Sensor Debugging Guide

Issue 07

Date 2019-08-02

HiSilicon (Shanghai) Technologies Co., Ltd.

Address: New R&D Center, 49 Wuhe Road, Bantian,
Longgang District,
Shenzhen 518129 P. R. China

Website: http://www.hisilicon.com/en/

Email: support@hisilicon.com

About This Document

Purpose

This document is intended for programmers who need to connect different sensors. It provides references for the interconnection procedure and precautions in the sensor interconnection process. This guide includes the development procedure of the driver which is connected to a new sensor and the adaptation of the new sensor in the software development kit (SDK).

Related Versions

The following table lists the product versions related to this document.

Product Name	Version
Hi3516C	V300
Hi3516E	V100
Hi3519	V101
Hi3516A	V200
Hi3559A	V100
Hi3559C	V100
Hi3519A	V100
Hi3516C	V500
Hi3516D	V300
Hi3516A	V300
Hi3516E	V200
Hi3516E	V300
Hi3518E	V300
Hi3516D	V200

Intended Audience

This document is intended for:

- Technical support engineers
- Software development engineers

Change History

Changes between document issues are cumulative. The latest document issue contains all changes made in previous issues.

Issue 07 (2019-08-02)

This issue is the seventh official release, which incorporates the following changes:

In section 1.3.3, the precautions descrption is added.

Issue 06 (2019-06-20)

This issue is the sixth official release, which incorporates the following changes:

Section 1.3.2 is modified.

Issue 05 (2019-04-28)

This issue is the fifth official release, which incorporates the following changes:

Section 2.1 is modified.

Chapter 2 is added.

Issue 04 (2019-02-28)

This issue is the fourth official release, which incorporates the following changes:

Sections 1.3.3 and 1.4 are modified.

Issue 03 (2018-07-10)

This issue is the third official release, which incorporates the following changes:

The content related to the Hi3519A V100 is added.

Issue 02 (2017-12-20)

This issue is the second official release, which incorporates the following changes:

Section 1.4 is modified.

Issue 01 (2017-08-24)

This issue is the first official release.

Contents

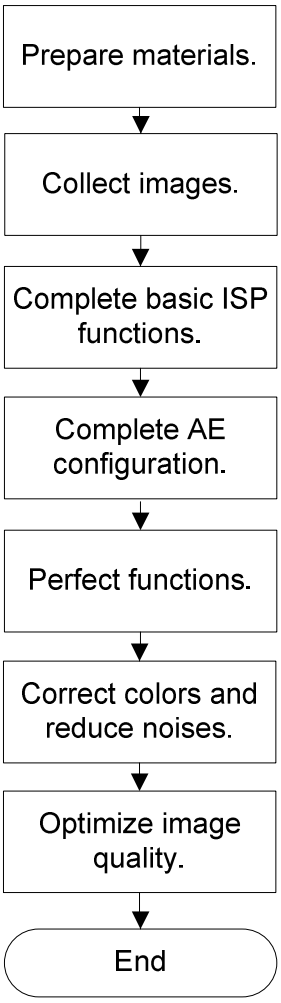
Figures

1 Sensor Debugging

1.1 Debugging Process

You can debug the sensor according to Figure 1-1.

Figure 1-1 Sensor debugging process



1.2 Material Preparation

1.2.1 Confirming Main Chip Specifications

You need to check whether the main chip supports the master mode, linear or WDR interface mode, and confirm the maximum input frequency.

1.2.2 Sensor Data Sheet

- Confirm the interface mode for image transmission and the output frequency.
- Confirm the exposure time and gain setting method, and frame rate modification method.
- Confirm the preceding two items in WDR mode.
- Confirm the synchronization code for the LVDS interface.

1.2.3 Initialization Settings

You need to obtain information about the sensor initialization settings. Generally, you need to prepare at least two sequences, maximum resolution and standard resolution.

1.3 Image Collection

1.3.1 Preparing Hardware

You need to first check whether the sensor register can be written or read.

