sections.

Description

Get a specific video device handle from the ISP context.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_GetVideoDevice2(
    STF_VOID *pIspCtx,
    CI_BUFFTYPE enBufferType
    );
```

Parameter

The API has the following parameters.

Table 5-12 STFLIB_ISP_GetVideoDevice2 Parameter Description

Parameter	Description	Input/Output
pIspCtx	ISP context handle	Input

Table 5-12 STFLIB_ISP_GetVideoDevice2 Parameter Description (continued)

Parameter	Description	Input/Output
enBufferType	Video device buffer type (CI_TYPE_UO … CI_TYPE_Y_HIST)	Input

Return

The API has the following return values.

Table 5-13 STFLIB_ISP_GetVideoDevice2 Return Description

Return	Description
STF_VOID *	Video device handle

Requirement

The file `libISPC.a` is required for using this API.

5.2.3. STFLIB_ISP_GetModule

The API is described in the following sections.

Description

Get a specific module handle form the ISP context.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_GetModule(
    STF_VOID *pIspCtx,
    EN_MODULE_ID enModuleId
    );
```

Parameter

The API has the following parameters.

Table 5-14 STFLIB_ISP_GetModule Parameter Description

Parameter	Description	Input/Output
pIspCtx	ISP context handle	Input
enModuleId	Module ID	Input

Return

The API has the following return values.

Table 5-15 STFLIB_ISP_GetModule Return Description

Return	Description
STF_VOID *	Module handle

Requirement

The file `libISPC.a` is required for using this API.

5.2.4. STFLIB_ISP_GetModuleName

The API is described in the following sections.

Description

Get a specific module name form the ISP context.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_GetModuleName(
    STF_VOID *pIspCtx,
    EN_MODULE_ID enModuleId
    );
```

Parameter

The API has the following parameters.

Table 5-16 STFLIB_ISP_GetModuleName Parameter Description

Parameter	Description	Input/Output
pIspCtx	ISP context handle	Input
enModuleId	Module ID	Input

Return

The API has the following return values.

Table 5-17 STFLIB_ISP_GetModuleName Return Description

Return	Description
STF_VOID *	Module handle

Requirement

The file `libISPC.a` is required for using this API.

5.2.5. STFLIB_ISP_GetModuleRdmaBuf

The API is described in the following sections.

Description

Get a specific module Rdma buffer form the ISP context.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_GetModuleRdmaBuf(
    STF_VOID *pIspCtx,
    EN_MODULE_ID enModuleId
    );
```

Parameter

The API has the following parameters.

Table 5-18 STFLIB_ISP_GetModuleRdmaBuf Parameter Description

Parameter	Description	Input/Output
pIspCtx	ISP context handle	Input
enModuleId	Module ID	Input

Return

The API has the following return values.

Table 5-19 STFLIB_ISP_GetModuleRdmaBuf Return Description

Return	Description
STF_VOID *	Rdma buffer pointer

Requirement

The file `libISPC.a` is required for using this API.

5.2.6. STFLIB_ISP_GetModuleIspRdma

The API is described in the following sections.

Description

Get a specific module IspRdma form the ISP context.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_GetModuleIspRdma(
    STF_VOID *pIspCtx,
    EN_MODULE_ID enModuleId
    );
```

Parameter

The API has the following parameters.

Table 5-20 STFLIB_ISP_GetModuleIspRdma Parameter Description

Parameter	Description	Input/Output
pIspCtx	ISP context handle	Input
enModuleId	Module ID	Input

Return

The API has the following return values.

Table 5-21 STFLIB_ISP_GetModuleIspRdma Return Description

Return	Description
STF_VOID *	IspRdma handle

Requirement

The file `libISPC.a` is required for using this API.

5.2.7. STFLIB_ISP_GetModuleRdma

The API is described in the following sections.

Description

Get a specific module Rdma form the ISP context.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_GetModuleRdma(
    STF_VOID *pIspCtx,
    EN_MODULE_ID enModuleId
    );
```

Parameter

The API has the following parameters.

Table 5-22 STFLIB_ISP_GetModuleRdma Parameter Description

Parameter	Description	Input/Output
pIspCtx	ISP context handle	Input
enModuleId	Module ID	Input

Return

The API has the following return values.

Table 5-23 STFLIB_ISP_GetModuleRdma Return Description

Return	Description
STF_VOID *	Rdma handle

Requirement

The file `libISPC.a` is required for using this API.

5.2.8. STFLIB_ISP_GetModuleParam

The API is described in the following sections.

Description

Get a specific module parameter from the ISP context.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_GetModuleParam(
    STF_VOID *pIspCtx,
    EN_MODULE_ID enModuleId
    );
```

Parameter

The API has the following parameters.

Table 5-24 STFLIB_ISP_GetModuleParam Parameter Description

Parameter	Description	Input/Output
pIspCtx	ISP context handle	Input
enModuleId	Module ID	Input

Return

The API has the following return values.

Table 5-25 STFLIB_ISP_GetModuleParam Return Description

Return	Description
STF_VOID *	Parameters handle

Requirement

The file `libISPC.a` is required for using this API.

5.2.9. STFLIB_ISP_GetControl

The API is described in the following sections.

Description

Get a specific control handle from the ISP context.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_GetControl(
    STF_VOID *pIspCtx,
    EN_CONTROL_ID enControlId
    );
```

Parameter

The API has the following parameters.

