OV5640 自动对焦摄像模组应用指南

(MIPI 接口)

修改日期: Nov. 4th, 2011

文档版本号: 2.13

内容目录

1. OV5640 应用指南

OV5640 是一款 1/4 英寸 5 百万像素的高性能图像传感器，支持 DVP 和 MIPI 接口。 此文档内容为 MIPI 接口的应用指南。DVP 接口的文档为“OV5640 自动对焦模组应用指南（DVP 接口）”

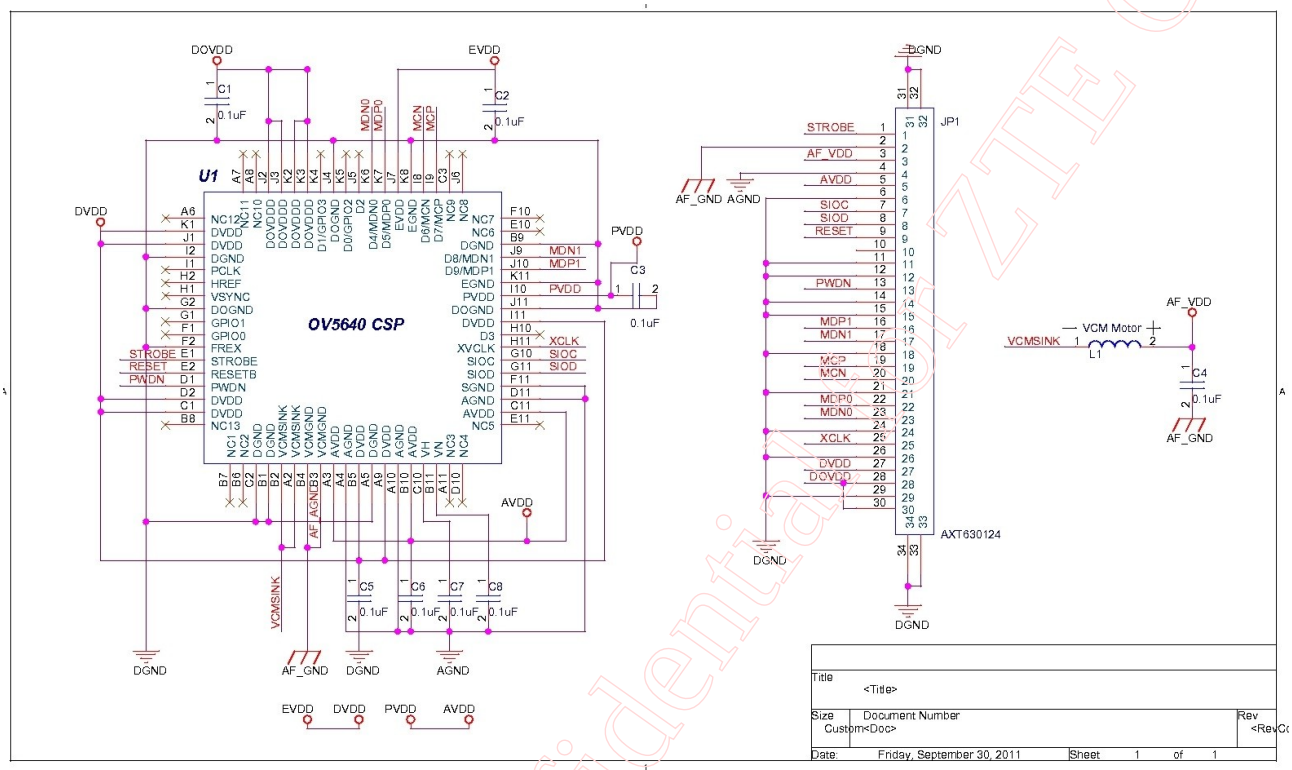
OV5640 传感器可用于：

手机 5 百万像素主摄像头

平板电脑 5 百万像素主摄像头

2. 硬件设计

2.1 OV5640 照相模组原理图参考设计



说明:

1. PWND 引脚，高电平有效。当上拉至和 DOVDD 电压一致的高电平时，使 OV5640 进入节电模式, 不使用时需在模组外部接地。
2. RESETB 引脚，低点平有效。置低时复位 OV5640, 不使用时需在模组外部与 DOVDD 连接。
3. AVDD 为传感器模拟电源引脚，电压范围为 2.6-3.0V (纹波小)。推荐接入 2.8V 电压。在 OTP 写入状态时，AVDD 必须接入 2.5V+-5%，但是 OPT 读取状态无此要求。
4. DVDD 是传感器数字电源引脚，电压 1.5V±5% (纹波小)。强烈建议使用内部 DVDD 稳压器。
5. DOVDD 是传感器数字 IO 电源引脚，电压范围 1.7V-3.0V(clean)，建议为 1.8V。
6. AGND 与 DGND 引脚应在模组内分开，在模组外 PCB 单点连接，不要在模组内连接。
7. 电容器的位置需靠近其相应的 OV5640 引脚。
8. MCP, MCN, MDP0 和 MDP1 为单通道 MIPI 接口引脚。MCP, MCN, MDP0, MDN0, MDP1 和 MCP1 为双通道 MIPI 接口引脚。

2.2 主机接口

2.1.1 引脚定义

OV5640 视频信号接口支持单通道 MIPI 传输双通道 MIPI 传输。MCP, MCN, MDP0 和 MDP1 为单通道 MIPI 接口引脚。MCP, MCN, MDP0, MDN0, MDP1 和 MCP1 为双通道 MIPI 接口引脚。

同步信号 Href 和 Hsync 使用同一引脚 Href。通过 SCCB 设置可选择此引脚为 Href 或 Hsync 信号。

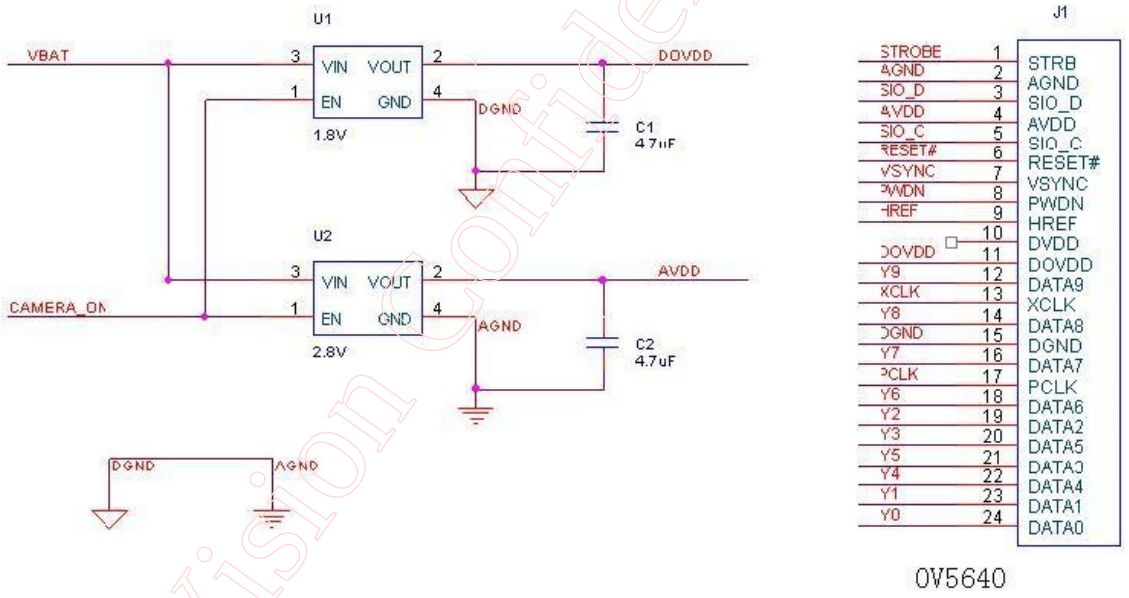
SIO_C 以及 SIO_D 总线需外接上拉电阻, 标准上拉电阻阻值为 5.1K 欧姆。

RESETB 引脚低有效，且内置上拉电阻。RESETB 引脚需由后端芯片控制以得到合适的上电顺序。

PWDN 引脚高有效，且内置下拉电阻 。PWDN 引脚需由后端芯片控制以得到合适的上电顺序。

2.2 供电设计

当 OV5640 与支持 MIPI 接口的后端芯片共用时，DOVDD 引脚电压需为 1.8V。DVDD 引脚的电压由 OV5640 内部的稳压器生成。所以需要 2 组稳压器来为 OV5640 供电。如下图：



