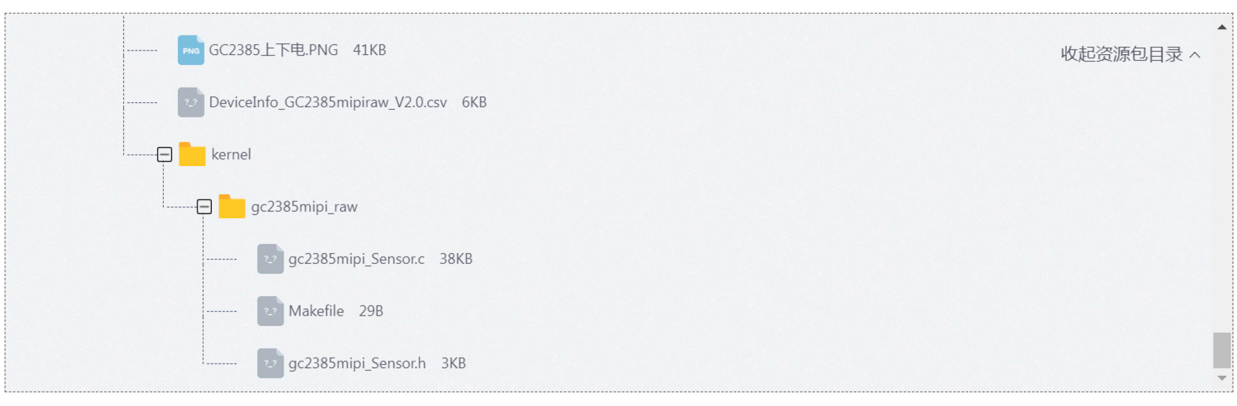


```
/*
 *
 * Filename:
 * ---------
 *     gc2385mipi_Sensor.c
 *
 * Project:
 * --------
 *     ALPS
 *
 * Description:
 * ------------
 *     Source code of Sensor driver
 *
 *
 *------------------------------------------------------------------------------
 * Upper this line, this part is controlled by CC/CQ. DO NOT MODIFY!!
 *============================================================================
 */

#include <linux/videodev2.h>
#include <linux/i2c.h>
#include <linux/platform_device.h>
#include <linux/delay.h>
#include <linux/cdev.h>
#include <linux/uaccess.h>
#include <linux/fs.h>
#include <linux/atomic.h>
#include <linux/types.h>

#include "kd_camera_typedef.h"
#include "kd_imgsensor.h"
#include "kd_imgsensor_define.h"
#include "kd_imgsensor_errcode.h"

#include "gc2385mipi_Sensor.h"

/************************** Modify Following Strings for Debug **************************/
#define PFX "gc2385_camera_sensor"
#define LOG_1 LOG_INF("GC2385, MIPI 1LANE\n")
/****************************   Modify end    *******************************************/
```

```
#define LOG_INF(format, args...)    pr_debug(PFX "[%s] " format, __func__, ##args)

static DEFINE_SPINLOCK(imgsensor_drv_lock);
kal_bool GC2385DuringTestPattern = KAL_FALSE;

static struct imgsensor_info_struct imgsensor_info = {
        .sensor_id = GC2385MIPI_SENSOR_ID,
        .checksum_value = 0xf7375923,
        .pre = {
                .pclk = 41000000,
                .linelength = 1079,
                .framelength = 1248,
                .startx = 0,
                .starty = 0,
                .grabwindow_width = 1600,
                .grabwindow_height = 1200,
                .mipi_data_lp2hs_settle_dc = 85,
                .max_framerate = 300,
        },
        .cap = {
                .pclk = 41000000,
                .linelength = 1079,
                .framelength = 1248,
                .startx = 0,
                .starty = 0,
                .grabwindow_width = 1600,
                .grabwindow_height = 1200,
                .mipi_data_lp2hs_settle_dc = 85,
                .max_framerate = 300,
        },
                .cap1 = {
                .pclk = 41000000,
                .linelength = 1079,
                .framelength = 1248,
                .startx = 0,
                .starty = 0,
                .grabwindow_width = 1600,
                .grabwindow_height = 1200,
                .mipi_data_lp2hs_settle_dc = 85,
                .max_framerate = 300,
        },
        .normal_video = {
                .pclk = 41000000,
                .linelength = 1079,
                .framelength = 1248,
                .startx = 0,
                .starty = 0,
                .grabwindow_width = 1600,
                .grabwindow_height = 1200,
                .mipi_data_lp2hs_settle_dc = 85,
                .max_framerate = 300,
        },
        .hs_video = {
                .pclk = 41000000,
                .linelength = 1079,
                .framelength = 1248,
                .startx = 0,
                .starty = 0,
                .grabwindow_width = 1600,
                .grabwindow_height = 1200,
                .mipi_data_lp2hs_settle_dc = 85,