Table 5-26 STFLIB_ISP_GetControl Parameter Description

Parameter	Description	Input/Output
pIspCtx	ISP context handle	Input
enControlId	Control ID	Input

Return

The API has the following return values.

Table 5-27 STFLIB_ISP_GetControl Return Description

Return	Description
STF_VOID *	Control handle

Requirement

The file `libISPC.a` is required for using this API.

5.2.10. STFLIB_ISP_GetControlName

The API is described in the following sections.

Description

Get a specific control name form the ISP context.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_GetControlName(
    STF_VOID *pIspCtx,
    EN_CONTROL_ID enControlId
    );
```

Parameter

The API has the following parameters.

Table 5-28 STFLIB_ISP_GetControlName Parameter Description

Parameter	Description	Input/Output
pIspCtx	ISP context handle	Input
enControlId	Control ID	Input

Return

The API has the following return values.

Table 5-29 STFLIB_ISP_GetControlName Return Description

Return	Description
STF_VOID *	Control name

Requirement

The file `libISPC.a` is required for using this API.

5.2.11. STFLIB_ISP_GetControlParam

The API is described in the following sections.

Description

Get a specific control parameter from the ISP context.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_GetControlParam(
    STF_VOID *pIspCtx,
    EN_CONTROL_ID enControlId
    );
```

Parameter

The API has the following parameters.

Table 5-30 STFLIB_ISP_GetControlParam Parameter Description

Parameter	Description	Input/Output
pIspCtx	ISP context handle	Input
enControlId	Control ID	Input

Return

The API has the following return values.

Table 5-31 STFLIB_ISP_GetControlParam Return Description

Return	Description
STF_VOID *	Parameters handle

Requirement

The file `libISPC.a` is required for using this API.

5.2.12. STFLIB_ISP_IqTuningDebugInfoEnable

The API is described in the following sections.

Description

Enable or disable debug information print for the IQ tuning command.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_IqTuningDebugInfoEnable(
    STF_VOID *pIspCtx,
    STF_BOOL8 bEnable
    );
```

Parameter

The API has the following parameters.

Table 5-32 STFLIB_ISP_IqTuningDebugInfoEnable Parameter Description

Parameter	Description	Input/Output
pIspCtx	ISP context handle	Input
bEnable	Enable/Disable	Input

Return

The API has the following return values.

Table 5-33 STFLIB_ISP_IqTuningDebugInfoEnable Return Description

Return	Description
0	Success
Other values	Failure

Requirement

The file `libISPC.a` is required for using this API.

5.2.13. STFLIB_ISP_IsIqTuningDebugInfoEnable

The API is described in the following sections.

Description

Return the debug information print status of IQ tuning.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_IsIqTuningDebugInfoEnable(
    STF_VOID *pIspCtx
    );
```

Parameter

The API has the following parameters.

Table 5-34 STFLIB_ISP_IqTuningDebugInfoEnable Parameter Description

Parameter	Description	Input/Output
pIspCtx	ISP context handle	Input

Return

The API has the following return values.

Table 5-35 STFLIB_ISP_IqTuningDebugInfoEnable Return Description

Return	Description
1	Enable
0	Disable

Requirement

The file `libISPC.a` is required for using this API.

5.2.14. STFLIB_ISP_PIPELINE_StructInitialize

The API is described in the following sections.

Description

Pipeline structure initialization function.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_PIPELINE_StructInitialize (
    ST_PIPELINE *pstPipeline,
    STF_U8 u8IspIdx,
    CI_CONNECTION *pCIConnection,
    ST_SENSOR *pstSensor,
    EN_MEM_TYPE enMemType,
```

```
    ST_SIZE stCaptureSize,
    STF_BOOL8 bCheckSensorResolution
    );
```

Parameter

The API has the following parameters.

Table 5-36 STFLIB_ISP_PIPELINE_StructInitialize Parameter Description

Parameter	Description	Input/Output
pstPipeline	Pipeline handle	Input
u8IspIdx	Indicated which ISP is used	Input
pCIConnection	Connection handle	Input
pstSensor	Sensor handle	Input
enMemType	Video buffer type	Input
stCaptureSize	Capture size	Input
bCheckSensorResolution	Check sensor output size	Input

Return

The API has the following return values.

Table 5-37 STFLIB_ISP_PIPELINE_StructInitialize Return Description

Return	Description
0	Success
Other values	Failure

Requirement

The file `libISPC.a` is required for using this API.

5.2.15. STFLIB_ISP_PIPELINE_StructUninitialize

The API is described in the following sections.

Description

Free all of allocated resource from the pipeline.

Syntax

The following code block shows the syntax of the API.

```
STFLIB_ISP_PIPELINE_StructUninitialize(
    ST_PIPELINE *pstPipeline
    );
```

Parameter

The API has the following parameters.

Table 5-38 TFLIB_ISP_PIPELINE_StructUninitialize Parameter Description

Parameter	Description	Input/Output
pstPipeline	Pipeline handle	Input

Return

The API has the following return values.

Table 5-39 TFLIB_ISP_PIPELINE_StructUninitialize Return Description

Return	Description
0	Success
Other values	Failure

Requirement

The file `libCI_User.a` is required for using this API.

Index