2.3 与镜头相关参数

2.3.1 边角失光

边角失光是指图像的周边（四个角）亮度比中心暗，这种现象是由镜头造成的。可开启 OV5640 的镜头均匀性补偿功能来对图片周边的亮度进行补偿，使得整个图片的亮度看上去保持一致。

2.3.2 暗角

有些镜头可能会有暗角存在。暗角表示图像的周边（四个角）亮度很暗，看起来几乎是黑的。这种状况是无法用芯片的镜头均匀性补偿功能来矫正的，所以这样的镜头模组不合格，请不要采用。

2.3.3 分辨率

摄像模组的分辨率取决于镜头的设计以及模组的调焦质量，同时和图像传感器的分辨率也有关系 。其中调焦质量是摄像模组组装时非常重要的指标。

2.3.4 光学对比度

就影响成像质量来讲，镜头的光学对比度也是非常重要的。如果光学对比度较低的话，会使图片看起来好像雾蒙蒙的。虽然可以通过增加传感器的对比度来使得图片显得更加锐利，但是较高的传感器成像对比度会导致图片在较暗的部分丢失细节。

2.3.5 保护玻璃

虽然保护玻璃是镜头里最便宜的部件，但是它依然对镜头的成像质量有很大的影响。保护玻璃必须由光学玻璃制造并且双面镀增透膜（AR coating）。否则会导致灵敏度损失或者严重的光学耀斑（flare）。

2.3.6 Lens 补偿（镜头均匀性补偿）

所有模组都必须做镜头均匀性补偿补偿，请联络 OVT 取得由 OVT FAE 工程师为不同镜头优化过的 Lens 补偿设置的详细参数。

2.3.6.1 Lens 补偿:

2.3.6.2 Lens 补偿:

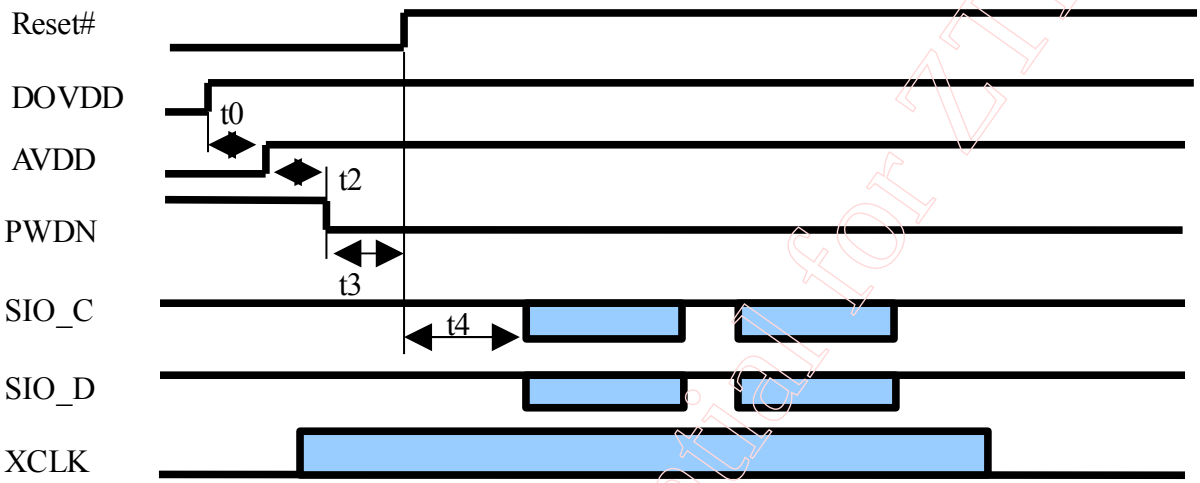
注:

请联络 OVT FAE 工程师取得模组 Lens 设置的参数。

3. 硬件操作

3.1 操作模式

3.1.1 上电（power up）



t0: >= 0 毫秒. 从 DOVDD 稳定到 AVDD 稳定的时间。
t2: >= 5 毫秒 从 AVDD 稳定到传感器上电稳定之间的时间。
t3: >= 1 毫秒. 传感器上电稳定到 ResetB 拉高之间的延迟。
t4: >=20 毫秒. ResetB 拉高到 SCCB 初始化之间的延迟。

步骤 1:
ResetB 拉低，复位 OV5640。 PWDN 引脚拉高。

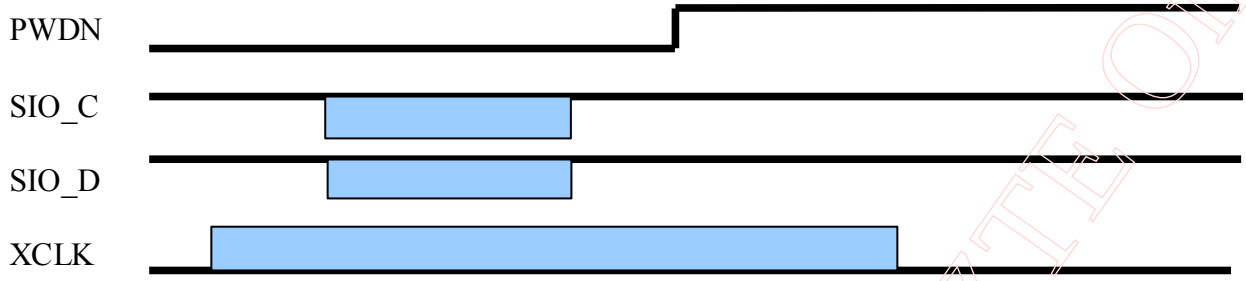
步骤 2:
DOVDD 和 AVDD 上电。这两路最好同时上电，如果不能同时上电，那么必须使 DOVDD 上电在先，AVDD 上电在后。

步骤 3:
等 AVDD 稳定 5 毫秒后, 拉低 PWDN。

步骤 4:
PWDN 置低 1 毫秒后, 拉高 ResetB。

步骤 5:
20 毫秒后, 初始化 OV5640 的 SCCB 寄存器设置。寄存器设置可参考第四章的 OV5640 软件应用指南。

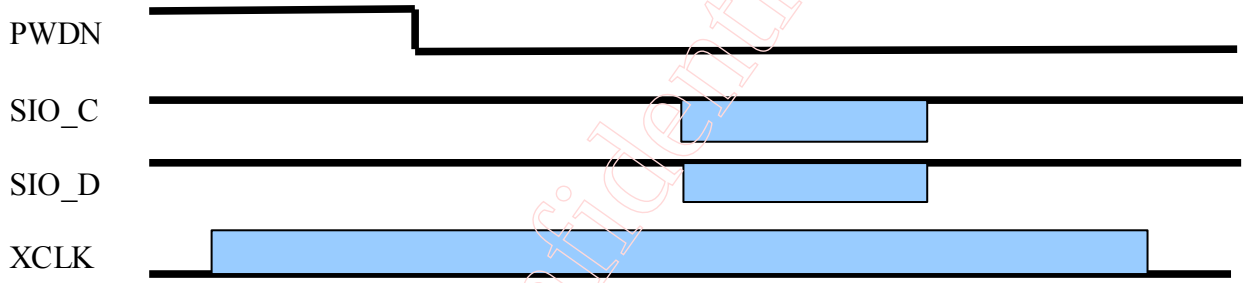
3.1.2 节电模式(Power Down)



步骤 1:
拉高 PWDN 引脚。

步骤 2:
0.1 毫秒后拉低 XCLK 。在 PWDN 拉高以后 XCLK 必须至少保持 0.1 毫秒高电平。

3.1.3 由节电模式唤醒



步骤 1:
加上 XCLK 时钟。

步骤 2:
0.1 毫秒后, 拉低 PWDN。

步骤 3(可选):
初始化 SCCB 寄存器。寄存器参数请见第四章。

3.1.4 断电

步骤 1.
拉低 XCLK。

步骤 2.
切断 AVDD, DVDD 和 DOVDD。这三路电源应同时切断，如无法同时切断，则必须做到 DVDD 为先，AVDD 第二，DOVDD 最后。

步骤 3.