Test the read and write functions of the sensor register using **i2c_read/i2c_write** or **ssp_read/ssp_write** command.

The commands are integrated in the default file system and can be directly called.

1.3.2 Completing the Initialization Sequence Configuration

For configuring the initialization sequence, you are advised to see the sensor driver in the SDK for rapid development. To facilitate the debugging, you need to eliminate the interference from the AE configuration and frame rate configuration.

Step 1 Prepare the sensor driver.

- You can modify the driver of a sensor with similar specifications (master/slave, i2c/spi, wdr/linear) and try to compile a sensor library. For details, see ***xxx*_cmos.c** and ***xxx*_sensor_ctl.c** files in the **isp/sensor/hi35*xx*/*xxxx*** directory.
- Modify the cmos_set_image_mode function and change the values of **u32MaxWidth** and **u32MaxHeight** in **cmos_get_isp_default** to make sure that the sensor resolution and frame rate are set correctly.
- In **sys_config.c**, modify the registers such as the sensor clock configuration, I^2C/SPI pin multiplexing, VI clock, and ISP clock registers. Perform adaption based on the sensor of similar specifications. For a slave sensor, the slave pin multiplexing needs to be configured. By default, slave pin multiplexing is enabled for Hi3559A V100 and Hi3519A V100. For Hi3516C V500, the slave pin multiplexing is set by calling vi_slave_mode_mux. (By default, this function is not called. Therefore, a judgment branch needs to be added for the slave sensor to implement this function.)

Step 2 Initialize the sensor sequence.

- Implement the void sensor_init() function. For details, see the sensor data sheet or the sensor sequence provided by the sensor vendor. For the slave sensor, HI_MPI_ISP_GetSnsSlaveAttr needs to be called in the sensor_init() function to adapt to the slave mode register. The following takes sensor_init for IMX334 in slave mode as an example:

```
void imx334slave_init(VI_PIPE ViPipe)
{
    HI_U8          u8ImgMode;
    HI_BOOL        bInit;
    HI_S32         SlaveDev;
    HI_U32         u32Data;

    bInit      = g_pastImx334Slave[ViPipe]->bInit;
    u8ImgMode   = g_pastImx334Slave[ViPipe]->u8ImgMode;
    SlaveDev    = g_Imx334SlaveBindDev[ViPipe];
    u32Data     = g_Imx334SalveSensorModeTime[ViPipe];
    /* hold sync signal as fixed * /
    CHECK_RET(HI_MPI_ISP_GetSnsSlaveAttr(SlaveDev,
&gstImx334Sync[ViPipe]));
    gstImx334Sync[ViPipe].unCfg.stBits.bitHEnable = 0;
    gstImx334Sync[ViPipe].unCfg.stBits.bitVEnable = 0;
    gstImx334Sync[ViPipe].u32SlaveModeTime = u32Data;
    CHECK_RET(HI_MPI_ISP_SetSnsSlaveAttr(SlaveDev,
&gstImx334Sync[ViPipe]));

    /* 1. sensor i2c init * /
    imx334slave_i2c_init(ViPipe);

    CHECK_RET(HI_MPI_ISP_GetSnsSlaveAttr(SlaveDev,
&gstImx334Sync[ViPipe]));

    //release hv sync
    gstImx334Sync[ViPipe].u32HsTime =
g_astImx334ModeTbl[u8ImgMode].u32InckPerHs;
    if
(g_pastImx334Slave[ViPipe]->astRegsInfo[0].stSlvSync.u32SlaveVsTime
== 0) {
        gstImx334Sync[ViPipe].u32VsTime =
g_astImx334ModeTbl[u8ImgMode].u32InckPerVs;
    } else {
        gstImx334Sync[ViPipe].u32VsTime =
g_pastImx334Slave[ViPipe]->astRegsInfo[0].stSlvSync.u32SlaveVsTime;
    }
```



```
    gstImx334Sync[ViPipe].unCfg.u32Bytes = 0xc0030000;
    gstImx334Sync[ViPipe].u32HsCyc = 0x3;
    gstImx334Sync[ViPipe].u32VsCyc = 0x3;
    CHECK_RET(HI_MPI_ISP_SetSnsSlaveAttr(SlaveDev,
&gstImx334Sync[ViPipe]));

    /* sensor registers init* /
    ……
    g_pastImx334Slave[ViPipe]->bInit = HI_TRUE;
    return;
}
```