                .max_framerate = 300,
        },
        .slim_video = {
                .pclk = 41000000,
                .linelength = 1079,
                .framelength = 1248,
                .startx = 0,
                .starty = 0,
                .grabwindow_width = 1600,
                .grabwindow_height = 1200,
                .mipi_data_lp2hs_settle_dc = 85,
                .max_framerate = 300,
        },
```

```
        .margin = 16,
        .min_shutter = 6,
        .max_frame_length = 0x24cf,
        .ae_shut_delay_frame = 0,
        .ae_sensor_gain_delay_frame = 0,
        .ae_ispGain_delay_frame = 2,
        .ihdr_support = 0,
        .ihdr_le_firstline = 0,
        .sensor_mode_num = 3,

        .cap_delay_frame = 2,
        .pre_delay_frame = 2,
        .video_delay_frame = 2,
        .hs_video_delay_frame = 2,
        .slim_video_delay_frame = 2,

        .isp_driving_current = ISP_DRIVING_6MA,
        .sensor_interface_type = SENSOR_INTERFACE_TYPE_MIPI,
        .mipi_sensor_type = MIPI_OPHY_NCSI2,
        .mipi_settle_delay_mode = MIPI_SETTLEDELAY_AUTO,
        .sensor_output_dataformat = SENSOR_OUTPUT_FORMAT_RAW_B,
        .mclk = 24,
        .mipi_lane_num = SENSOR_MIPI_1_LANE,
        .i2c_addr_table = {0x6e, 0xff},
};

static struct imgsensor_struct imgsensor = {
        .mirror = IMAGE_NORMAL,
        .sensor_mode = IMGSENSOR_MODE_INIT,
        .shutter = 0x3ED,
        .gain = 0x40,
        .dummy_pixel = 0,
        .dummy_line = 0,
        .current_fps = 300,
        .autoflicker_en = KAL_FALSE,
        .test_pattern = KAL_FALSE,
        .current_scenario_id = MSDK_SCENARIO_ID_CAMERA_PREVIEW,
        .ihdr_en = 0,
        .i2c_write_id = 0x6e,
};

/* Sensor output window information */
static SENSOR_WINSIZE_INFO_STRUCT imgsensor_winsize_info[5] = {
        {1600, 1200, 0, 0, 1600, 1200, 1600, 1200, 0000, 0000, 1600, 1200, 0, 0, 1600, 1200}, /* Preview */
        {1600, 1200, 0, 0, 1600, 1200, 1600, 1200, 0000, 0000, 1600, 1200, 0, 0, 1600, 1200}, /* capture */
        {1600, 1200, 0, 0, 1600, 1200, 1600, 1200, 0000, 0000, 1600, 1200, 0, 0, 1600, 1200}, /* video */
        {1600, 1200, 0, 0, 1600, 1200, 1600, 1200, 0000, 0000, 1600, 1200, 0, 0, 1600, 1200}, /* HS video */
        {1600, 1200, 0, 0, 1600, 1200, 1600, 1200, 0000, 0000, 1600, 1200, 0, 0, 1600, 1200}  /* slim video */
};

static kal_uint16 read_cmos_sensor(kal_uint32 addr)
{
        kal_uint16 get_byte = 0;
        char pu_send_cmd[1] = {(char)(addr & 0xff)};

        iReadRegI2C(pu_send_cmd, 1, (u8 *)&get_byte, 1, imgsensor.i2c_write_id);

        return get_byte;
}

static void write_cmos_sensor(kal_uint32 addr, kal_uint32 para)
{
        char pu_send_cmd[2] = {(char)(addr & 0xff), (char)(para & 0xff)};

        iWriteRegI2C(pu_send_cmd, 2, imgsensor.i2c_write_id);
}

static void set_dummy(void)
{
}

static kal_uint32 return_sensor_id(void)
{
        return ((read_cmos_sensor(0xf0) << 8) | read_cmos_sensor(0xf1));
}

static void set_max_framerate(UINT16 framerate, kal_bool min_framelength_en)
{
        /* kal_int16 dummy_line; */
        kal_uint32 frame_length = imgsensor.frame_length;
        /* unsigned long flags; */

        frame_length = imgsensor.pclk / framerate * 10 / imgsensor.line_length;