拉低 PWDN 和 RESETB。

3.1.5 硬件复位

OV5640 传感器的 RESETB 引脚在拉低时（接地）可以实现硬件复位。此时 OV5640 清空所有寄存器并使其恢复到初始值。通过 SCCB 接口向寄存器 0x12 的[7]位写“1”，同样可以使传感器复位。

上电时整个芯片都会做复位，但即使这样，也建议在上电后手动进行硬件复位。硬件复位低位有效且基于异步设计，复位的脉冲时间必须等于 1 毫秒或更长。

3.2 操作流程

3.2.1 不使用时断电

模式	操作流程
连接电池	无
打开相机	上电 硬件复位 初始化
退出相机	断电

3.2.2 不使用时处于节电模式（power down）

模式	操作流程
连接电池	上电 硬件复位 节电模式
打开相机	从节电模式唤醒 初始化
退出相机	断电

3.2.3 OV5640 与其他设备共用 I2C 总线

其他设备接入 I^2C 总线	其他设备无法接入 I^2C 总线
断电 进入节电模式 从节电模式唤醒 上电	无

4. 软件指南

4.1 单通道 MIPI 接口

4.1.1 YCbCr 初始化参数

```
//
//OV5640 setting Version History
//dated 04/08/2010 A02
//--Based on v08 release
//
//dated 04/20/2010 A03
//--Based on V10 release
//
//dated 04/22/2010 A04
//--Based on V10 release
//--updated ccr & awb setting
//
//dated 04/22/2010 A06
//--Based on A05 release
//--Add pg setting
//
//dated 05/19/2011 A09
//--changed pchg 3708 setting

write_i2c(0x3103, 0x11); // SCCB system control
write_i2c(0x3008, 0x82); // software reset
// delay 5ms
write_i2c(0x3008, 0x42); // software power down
write_i2c(0x3103, 0x03); // SCCB system control
write_i2c(0x3017, 0x00); // set Frex, Vsync, Href, PCLK, D[9:6] input
write_i2c(0x3018, 0x00); // set d[5:0], GPIO[1:0] input
write_i2c(0x3034, 0x18); // MIPI 8-bit mode
write_i2c(0x3037, 0x13); // PLL
write_i2c(0x3108, 0x01); // system divider
write_i2c(0x3630, 0x36);
write_i2c(0x3631, 0x0e);
write_i2c(0x3632, 0xe2);
write_i2c(0x3633, 0x12);
write_i2c(0x3621, 0xe0);
write_i2c(0x3704, 0xa0);
write_i2c(0x3703, 0x5a);
write_i2c(0x3715, 0x78);
write_i2c(0x3717, 0x01);
write_i2c(0x370b, 0x60);
write_i2c(0x3705, 0x1a);
```

```
write_i2c(0x3905, 0x02);
write_i2c(0x3906, 0x10);
write_i2c(0x3901, 0x0a);
write_i2c(0x3731, 0x12);
write_i2c(0x3600, 0x08); // VCM debug mode
write_i2c(0x3601, 0x33); // VCM debug mode
write_i2c(0x302d, 0x60); // system control
write_i2c(0x3620, 0x52);
write_i2c(0x371b, 0x20);
write_i2c(0x471c, 0x50);
write_i2c(0x3a13, 0x43); // AGC pre-gain, 0x40 = 1x
write_i2c(0x3a18, 0x00); // gain ceiling
write_i2c(0x3a19, 0xf8);  // gain ceiling
write_i2c(0x3635, 0x13);
write_i2c(0x3636, 0x03);
write_i2c(0x3634, 0x40);
write_i2c(0x3622, 0x01);

// 50Hz/60Hz    50/60Hz 灯光条纹过滤
write_i2c(0x3c01, 0x34); // 50/60Hz
write_i2c(0x3c04, 0x28); // threshold for low sum
write_i2c(0x3c05, 0x98); // threshold for high sum
write_i2c(0x3c06, 0x00); // light meter 1 threshold high
write_i2c(0x3c08, 0x00); // light meter 2 threshold high
write_i2c(0x3c09, 0x1c); // light meter 2 threshold low
write_i2c(0x3c0a, 0x9c); // sample number high
write_i2c(0x3c0b, 0x40); // sample number low

// timing    时序
write_i2c(0x3800, 0x00); // HS
write_i2c(0x3801, 0x00); // HS
write_i2c(0x3802, 0x00); // VS
write_i2c(0x3804, 0x0a); // HW
write_i2c(0x3805, 0x3f);  // HW
write_i2c(0x3810, 0x00); // H offset high
write_i2c(0x3811, 0x10); // H offset low
write_i2c(0x3812, 0x00); // V offset high
write_i2c(0x3708, 0x64);
write_i2c(0x3a08, 0x01); // B50
write_i2c(0x4001, 0x02); // BLC start line
write_i2c(0x4005, 0x1a); // BLC always update
write_i2c(0x3000, 0x00); // system reset 0
write_i2c(0x3002, 0x1c); // system reset 2
write_i2c(0x3004, 0xff);  // clock enable 00
write_i2c(0x3006, 0xc3); // clock enable 2
write_i2c(0x300e, 0x25); // MIPI control, 1 lane, MIPI enable
write_i2c(0x302e, 0x08);
write_i2c(0x4300, 0x30); // YUV 422, YUYV
```

```
write_i2c(0x501f, 0x00);  // ISP YUV 422
write_i2c(0x4407, 0x04); // JPEG QS
write_i2c(0x440e, 0x00);
write_i2c(0x5000, 0xa7); // ISP control, Lenc on, gamma on, BPC on, WPC on, CIP on

// AWB    自动白平衡
write_i2c(0x5180, 0xff);
write_i2c(0x5181, 0xf2);
write_i2c(0x5182, 0x00);
write_i2c(0x5183, 0x14);
write_i2c(0x5184, 0x25);
write_i2c(0x5185, 0x24);
write_i2c(0x5186, 0x09);
write_i2c(0x5187, 0x09);
write_i2c(0x5188, 0x09);
write_i2c(0x5189, 0x75);
write_i2c(0x518a, 0x54);
write_i2c(0x518b, 0xe0);
write_i2c(0x518c, 0xb2);
write_i2c(0x518d, 0x42);
write_i2c(0x518e, 0x3d);
write_i2c(0x518f, 0x56);
write_i2c(0x5190, 0x46);
write_i2c(0x5191, 0xf8);
write_i2c(0x5192, 0x04);
write_i2c(0x5193, 0x70);
write_i2c(0x5194, 0xf0);
write_i2c(0x5195, 0xf0);
write_i2c(0x5196, 0x03);
write_i2c(0x5197, 0x01);
write_i2c(0x5198, 0x04);
write_i2c(0x5199, 0x12);
write_i2c(0x519a, 0x04);
write_i2c(0x519b, 0x00);
write_i2c(0x519c, 0x06);
write_i2c(0x519d, 0x82);
write_i2c(0x519e, 0x38);
// color matrix    色彩矩阵
write_i2c(0x5381, 0x1e);
write_i2c(0x5382, 0x5b);
write_i2c(0x5383, 0x08);
write_i2c(0x5384, 0x0a);
write_i2c(0x5385, 0x7e);
write_i2c(0x5386, 0x88);
write_i2c(0x5387, 0x7c);
write_i2c(0x5388, 0x6c);
write_i2c(0x5389, 0x10);
write_i2c(0x538a, 0x01);
```

```
write_i2c(0x538b, 0x98);

// CIP  锐化和降噪
write_i2c(0x5300, 0x08); // sharpen MT th1
write_i2c(0x5301, 0x30); // sharpen MT th2
write_i2c(0x5302, 0x10); // sharpen MT offset 1
write_i2c(0x5303, 0x00); // sharpen MT offset 2
write_i2c(0x5304, 0x08); // DNS threshold 1
write_i2c(0x5305, 0x30); // DNS threshold 2
write_i2c(0x5306, 0x08); // DNS offset 1
write_i2c(0x5307, 0x16); // DNS offset 2
write_i2c(0x5309, 0x08); // sharpen TH th1
write_i2c(0x530a, 0x30); // sharpen TH th2
write_i2c(0x530b, 0x04); // sharpen TH offset 1
write_i2c(0x530c, 0x06); // sharpen Th offset 2

// gamma      伽玛曲线
write_i2c(0x5480, 0x01);
write_i2c(0x5481, 0x08);
write_i2c(0x5482, 0x14);
write_i2c(0x5483, 0x28);
write_i2c(0x5484, 0x51);
write_i2c(0x5485, 0x65);
write_i2c(0x5486, 0x71);
write_i2c(0x5487, 0x7d);
write_i2c(0x5488, 0x87);
write_i2c(0x5489, 0x91);
write_i2c(0x548a, 0x9a);
write_i2c(0x548b, 0xaa);
write_i2c(0x548c, 0xb8);
write_i2c(0x548d, 0xcd);
write_i2c(0x548e, 0xdd);
write_i2c(0x548f, 0xea);
write_i2c(0x5490, 0x1d);

// UV adjust      UV 色彩饱和度调整
write_i2c(0x5580, 0x06); // sat on, contrast on
write_i2c(0x5583, 0x40); // sat U
write_i2c(0x5584, 0x10); // sat V
write_i2c(0x5589, 0x10); // UV adjust th1
write_i2c(0x558a, 0x00); // UV adjust th2[8]
write_i2c(0x558b, 0xf8);  // UV adjust th2[7:0]
write_i2c(0x501d, 0x04); // enable manual offset of contrast

// lens correction     镜头补偿
write_i2c(0x5800, 0x23);
write_i2c(0x5801, 0x14);
```

```
write_i2c(0x5802, 0x0f);
write_i2c(0x5803, 0x0f);
write_i2c(0x5804, 0x12);
write_i2c(0x5805, 0x26);
write_i2c(0x5806, 0x0c);
write_i2c(0x5807, 0x08);
write_i2c(0x5808, 0x05);
write_i2c(0x5809, 0x05);
write_i2c(0x580a, 0x08);
write_i2c(0x580b, 0x0d);
write_i2c(0x580c, 0x08);
write_i2c(0x580d, 0x03);
write_i2c(0x580e, 0x00);
write_i2c(0x580f, 0x00);
write_i2c(0x5810, 0x03);
write_i2c(0x5811, 0x09);
write_i2c(0x5812, 0x07);
write_i2c(0x5813, 0x03);
write_i2c(0x5814, 0x00);
write_i2c(0x5815, 0x01);
write_i2c(0x5816, 0x03);
write_i2c(0x5817, 0x08);
write_i2c(0x5818, 0x0d);
write_i2c(0x5819, 0x08);
write_i2c(0x581a, 0x05);
write_i2c(0x581b, 0x06);
write_i2c(0x581c, 0x08);
write_i2c(0x581d, 0x0e);
write_i2c(0x581e, 0x29);
write_i2c(0x581f, 0x17);
write_i2c(0x5820, 0x11);
write_i2c(0x5821, 0x11);
write_i2c(0x5822, 0x15);
write_i2c(0x5823, 0x28);
write_i2c(0x5824, 0x46);
write_i2c(0x5825, 0x26);
write_i2c(0x5826, 0x08);
write_i2c(0x5827, 0x26);
write_i2c(0x5828, 0x64);
write_i2c(0x5829, 0x26);
write_i2c(0x582a, 0x24);
write_i2c(0x582b, 0x22);
write_i2c(0x582c, 0x24);
write_i2c(0x582d, 0x24);
write_i2c(0x582e, 0x06);
write_i2c(0x582f, 0x22);
write_i2c(0x5830, 0x40);
write_i2c(0x5831, 0x42);
```

```
write_i2c(0x5832, 0x24);
write_i2c(0x5833, 0x26);
write_i2c(0x5834, 0x24);
write_i2c(0x5835, 0x22);
write_i2c(0x5836, 0x22);
write_i2c(0x5837, 0x26);
write_i2c(0x5838, 0x44);
write_i2c(0x5839, 0x24);
write_i2c(0x583a, 0x26);
write_i2c(0x583b, 0x28);
write_i2c(0x583c, 0x42);
write_i2c(0x583d, 0xce);

write_i2c(0x5025, 0x00);
write_i2c(0x3a0f, 0x30);  // stable in high
write_i2c(0x3a10, 0x28); // stable in low
write_i2c(0x3a1b, 0x30); // stable out high
write_i2c(0x3a1e, 0x26);  // stable out low
write_i2c(0x3a11, 0x60);  // fast zone high
write_i2c(0x3a1f, 0x14);  // fast zone low
write_i2c(0x3008, 0x02); // wake up
```