- In the **xxx_sensor_ctl.c**, set the base address of the sensor register to **sensor_i2c_addr**. The address bit width is **sensor_addr_byte**, and the bit width information of the sensor is **sensor_data_byte**.
- In the **xxx_cmos.c** file, comment out all sensor_write_register. In the cmos_get_sns_regs_info function, set **u32RegNum** to **0**. In this way, the AE does not configure the sensor, and therefore the interference is eliminated.

----End

1.3.3 Sensor Output

This section describes the entire channel output based on the **sample** file in the **mpp** directory. The prerequisite is that the sensor sequence is complete. The operations mainly include the configurations of MIPI, VI, ISP, and VPSS. To configure these modules, make simple modifications based on the existing sensor configurations. If the integrated environment is ready, directly configure parameters to start running. Take the startup script of the HiSilicon PQ Tool as an example, the startup configuration file exists in the corresponding sensor directory, and you only need to configure parameters correctly.

Step 1 Compile the sensor to generate a new sensor library in the **ISP** directory after the initialization configuration is completed. The paths of the new library are **mpp/lib/libsns_xxx.a** and **mpp/lib/libsns_xxx.so**.

Step 2 Verify the new sensor based on the **sample** file in the **mpp** directory. In the **sample/Makefile.param** file, add a SENSOR_TYPE for sensor compilation configuration, and then add a corresponding **libsns_xxx.a** file.

Step 3 Add the sensor type to the SAMPLE_VI_MODE_E in **sample_comm.h**. Note that the sensor type must be consistent with the newly added SENSOR_TYPE in the **sample/Makefile.param** file. Then add the attributes such as Bayer pattern, frame rate, and width and height information of the sensor type to the SAMPLE_COMM_ISP_Init function in **sample_comm_isp.c**.

Step 4 Configure MIPI attributes. Add the MIPI attributes to SAMPLE_COMM_VI_SetMipiAttr in **sample_comm_vi.c**. For details about debugging MIPI/LVDS, see *MIPI User Guide*.

Step 5 Configure VI attributes. Add VI attributes to SAMPLE_COMM_VI_StartDev in **sample_comm_vi.c**.

Step 6 Compile and run the corresponding application **sample_vio**. If everything goes smoothly, the entire system is running. You can run the **cat /proc/umap/isp** or **cat /proc/umap/mipi_rx** command to view information.

Step 7 If the ISP is not interrupted, check whether the sensor input clock, output signals, and sensor register configurations are normal. For operation details, see Hi35xx *Professional HD IP Camera SoC Data Sheet or Hi35xxVxxx ultra-HD Mobile Camera SoC Data Sheet.*

Step 8 If the MIPI, VI, and ISP are normal and image quality adjustment is required, transplant the preceding configurations to the sensor configuration file corresponding to the PQTool (create a new **sensor** directory in the **config** directory and make corresponding modifications based on similar sensor configurations), and then view video on demand.

----End

Precautions

When multiple slave sensors are used, some of them may require highly precise timing matching between the Vsync and Hsync signals.

- When the sync mode is enabled at the VI end, the sensor boot process needs to be specially handled. Take SONY IMX477 working in slave mode as an example. The Vsync signal is generated by the VI end, while the Hsync signal is generated by the sensor. Therefore, the Vsync and Hsync timings must strictly match each other.
- When the sync mode is enabled at the VI end, the timing of the Vsync signal is changed so that the Vsync signals in multiple channels can be synchronized. In this case, the timing of the Vsync signal is different from that of the Hsync signal, leading to abnormal IMX477 data output. Therefore, for the IMX477 sensor that requires highly precise timing matching between the Vsync and Hsync signals, the sensor boot process needs to be modified. Before the sync mode is enabled at the VI end, bring the sensor to the standby mode first. After the Vsync signals at the VI end are synchronized, bring the sensor to the data output mode. For details, see the sample of using multiple IMX477 slave sensors.