        spin_lock(&imgsensor_drv_lock);
        imgsensor.frame_length = (frame_length > imgsensor.min_frame_length)
                ? frame_length : imgsensor.min_frame_length;
        imgsensor.dummy_line = imgsensor.frame_length - imgsensor.min_frame_length;
        if (imgsensor.frame_length > imgsensor_info.max_frame_length) {
                imgsensor.frame_length = imgsensor_info.max_frame_length;
                imgsensor.dummy_line = imgsensor.frame_length - imgsensor.min_frame_length;
        }
        if (min_framelength_en)
                imgsensor.min_frame_length = imgsensor.frame_length;
        spin_unlock(&imgsensor_drv_lock);
        set_dummy();
}

static void set_shutter(kal_uint16 shutter)
{
        unsigned long flags;
        /* kal_uint32 frame_length = 0; */
        spin_lock_irqsave(&imgsensor_drv_lock, flags);
        imgsensor.shutter = shutter;
        spin_unlock_irqrestore(&imgsensor_drv_lock, flags);

        /* if shutter bigger than frame_length, should extend frame length first */
        spin_lock(&imgsensor_drv_lock);
        if (shutter > imgsensor.min_frame_length - imgsensor_info.margin)
                imgsensor.frame_length = shutter + imgsensor_info.margin;
        else
                imgsensor.frame_length = imgsensor.min_frame_length;
        if (imgsensor.frame_length > imgsensor_info.max_frame_length)
                imgsensor.frame_length = imgsensor_info.max_frame_length;
        spin_unlock(&imgsensor_drv_lock);
        shutter = (shutter < imgsensor_info.min_shutter) ? imgsensor_info.min_shutter : shutter;
        shutter = (shutter > (imgsensor_info.max_frame_length - imgsensor_info.margin)) ?
                (imgsensor_info.max_frame_length - imgsensor_info.margin) : shutter;

        if (shutter > 16383)
                shutter = 16383;
        if (shutter < 6)
                shutter = 6;

        write_cmos_sensor(0xfe, 0x00);
        write_cmos_sensor(0x03, (shutter >> 8) & 0x3f);
        write_cmos_sensor(0x04, shutter & 0xff);

        LOG_INF("Exit! shutter = %d, framelength = %d\n", shutter, imgsensor.frame_length);
}

/*
*static kal_uint16 gain2reg(const kal_uint16 gain)
*{
*    kal_uint16 reg_gain = 0x0000;
*
*    reg_gain = ((gain / BASEGAIN) << 4) + ((gain % BASEGAIN) * 16 / BASEGAIN);
*    reg_gain = reg_gain & 0xFFFF;
*    return (kal_uint16)reg_gain;
*}
*/

#define ANALOG_GAIN_1 64   /* 1.00x */
#define ANALOG_GAIN_2 92   /* 1.43x */
#define ANALOG_GAIN_3 127  /* 1.99x */
#define ANALOG_GAIN_4 183  /* 2.86x */
#define ANALOG_GAIN_5 257  /* 4.01x */
#define ANALOG_GAIN_6 369  /* 5.76x */
#define ANALOG_GAIN_7 531  /* 8.30x */
#define ANALOG_GAIN_8 750  /* 11.72x */
#define ANALOG_GAIN_9 1092 /* 17.06x */

static kal_uint16 set_gain(kal_uint16 gain)
{
        kal_uint16 iReg, temp;

        iReg = gain;

        if (iReg < 0x40)
                iReg = 0x40;

        if ((iReg >= ANALOG_GAIN_1) && (iReg < ANALOG_GAIN_2)) {
                write_cmos_sensor(0xfe, 0x00);
                write_cmos_sensor(0x20, 0x73);
                write_cmos_sensor(0x22, 0xa2);
                /* analog gain */
                write_cmos_sensor(0xb6, 0x00);
                temp = 256 * iReg / ANALOG_GAIN_1;
                write_cmos_sensor(0xb1, te
```