4.1.2 YCbCr VGA 预览 30 帧/秒

```
write_i2c(0x3108, 0x01); // system divider
write_i2c(0x3035, 0x12); // pll
write_i2c(0x3036, 0x38); // pll
write_i2c(0x3c07, 0x08); // light meter 1 threshold
write_i2c(0x3820, 0x41); // ISP flip off, sensor flip off
write_i2c(0x3821, 0x07); // ISP mirror on, sensor mirror on
// timing 时序
write_i2c(0x3814, 0x31); // X inc
write_i2c(0x3815, 0x31); // Y inc
write_i2c(0x3803, 0x04); // VS
write_i2c(0x3806, 0x07); // VH
write_i2c(0x3807, 0x9b); // VH
write_i2c(0x3808, 0x02); // DVPHO
write_i2c(0x3809, 0x80); // DVPHO
write_i2c(0x380a, 0x01); // DVPVO
write_i2c(0x380b, 0xe0); // DVPVO
write_i2c(0x380c, 0x07); // HTS
write_i2c(0x380d, 0x68); // HTS
write_i2c(0x380e, 0x03); // VTS
write_i2c(0x380f, 0xd8);  // VTS
write_i2c(0x3813, 0x06); // V offset

write_i2c(0x3618, 0x00);
```

```
write_i2c(0x3612, 0x29);
write_i2c(0x3709, 0x52);
write_i2c(0x370c, 0x03);

write_i2c(0x3a02, 0x03); // 60Hz max exposure
write_i2c(0x3a03, 0xd8); // 60Hz max exposure
write_i2c(0x3a09, 0x27); // B50 low
write_i2c(0x3a0a, 0x00); // B60 high
write_i2c(0x3a0b, 0xf6); // B60 low
write_i2c(0x3a0e, 0x03); // B50 max
write_i2c(0x3a0d, 0x04); // B60 max
write_i2c(0x3a14, 0x03); // 50Hz max exposure
write_i2c(0x3a15, 0xd8); // 50Hz max exposure

write_i2c(0x4004, 0x02); // BLC line number
write_i2c(0x4713, 0x03); // JPEG mode 3
write_i2c(0x460b, 0x35); // debug
write_i2c(0x460c, 0x22); // VFIFO, PCLK manual
write_i2c(0x4837, 0x44); // MIPI global timing
write_i2c(0x3824, 0x02); // PCLK divider
write_i2c(0x5001, 0xa3); // SDE on, scale on, UV average off, CMX on, AWB on
```

4.1.3 YCbCr 720p 视频 30 帧/秒

```
write_i2c(0x3108, 0x02); // system divider
write_i2c(0x3035, 0x11); // pll
write_i2c(0x3036, 0x54); // pll
write_i2c(0x3c07, 0x07); // light meter 1 threshold
write_i2c(0x3820, 0x41); // ISP flip off, sensor flip off
write_i2c(0x3821, 0x07); // ISP mirror on, sensor mirror on
// timing 时序
write_i2c(0x3814, 0x31); // X inc
write_i2c(0x3815, 0x31); // Y inc
write_i2c(0x3803, 0xfa); // VS
write_i2c(0x3806, 0x06); // VH
write_i2c(0x3807, 0xa9); // VH
write_i2c(0x3808, 0x05); // DVPHO
write_i2c(0x3809, 0x00); // DVPHO
write_i2c(0x380a, 0x02); // DVPVO
write_i2c(0x380b, 0xd0); // DVPVO
write_i2c(0x380c, 0x07); // HTS
write_i2c(0x380d, 0x64); // HTS
write_i2c(0x380e, 0x02); // VTS
write_i2c(0x380f, 0xe4); // VTS
write_i2c(0x3813, 0x04); // V offset

write_i2c(0x3618, 0x00);
```

```
write_i2c(0x3612, 0x29);
write_i2c(0x3709, 0x52);
write_i2c(0x370c, 0x03);
// banding filter  去除灯光条纹
write_i2c(0x3a02, 0x02); // 60Hz max exposure
write_i2c(0x3a03, 0xe4); // 60Hz max exposure
write_i2c(0x3a09, 0xbc); // B50 low
write_i2c(0x3a0a, 0x01); // B60 high
write_i2c(0x3a0b, 0x72); // B60 low
write_i2c(0x3a0e, 0x01); // B50 max
write_i2c(0x3a0d, 0x02); // B60 max
write_i2c(0x3a14, 0x02); // 50Hz max exposure
write_i2c(0x3a15, 0xe4); // 50Hz max exposure

write_i2c(0x4004, 0x02); // BLC line number
write_i2c(0x4713, 0x02); // JPEG mode 2
write_i2c(0x460b, 0x37);
write_i2c(0x460c, 0x20); // VFIFO, PCLK auto
write_i2c(0x4837, 0x16); // MIPI global timing
write_i2c(0x3824, 0x04); // PCLK divider
write_i2c(0x5001, 0x83); // SDE on, scale off, UV average off, CMX on, AWB on
```