1.4 ISP Basic Functions

For details about the sensor in this section, see the sensor data sheet, or contact the sensor manufacturer FAE.

For details about the structure, see the *HiISP Development Reference*.

The driver files are classified into ***xxx*_cmos.c**, ***xxx*_cmos_ex.h**, and ***xxx*_sensor_ctl.c** files which are used to implement the ISP functions and initialize the sensor sequence, respectively. The ***xxx*_cmos_ex.h** file stores the global variables of the defined driver file.

The driver files have three callback functions, which are the interfaces used by the sensor drivers to register functions with the firmware. HI_MPI_ISP_SensorRegCallBack(), HI_MPI_AE_SensorRegCallBack(), and HI_MPI_AWB_SensorRegCallBack() correspond to the ISP, HiSilicon AE, and HiSilicon AWB, respectively.

Development Process

The ISP basic functions are implemented in the following order:

1. cmos_set_image_mode(), cmos_set_wdr_mode()

2. sensor_global_init()
3. sensor_init(), sensor_exit()
4. cmos_get_isp_default(), cmos_get_isp_black_level()

Precautions

- cmos_set_image_mode ()

  This function is used to differentiate resolutions and uses **u8ImgMode** in ISP_SNS_STATE_S to transfer the resolution mode.

  Pay attention to the return value. The return value **0** indicates that the sensor needs to be configured again and sensor_init() is called. The return value **–2** indicates that the sensor does not need to be configured again and no operation is performed.

  Pay attention to the differences between **u32FLStd** and **au32FL** in ISP_SNS_STATE_S. **u32FLStd** indicates the total lines at the standard frame rate (generally 30 fps) in the current resolution and WDR modes. **au32FL** indicates the actual total lines. Its value can be changed based on **u32FLStd** and the frame rate caused by frame rate reduction in other functions.

- cmos_set_wdr_mode()

  This function is used to differentiate WDR modes and uses **enWDRMode** in ISP_SNS_STATE_S to transfer the WDR mode.

  In different WDR modes, AE-related functions need to be modified, such as parameters in the ISP default functions and initialization sequence.

- sensor_init()

  You need to configure different sequences according to different resolutions and WDR modes.

- sensor_exit()

  For the implementation of the function, see the drivers of similar sensors.

- cmos_get_isp_default()

  This function is used to debug or correct parameters. You can change parameter values during debugging and correction.

  Note that parameters in different modules such as Gamma and DRC may vary in different WDR modes. For details, see the *HiISP Development Reference*.

- cmos_get_isp_black_level()

  This function is used to configure the black level of the four RAW data channels.

NOTICE

The black level of some sensors deviates with the change of the gain value. In this case, you need to correct the corresponding black level values under different ISO values with the cmos_get_isp_black_level() function.

- sensor_global_init()

  This function is used for sensor initialization configuration, including the resolution, WDR mode, default value of **u32FLStd**, initialization status value, and other status values.

1.5 AE Configuration

After the AE configuration is completed, pictures are normal.