4.1.4 YCbCr 5 百万拍照 7.5 帧/秒

```
write_i2c(0x3108, 0x02); // system divider
write_i2c(0x3035, 0x11); // pll
write_i2c(0x3036, 0x54); // pll
write_i2c(0x3c07, 0x07); // light meter 1 threshold
write_i2c(0x3820, 0x40); // ISP flip off, sensor flip off
write_i2c(0x3821, 0x06); // ISP mirror on, sensor mirror on
// timing  时序
write_i2c(0x3814, 0x11); // X inc
write_i2c(0x3815, 0x11); // Y inc
write_i2c(0x3803, 0x00); // VS
write_i2c(0x3806, 0x07); // VH
write_i2c(0x3807, 0x9f); // VH
write_i2c(0x3808, 0x0a); // DVPHO
write_i2c(0x3809, 0x20); // DVPHO
write_i2c(0x380a, 0x07); // DVPVO
write_i2c(0x380b, 0x98); // DVPVO
write_i2c(0x380c, 0x0b); // HTS
write_i2c(0x380d, 0x1c); // HTS
write_i2c(0x380e, 0x07); // VTS
write_i2c(0x380f, 0xb0); // VTS
write_i2c(0x3813, 0x04); // V offset

write_i2c(0x3618, 0x04);
```

```
write_i2c(0x3612, 0x2b);
write_i2c(0x3709, 0x12);
write_i2c(0x370c, 0x00);
// banding filter  去除灯光条纹
write_i2c(0x3a02, 0x07); // 60Hz max exposure
write_i2c(0x3a03, 0xb0); // 60Hz max exposure
write_i2c(0x3a09, 0x27); // B50 low
write_i2c(0x3a0a, 0x00); // B60 high
write_i2c(0x3a0b, 0xf6);  // B60 low
write_i2c(0x3a0e, 0x06); // B50 max
write_i2c(0x3a0d, 0x08); // B60 max
write_i2c(0x3a14, 0x07); // 50Hz max exposure
write_i2c(0x3a15, 0xb0); // 50Hz max exposure

write_i2c(0x4004, 0x06); // BLC line number
write_i2c(0x4713, 0x00); // JPEG mode
write_i2c(0x460b, 0x35);
write_i2c(0x460c, 0x22); // VFIFO, PCLK manual
write_i2c(0x4837, 0x16); // MIPI global timing
write_i2c(0x3824, 0x01); // PCLK divider
write_i2c(0x5001, 0x83); // SDE on, scale off, UV average off, CMX on, AWB on
```

4.2 双通道 MIPI 接口

4.2.1 YCbCr 初始化参数

```
//
//OV5640 setting Version History
//dated 04/08/2010 A02
//--Based on v08 release
//
//dated 04/20/2010 A03
//--Based on V10 release
//
//dated 04/22/2010 A04
//--Based on V10 release
//--updated ccr & awb setting
//
//dated 04/22/2010 A06
//--Based on A05 release
//--Add pg setting
//
//dated 05/19/2011 A09
//--changed pchg 3708 setting

write_i2c(0x3103, 0x11); // SCCB system control
write_i2c(0x3008, 0x82); // software reset
// delay 5ms
write_i2c(0x3008, 0x42); // software power down
write_i2c(0x3103, 0x03); // SCCB system control
write_i2c(0x3017, 0x00); // set Frex, Vsync, Href, PCLK, D[9:6] input
write_i2c(0x3018, 0x00); // set d[5:0], GPIO[1:0] input
write_i2c(0x3034, 0x18); // MIPI 8-bit mode
write_i2c(0x3037, 0x13); // PLL
write_i2c(0x3108, 0x01); // system divider
write_i2c(0x3630, 0x36);
write_i2c(0x3631, 0x0e);
write_i2c(0x3632, 0xe2);
write_i2c(0x3633, 0x12);
write_i2c(0x3621, 0xe0);
write_i2c(0x3704, 0xa0);
write_i2c(0x3703, 0x5a);
write_i2c(0x3715, 0x78);
write_i2c(0x3717, 0x01);
write_i2c(0x370b, 0x60);
write_i2c(0x3705, 0x1a);
write_i2c(0x3905, 0x02);
write_i2c(0x3906, 0x10);
write_i2c(0x3901, 0x0a);
```

```
write_i2c(0x3731, 0x12);
write_i2c(0x3600, 0x08); // VCM debug mode
write_i2c(0x3601, 0x33); // VCM debug mode
write_i2c(0x302d, 0x60); // system control
write_i2c(0x3620, 0x52);
write_i2c(0x371b, 0x20);
write_i2c(0x471c, 0x50);
write_i2c(0x3a13, 0x43); // AGC pre-gain, 0x40 = 1x
write_i2c(0x3a18, 0x00); // gain ceiling
write_i2c(0x3a19, 0xf8);  // gain ceiling
write_i2c(0x3635, 0x13);
write_i2c(0x3636, 0x03);
write_i2c(0x3634, 0x40);
write_i2c(0x3622, 0x01);

// 50Hz/60Hz    去除 50/60Hz 灯光条纹
write_i2c(0x3c01, 0x34); // 50/60Hz
write_i2c(0x3c04, 0x28); // threshold for low sum
write_i2c(0x3c05, 0x98); // threshold for high sum
write_i2c(0x3c06, 0x00); // light meter 1 threshold high
write_i2c(0x3c08, 0x00); // light meter 2 threshold high
write_i2c(0x3c09, 0x1c); // light meter 2 threshold low
write_i2c(0x3c0a, 0x9c); // sample number high
write_i2c(0x3c0b, 0x40); // sample number low

// timing    时序
write_i2c(0x3800, 0x00); // HS
write_i2c(0x3801, 0x00); // HS
write_i2c(0x3802, 0x00); // VS
write_i2c(0x3804, 0x0a); // HW
write_i2c(0x3805, 0x3f);  // HW
write_i2c(0x3810, 0x00); // H offset high
write_i2c(0x3811, 0x10); // H offset low
write_i2c(0x3812, 0x00); // V offset high
write_i2c(0x3708, 0x64);
write_i2c(0x3a08, 0x01); // B50
write_i2c(0x4001, 0x02); // BLC start line
write_i2c(0x4005, 0x1a); // BLC always update
write_i2c(0x3000, 0x00); // system reset 0
write_i2c(0x3002, 0x1c); // system reset 2
write_i2c(0x3004, 0xff);  // clock enable 00
write_i2c(0x3006, 0xc3); // clock enable 2
write_i2c(0x300e, 0x45); // MIPI control, 2 lane, MIPI enable
write_i2c(0x302e, 0x08);
write_i2c(0x4300, 0x30); // YUV 422, YUYV
write_i2c(0x501f, 0x00);  // ISP YUV 422
write_i2c(0x4407, 0x04); // JPEG QS
write_i2c(0x440e, 0x00);
```

```
write_i2c(0x5000, 0xa7); // ISP control, Lenc on, gamma on, BPC on, WPC on, CIP on

// AWB    自动白平衡
write_i2c(0x5180, 0xff);
write_i2c(0x5181, 0xf2);
write_i2c(0x5182, 0x00);
write_i2c(0x5183, 0x14);
write_i2c(0x5184, 0x25);
write_i2c(0x5185, 0x24);
write_i2c(0x5186, 0x09);
write_i2c(0x5187, 0x09);
write_i2c(0x5188, 0x09);
write_i2c(0x5189, 0x75);
write_i2c(0x518a, 0x54);
write_i2c(0x518b, 0xe0);
write_i2c(0x518c, 0xb2);
write_i2c(0x518d, 0x42);
write_i2c(0x518e, 0x3d);
write_i2c(0x518f, 0x56);
write_i2c(0x5190, 0x46);
write_i2c(0x5191, 0xf8);
write_i2c(0x5192, 0x04);
write_i2c(0x5193, 0x70);
write_i2c(0x5194, 0xf0);
write_i2c(0x5195, 0xf0);
write_i2c(0x5196, 0x03);
write_i2c(0x5197, 0x01);
write_i2c(0x5198, 0x04);
write_i2c(0x5199, 0x12);
write_i2c(0x519a, 0x04);
write_i2c(0x519b, 0x00);
write_i2c(0x519c, 0x06);
write_i2c(0x519d, 0x82);
write_i2c(0x519e, 0x38);

// color matrix    色彩矩阵
write_i2c(0x5381, 0x1e);
write_i2c(0x5382, 0x5b);
write_i2c(0x5383, 0x08);
write_i2c(0x5384, 0x0a);
write_i2c(0x5385, 0x7e);
write_i2c(0x5386, 0x88);
write_i2c(0x5387, 0x7c);
write_i2c(0x5388, 0x6c);
write_i2c(0x5389, 0x10);
write_i2c(0x538a, 0x01);
write_i2c(0x538b, 0x98);
```

```
// CIP 锐化和降噪
write_i2c(0x5300, 0x08); // sharpen MT th1
write_i2c(0x5301, 0x30); // sharpen MT th2
write_i2c(0x5302, 0x10); // sharpen MT offset 1
write_i2c(0x5303, 0x00); // sharpen MT offset 2
write_i2c(0x5304, 0x08); // DNS threshold 1
write_i2c(0x5305, 0x30); // DNS threshold 2
write_i2c(0x5306, 0x08); // DNS offset 1
write_i2c(0x5307, 0x16); // DNS offset 2
write_i2c(0x5309, 0x08); // sharpen TH th1
write_i2c(0x530a, 0x30); // sharpen TH th2
write_i2c(0x530b, 0x04); // sharpen TH offset 1
write_i2c(0x530c, 0x06); // sharpen Th offset 2

// gamma    伽玛曲线
write_i2c(0x5480, 0x01);
write_i2c(0x5481, 0x08);
write_i2c(0x5482, 0x14);
write_i2c(0x5483, 0x28);
write_i2c(0x5484, 0x51);
write_i2c(0x5485, 0x65);
write_i2c(0x5486, 0x71);
write_i2c(0x5487, 0x7d);
write_i2c(0x5488, 0x87);
write_i2c(0x5489, 0x91);
write_i2c(0x548a, 0x9a);
write_i2c(0x548b, 0xaa);
write_i2c(0x548c, 0xb8);
write_i2c(0x548d, 0xcd);
write_i2c(0x548e, 0xdd);
write_i2c(0x548f, 0xea);
write_i2c(0x5490, 0x1d);

// UV adjust    UV 色彩饱和度调整
write_i2c(0x5580, 0x06); // sat on, contrast on
write_i2c(0x5583, 0x40); // sat U
write_i2c(0x5584, 0x10); // sat V
write_i2c(0x5589, 0x10); // UV adjust th1
write_i2c(0x558a, 0x00); // UV adjust th2[8]
write_i2c(0x558b, 0xf8);  // UV adjust th2[7:0]
write_i2c(0x501d, 0x04); // enable manual offset of contrast

// lens correction    镜头补偿
write_i2c(0x5800, 0x23);
write_i2c(0x5801, 0x14);
write_i2c(0x5802, 0x0f);
write_i2c(0x5803, 0x0f);
write_i2c(0x5804, 0x12);
```

```
write_i2c(0x5805, 0x26);
write_i2c(0x5806, 0x0c);
write_i2c(0x5807, 0x08);
write_i2c(0x5808, 0x05);
write_i2c(0x5809, 0x05);
write_i2c(0x580a, 0x08);
write_i2c(0x580b, 0x0d);
write_i2c(0x580c, 0x08);
write_i2c(0x580d, 0x03);
write_i2c(0x580e, 0x00);
write_i2c(0x580f, 0x00);
write_i2c(0x5810, 0x03);
write_i2c(0x5811, 0x09);
write_i2c(0x5812, 0x07);
write_i2c(0x5813, 0x03);
write_i2c(0x5814, 0x00);
write_i2c(0x5815, 0x01);
write_i2c(0x5816, 0x03);
write_i2c(0x5817, 0x08);
write_i2c(0x5818, 0x0d);
write_i2c(0x5819, 0x08);
write_i2c(0x581a, 0x05);
write_i2c(0x581b, 0x06);
write_i2c(0x581c, 0x08);
write_i2c(0x581d, 0x0e);
write_i2c(0x581e, 0x29);
write_i2c(0x581f, 0x17);
write_i2c(0x5820, 0x11);
write_i2c(0x5821, 0x11);
write_i2c(0x5822, 0x15);
write_i2c(0x5823, 0x28);
write_i2c(0x5824, 0x46);
write_i2c(0x5825, 0x26);
write_i2c(0x5826, 0x08);
write_i2c(0x5827, 0x26);
write_i2c(0x5828, 0x64);
write_i2c(0x5829, 0x26);
write_i2c(0x582a, 0x24);
write_i2c(0x582b, 0x22);
write_i2c(0x582c, 0x24);
write_i2c(0x582d, 0x24);
write_i2c(0x582e, 0x06);
write_i2c(0x582f, 0x22);
write_i2c(0x5830, 0x40);
write_i2c(0x5831, 0x42);
write_i2c(0x5832, 0x24);
write_i2c(0x5833, 0x26);
write_i2c(0x5834, 0x24);
```

```
write_i2c(0x5835, 0x22);
write_i2c(0x5836, 0x22);
write_i2c(0x5837, 0x26);
write_i2c(0x5838, 0x44);
write_i2c(0x5839, 0x24);
write_i2c(0x583a, 0x26);
write_i2c(0x583b, 0x28);
write_i2c(0x583c, 0x42);
write_i2c(0x583d, 0xce);

write_i2c(0x5025, 0x00);
write_i2c(0x3a0f, 0x30);  // stable in high
write_i2c(0x3a10, 0x28); // stable in low
write_i2c(0x3a1b, 0x30); // stable out high
write_i2c(0x3a1e, 0x26); // stable out low
write_i2c(0x3a11, 0x60); // fast zone high
write_i2c(0x3a1f, 0x14);  // fast zone low
write_i2c(0x3008, 0x02); // wake up
```

4.2.2 YCbCr VGA 预览 30 帧/秒

```
write_i2c(0x3035, 0x14); // pll
write_i2c(0x3036, 0x38); // pll
write_i2c(0x3c07, 0x08); // light meter 1 threshold
write_i2c(0x3820, 0x41); // ISP flip off, sensor flip off
write_i2c(0x3821, 0x07); // ISP mirror on, sensor mirror on
// timing 时序
write_i2c(0x3814, 0x31); // X inc
write_i2c(0x3815, 0x31); // Y inc
write_i2c(0x3803, 0x04); // VS
write_i2c(0x3806, 0x07); // VH
write_i2c(0x3807, 0x9b); // VH
write_i2c(0x3808, 0x02); // DVPHO
write_i2c(0x3809, 0x80); // DVPHO
write_i2c(0x380a, 0x01); // DVPVO
write_i2c(0x380b, 0xe0); // DVPVO
write_i2c(0x380c, 0x07); // HTS
write_i2c(0x380d, 0x68); // HTS
write_i2c(0x380e, 0x03); // VTS
write_i2c(0x380f, 0xd8);  // VTS
write_i2c(0x3813, 0x06); // V offset

write_i2c(0x3618, 0x00);
write_i2c(0x3612, 0x29);
write_i2c(0x3709, 0x52);
write_i2c(0x370c, 0x03);
```

```
write_i2c(0x3a02, 0x03); // 60Hz max exposure
write_i2c(0x3a03, 0xd8); // 60Hz max exposure
write_i2c(0x3a09, 0x27); // B50 low
write_i2c(0x3a0a, 0x00); // B60 high
write_i2c(0x3a0b, 0xf6);  // B60 low
write_i2c(0x3a0e, 0x03); // B50 max
write_i2c(0x3a0d, 0x04); // B60 max
write_i2c(0x3a14, 0x03); // 50Hz max exposure
write_i2c(0x3a15, 0xd8); // 50Hz max exposure

write_i2c(0x4004, 0x02); // BLC line number
write_i2c(0x4713, 0x03); // JPEG mode 3
write_i2c(0x460b, 0x35); // debug
write_i2c(0x460c, 0x22); // VFIFO, PCLK manual
write_i2c(0x4837, 0x44); // MIPI global timing
write_i2c(0x3824, 0x02); // PCLK divider
write_i2c(0x5001, 0xa3); // SDE on, scale on, UV average off, CMX on, AWB on
```