Development Process

The AE is configured in the following order:

1. cmos_get_sns_regs_info()
2. cmos_get_ae_default(), cmos_again_calc_table(), cmos_dgain_calc_table()
3. cmos_get_inttime_max()
4. cmos_gains_update(), cmos_inttime_update()
5. cmos_fps_set(), cmos_slow_framerate_set()

Precautions

- cmos_get_sns_regs_info()

  This function is used to configure the sensors and ISP registers which need to ensure synchronization, such as the exposure time, gains, and total lines. Although these registers can be configured by directly calling sensor_write_register(), the synchronization cannot be guaranteed and flicker may occur. Therefore, these registers must be configured using this function.

  u8DelayFrmNum indicates register configuration delay. For example, gains of many sensors take effect in the next frame, but the exposure time takes effect after the next frame. Therefore, the gains need to be configured after a delay of one frame to make sure that the gain and exposure time take effect simultaneously. In this case, the delay function is needed. **u8Cfg2ValidDelayMax** is configured to control the synchronization between the ISP and sensor. ISP includes parameters such as ISP Dgain and WDR exposure ratio. You can check the parameter correctness by checking the synchronization status between the ISP Dgain and sensor gain. This parameter is used to control the validity time and generally it is one greater than the maximum sensor register delay.

  - **bUpdate** is used to determine whether to update the register. If no modification is required, set it to **false**.

- cmos_get_ae_default()
  - You need to change the parameter values based on the sensor. **enAccuType** indicates the type of the calculation precision, usually AE_ACCURACY_TABLE and AE_ACCURACY_LINEAR. Due to the CPU calculation precision issue, the AE_ACCURACY_DB can be used only when the precision is very low. In other cases, AE_ACCURACY_TABLE is used.
  - The linear mode indicates that the exposure time or gain increases linearly in fixed steps. For example, the exposure time or gain is increased by a multiple of 0.325 each step, or the exposure time is increased by 1 each step. The step is determined by **f32Accuracy**.
  - The table mode applies to gain. That is, the gain each step can reach can be obtained through calculation in cmos_again_calc_table() or cmos_dgain_calc_table() function in a table look-up manner. In this case, **f32Accuracy** is invalid.

  The calculation order of HiSilicon AE is exposure time, Again, Dgain, and ISP Dgain by default. You can adjust the order by setting **AE Route** or **AE RouteEx**.

- cmos_again_calc_table(), cmos_dgain_calc_table()

The input and output of the two functions are the same and the two functions correspond to the table mode of the Again and Dgain, respectively. The following takes Again as an example:

- **pu32AgainLin** is used as the input and output simultaneously. When it is used as the input, it is the expected gain calculated by AE and 1024 indicates one multiple. In this function, you need to find the maximum gain that can be implemented by the sensor and is smaller than **pu32AgainLin**. Re-assign this value to **pu32AgainLin** as the AE output.
- **pu32AgainDb** is used as the output. It is not used for calculation in the AE and just acts as the input of cmos_gains_update(). It is used to transfer the sensor register value of the current gain.

For example, the sensor gain is increased by 0.3 dB. If the sensor register value starts from 0 and is increased by 1 each time, the corresponding gains are 0 dB, 0.3 dB, 0.6 dB, 0.9 dB....

Calculate a look-up table which converts the dB into linear multiple in offline mode, and the corresponding values are 1024, 1060, 1097, 1136….

Compare the input gain with the gain in the look-up table in the function. If the input is 1082, the maximum gain in the table is 1060 and the returned value 1060 is the actual valid gain.

- cmos_get_inttime_max()

  This function takes effect only in xto1 WDR mode, and is used to calculate the maximum exposure time in different exposure ratios.

  The function is needed in row combination mode only. In the row combination mode, the sum of the long exposure time and short exposure time should be less than the length of one frame. Therefore, the maximum exposure time varies under different exposure ratios, and needs to be re-calculated.

- cmos_gains_update(), cmos_inttime_update()

  The two functions are used to configure the sensor register according to the input Again, Dgain, or exposure time. When the table precision mode is used, the input parameters correspond to **pu32AgainDb** and **pu32DgainDb** which are returned in cmos_again_calc_table() or cmos_dgain_calc_table().

  When the linear precision mode is used, the input parameters are the valid gains and exposure time divided by **f32Accuracy**. For example, if **f32Accuracy** is **0.0078125** and the actual valid gain is 1.5 multiple, the input value is 192 (1.5/0.0078125).