4.2.3 YCbCr 720p 视频 60 帧/秒

```
write_i2c(0x3035, 0x11); // pll
write_i2c(0x3036, 0x54); // pll
write_i2c(0x3c07, 0x07); // light meter 1 threshold
write_i2c(0x3820, 0x41); // ISP flip off, sensor flip off
write_i2c(0x3821, 0x07); // ISP mirror on, sensor mirror on
// timing 时序
write_i2c(0x3814, 0x31); // X inc
write_i2c(0x3815, 0x31); // Y inc
write_i2c(0x3803, 0xfa);  // VS
write_i2c(0x3806, 0x06); // VH
write_i2c(0x3807, 0xa9); // VH
write_i2c(0x3808, 0x05); // DVPHO
write_i2c(0x3809, 0x00); // DVPHO
write_i2c(0x380a, 0x02); // DVPVO
write_i2c(0x380b, 0xd0); // DVPVO
write_i2c(0x380c, 0x07); // HTS
write_i2c(0x380d, 0x64); // HTS
write_i2c(0x380e, 0x02); // VTS
write_i2c(0x380f, 0xe4);  // VTS
write_i2c(0x3813, 0x04); // V offset

write_i2c(0x3618, 0x00);
write_i2c(0x3612, 0x29);
write_i2c(0x3709, 0x52);
write_i2c(0x370c, 0x03);
// banding filter 去除灯光条纹
write_i2c(0x3a02, 0x02); // 60Hz max exposure
```

```
write_i2c(0x3a03, 0xe4); // 60Hz max exposure
write_i2c(0x3a09, 0xbc); // B50 low
write_i2c(0x3a0a, 0x01); // B60 high
write_i2c(0x3a0b, 0x72); // B60 low
write_i2c(0x3a0e, 0x01); // B50 max
write_i2c(0x3a0d, 0x02); // B60 max
write_i2c(0x3a14, 0x02); // 50Hz max exposure
write_i2c(0x3a15, 0xe4); // 50Hz max exposure

write_i2c(0x4004, 0x02); // BLC line number
write_i2c(0x4713, 0x02); // JPEG mode 2
write_i2c(0x460b, 0x37);
write_i2c(0x460c, 0x20); // VFIFO, PCLK auto
write_i2c(0x4837, 0x16); // MIPI global timing
write_i2c(0x3824, 0x04); // PCLK divider
write_i2c(0x5001, 0x83); // SDE on, scale off, UV average off, CMX on, AWB on
```

4.2.4 YCbCr 5 百万拍照 15 帧/秒

```
write_i2c(0x3035, 0x11); // pll
write_i2c(0x3036, 0x54); // pll
write_i2c(0x3c07, 0x07); // light meter 1 threshold
write_i2c(0x3820, 0x40); // ISP flip off, sensor flip off
write_i2c(0x3821, 0x06); // ISP mirror on, sensor mirror on
// timing 时序
write_i2c(0x3814, 0x11); // X inc
write_i2c(0x3815, 0x11); // Y inc
write_i2c(0x3803, 0x00); // VS
write_i2c(0x3806, 0x07); // VH
write_i2c(0x3807, 0x9f); // VH
write_i2c(0x3808, 0x0a); // DVPHO
write_i2c(0x3809, 0x20); // DVPHO
write_i2c(0x380a, 0x07); // DVPVO
write_i2c(0x380b, 0x98); // DVPVO
write_i2c(0x380c, 0x0b); // HTS
write_i2c(0x380d, 0x1c); // HTS
write_i2c(0x380e, 0x07); // VTS
write_i2c(0x380f, 0xb0); // VTS
write_i2c(0x3813, 0x04); // V offset

write_i2c(0x3618, 0x04);
write_i2c(0x3612, 0x2b);
write_i2c(0x3709, 0x12);
write_i2c(0x370c, 0x00);
// banding filter 去除灯光条纹
write_i2c(0x3a02, 0x07); // 60Hz max exposure
write_i2c(0x3a03, 0xb0); // 60Hz max exposure
```

```
write_i2c(0x3a09, 0x27); // B50 low
write_i2c(0x3a0a, 0x00); // B60 high
write_i2c(0x3a0b, 0xf6); // B60 low
write_i2c(0x3a0e, 0x06); // B50 max
write_i2c(0x3a0d, 0x08); // B60 max
write_i2c(0x3a14, 0x07); // 50Hz max exposure
write_i2c(0x3a15, 0xb0); // 50Hz max exposure

write_i2c(0x4004, 0x06); // BLC line number
write_i2c(0x4713, 0x00); // JPEG mode
write_i2c(0x460b, 0x35);
write_i2c(0x460c, 0x22); // VFIFO, PCLK manual
write_i2c(0x4837, 0x0a); // MIPI global timing
write_i2c(0x3824, 0x01); // PCLK divider
write_i2c(0x5001, 0x83); // SDE on, scale off, UV average off, CMX on, AWB on
```

4.3 驱动能力

寄存器	功能
0x302c[7:6]	00 – 1 倍驱动能力 01 – 2 倍驱动能力 10 – 3 倍驱动能力 11 – 4 倍驱动能力

4.4 I/O 控制

信号	输入/高阻态	输出数据路径	输出 - 0	输出 - 1
MDP1, MDN1, MCP, MCN, MDP0, MDN0	0x3017[3:0]=0x0 0x3018[7:6]=0x0	0x3017[3:0]=0xf 0x3018[7:6]=0x3 0x301d[3:0]=0x0 0x301e[7:6]=0x0	0x3017[3:0]=0xf 0x3018[7:6]=0x3 0x301d[3:0]=0xf 0x301e[7:6]=0x3 0x301a[3:0]=0x0 0x301b[76]=0x0	0x3017[3:0]=0xf 0x3018[7:6]=0x3 0x301d[3:0]=0xf 0x301e[7:6]=0x3 0x301a[3:0]=0xf 0x301b[7:6]=0x3
Vsync	0x3017[6]=0	0x3017[6]=1 0x301d[6]=0	0x3017[6]=1 0x301d[6]=1 0x301a[6]=0	0x3017[6]=1 0x301d[6]=1 0x301a[6]=1
Href	0x3017[5]=0	0x3017[5]=1 0x301d[5]=0	0x3017[5]=1 0x301d[5]=1 0x301a[5]=0	0x3017[5]=1 0x301d[5]=1 0x301a[5]=1
PCLK	0x3017[4]=0	0x3017[4]=1 0x301d[4]=0	0x3017[4]=1 0x301d[4]=1 0x301a[4]=0	0x3017[4]=1 0x301d[4]=1 0x301a[4]=1
Strobe	0x3016[1]=0	0x3016[1]=1	0x3016[1]=1	0x3016[1]=1

		0x301c[1]=0	0x301c[1]=1 0x3019[1]=0	0x301c[1]=1 0x3019[1]=1
FREX	0x3017[7]=0	0x3017[7]=1 0x301d[7]=0	0x3017[7]=1 0x301d[7]=1 0x301a[7]=0	0x3017[7]=1 0x301d[7]=1 0x301a[7]=1
GPIO1	0x3018[1]=0	0x3018[1]=1 0x301e[1]=0	0x3018[1]=1 0x301e[1]=1 0x301b[1]= 0	0x3018[1]=1 0x301e[1]=1 0x301b[1]= 0
GPIO0	0x3018[0]=0	0x3018[0]=1 0x301e[0]=0	0x3018[0]=1 0x301e[0]=1 0x301b[0]= 0	0x3018[0]=1 0x301e[0]=1 0x301b[0]= 0