  In xto1 WDR mode, you need to configure the exposure time of each long frame and each short frame. cmos_inttime_update() will be called for *X* times and the exposure time of different frames will be input. The exposure time of the short frame will be input first.

- cmos_fps_set(), cmos_slow_framerate_set()

  cmos_fps_set() is the manual configuration function for frame rates. You need to configure the sensor register based on the input frame rate, implement the function of changing sensor frame rate, and return the actual frame rate and the maximum number of exposure lines.

  cmos_slow_framerate_set() is the automatic configuration function for frame rate reduction. You need to configure the sensor register based on the actual required number of exposure lines, implement the function of sensor frame rate reduction, and return the actual number of exposure lines.

1.6 Function Perfection

You need to perfect all other functions and ensure that all functions are normal.

Because synchronization errors easily occur in AE, you need to pay special attention to the verification of synchronization.

1.7 Color Correction and Noise Reduction

You can correct sensor parameters according to the *HiSilicon PQ Tools User Guide*.

1.8 Picture Quality Optimization

You can optimize the picture quality according to the *ISP Tuning Guide*.

2 Sensor Precautions in WDR Modes

2.1 Frame-Combination WDR Mode

2.1.1 Sensor Driver

- The sensor_init function uses the initialization sequence in linear mode.
- For the frame WDR sensor driver, preferentially refer to the existing driver in the SDK, for example, GC2053 in the Hi3518E V300 SDK. Complete the adaption to the cmos_set_wdr_mode, cmos_get_inttime_max, and cmos_inttime_update functions according to the reference.
- Pay attention to the cmos_get_sns_regs_info function. Generally, the exposure time register of the sensor is set to the exposure time for long frames and exposure time for short frames in turn. Therefore, the following requirements must be met for the cmos_get_sns_regs_info function:

  One more group of sensor register configurations are added to set the exposure time for the short frames. The **u32RegAddr** value of the exposure time register in the added group of configurations is the same as that in linear mode. Therefore, u32RegNum = u32RegNum in linear mode + 1. The **u32RegAddr** value in the added sensor register configurations is the same as that in linear mode.

  u8DelayFrmNum of the exposure time for long frames = **u8DelayFrmNum** of the exposure time for short frames + 1

  bUpdate of the exposure time for long frames and exposure time for short frames = **HI_TRUE**

NOTICE

Generally, in the frame WDR mode, you are advised to configure the exposure time for short frame and long frames in sequence to reduce the motion smearing.

2.1.2 Sensor Output

Configure the MIPI, VI, and ISP by referring to section 1.3.3 "Sensor Output." Most configurations in WDR mode are the same as those in linear mode. Note that the WDR modes of the VI and ISP are both set to **WDR_MODE_2To1_FRAME**.

2.2 Built-in WDR Mode

The raw output by the sensor is compressed. Therefore, the Split or Expander module needs to be able to decompress the data. You can set the ISP_CMOS_SPLIT_S or ISP_CMOS_EXPANDER_S structure in the cmos_get_isp_default function. For details, see the *HiISP Development Reference*. Configuration examples are provided as follows (taking sensor OV2718 as an example):

```
static const ISP_CMOS_SPLIT_S g_stCmosSplit = {
    1, /* bEnable * /
    0, /* u8InputWidthSel * /
    3, /* u8ModeIn * /
    0, /* u8ModeOut * /
    0x10, /* u32BitDepthOut * /
    /* ISP_CMOS_SPLIT_POINT_S * /
    {
        {16,  512},
        {24,  1024},
        {80,  8192},
        {128, 32768},
        {129, 32768},
    },
};
```

```
static const ISP_CMOS_EXPANDER_S g_stCmosExpander = {
    1, /* bEnable * /
    12, /* u8BitDepthIn * /
    16, /* u8BitDepthOut * /
    /* ISP_CMOS_EXPANDER_POINT_S * /
    {
        {32,  16384},
        {48,  32768},
        {160, 262144},
        {256, 1048576},
        {257, 1048576},
    },
};
```