注:

1. 如果任引脚在模组或系统中开路，引脚状态须设置为输出。

2. 如果任何引脚与其他设备共用总线，当 OV5640 处于低功耗状态而由其他设备控制总线时，引脚须设置为输入/高阻态；当 OV5640 和其他设备都处于低功耗状态时，其中须有一个引脚状态设置为输出以控制总线，而其他共用引脚设置为输入/高阻态。

4.5 MIPI 数据流控制

4.5.1 MIPI 数据流开启

write_i2c(0x4202, 0x00);

4.5.2 MIPI 数据流关闭

write_i2c(0x4202, 0x0f);

4.6 YUV 序列

为得到正确的图像，OV5640 输出图像的 YUV 序列必须与基带芯片或 ISP 匹配。

0x4300[0:1] 控制 YUV 输出顺序
//Y U Y V
write_i2c(0x4300, 0x30);

//Y V Y U
write_i2c(0x4300, 0x31);

//V Y U Y
write_i2c(0x4300, 0x33);

//U Y V Y
write_i2c(0x4300, 0x32);

4.7 镜像和翻转

因为 OV5640 是一款 BSI 图像传感器, 成像光线是从芯片背面射入的，所以原始生成的图像看起来是左右相反的，故此需要对图像做镜像处理使其显示正常。

i2c_salve_Address = 0x78;

MIRROR (default)　镜像（默认）

```
reg3820 = read_i2c(0x3820);
reg3820 = reg3820 & 0xf9;     // flip off
write_i2c(0x3820, reg3820);
reg3821 = read_i2c(0x3821);
reg3821 = reg3821 | 0x06;     // mirror on
write_i2c(0x3821, reg3821);
```

MIRROR

FLIP　翻转

```
reg3820 = read_i2c(0x3820);
reg3820 = reg3820 | 0x06;     // flip on
write_i2c(0x3820, reg3820);
reg3821 = read_i2c(0x3821);
reg3821 = reg3821 & 0xf9;     // mirror off
write_i2c(0x3821, reg3821);
```

FLIP

MIRROR&FLIP　镜像加翻转

```
reg3820 = read_i2c(0x3820);
reg3820 = reg3820 | 0x06;     // flip on
write_i2c(0x3820, reg3820);
reg3821 = read_i2c(0x3821);
reg3821 = reg3821 | 0x06;     // mirror on
write_i2c(0x3821, reg3821)
```

MIRROR&FLIP

Normal 正常

```
reg3820 = read_i2c(0x3820);
reg3820 = reg3820 & 0xf9;     // flip off
write_i2c(0x3820, reg3820);
reg3821 = read_i2c(0x3821);
reg3821 = reg3821 & 0xf9;     // mirror off
write_i2c(0x3821, reg3821);
```

NORML

4.8 测试图案

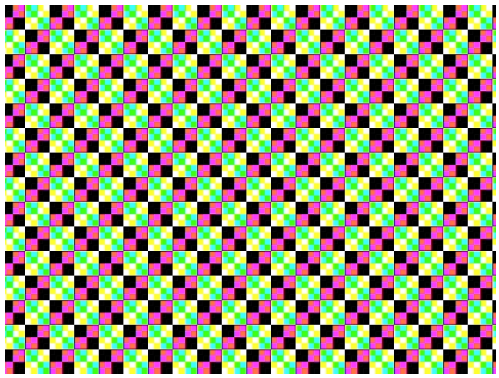
Color bar 彩色条

```
write_i2c(0x503d, 0x80);
write_i2c(0x4741, 0x00);
```

Color bar

Color square 彩色四方形

```
write_i2c(0x503d, 0x82);
write_i2c(0x4741, 0x0);
```



Color square

4.9 移除灯光条纹

OV5640_set_bandingfilter() 功能可在 camera 驱动中自动设置 banding filters（灯光条纹过滤）。

去除灯光条纹是通过将 OV5640 曝光时间设置为 $\frac{n}{100}$ 秒（$\frac{n}{120}$ 当交流电频率为 60Hz 时）来实现的。banding filter 数值明确了 OV5640 为得到 $\frac{n}{100}$ 秒（$\frac{n}{120}$ 60Hz）的曝光时间所对应的曝光行数。

$$Banding\_filter\_50hz = \frac{\frac{1}{100}}{line\_period} = \frac{\frac{1}{100}}{HTS \times sysclk} = \frac{sysclk}{HTS \times 100}$$

$$Banding\_filter\_60hz = \frac{\frac{1}{120}}{line\_period} = \frac{\frac{1}{120}}{HTS \times sysclk} = \frac{sysclk}{HTS \times 120}$$

HTS = register {0x380c, 0x380d}

sysclk 需由 OV5640_get_sysclk() 功能算得。

4.10 用户界面功能

4.10.1 亮度

亮度 +4
```
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5587, 0x40);
write_i2c(0x5588, 0x01);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3
```

亮度 +3
```
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5587, 0x30);
write_i2c(0x5588, 0x01);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3
```

亮度 +2
```
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5587, 0x20);
write_i2c(0x5588, 0x01);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3
```

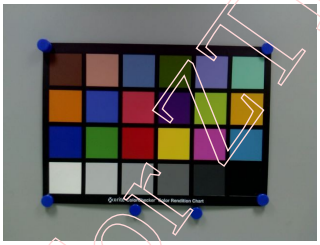
亮度+1
```
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5587, 0x10);
write_i2c(0x5588, 0x01);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3
```

缺省亮度
```
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5587, 0x00);
write_i2c(0x5588, 0x01);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3
```

亮度 -1
```
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5587, 0x10);
write_i2c(0x5588, 0x09);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3
```

亮度 -2
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5587, 0x20);
write_i2c(0x5588, 0x09);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3

亮度 -3
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5587, 0x30);
write_i2c(0x5588, 0x09);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3



亮度 -4
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5587, 0x40);
write_i2c(0x5588, 0x09);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3

4.10.2 对比度

对比度 +3
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5586, 0x2c);
write_i2c(0x5585, 0x1c);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3

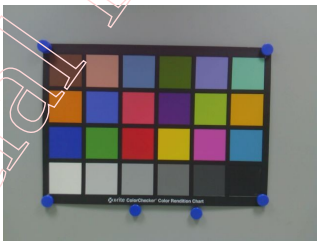
对比度 +2
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5586, 0x28);
write_i2c(0x5585, 0x18);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3

对比度 +1
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5586, 0x24);
write_i2c(0x5585, 0x10);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3

缺省对比度
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5586, 0x20);
write_i2c(0x5585, 0x00);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3

对比度 -1
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5586, 0x1c);
write_i2c(0x5585, 0x1c);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3

对比度 -2
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5586, 0x18);
write_i2c(0x5585, 0x18);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3



对比度 -3
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5586, 0x14);
write_i2c(0x5585, 0x14);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3

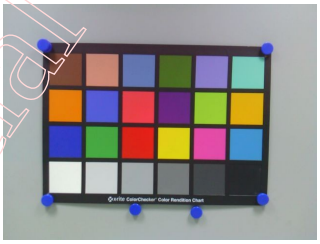
4.10.3 色饱和度

色饱和度 +3
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5381, 0x1c);
write_i2c(0x5382, 0x5a);
write_i2c(0x5383, 0x06);
write_i2c(0x5384, 0x2b);
write_i2c(0x5385, 0xab);
write_i2c(0x5386, 0xd6);
write_i2c(0x5387, 0xda);
write_i2c(0x5388, 0xd6);
write_i2c(0x5389, 0x04);
write_i2c(0x538b, 0x98);
write_i2c(0x538a, 0x01);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3

色饱和度　+2

```
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5381, 0x1c);
write_i2c(0x5382, 0x5a);
write_i2c(0x5383, 0x06);
write_i2c(0x5384, 0x24);
write_i2c(0x5385, 0x8f);
write_i2c(0x5386, 0xb3);
write_i2c(0x5387, 0xb6);
write_i2c(0x5388, 0xb3);
write_i2c(0x5389, 0x03);
write_i2c(0x538b, 0x98);
write_i2c(0x538a, 0x01);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3
```

色饱和度　+1



```
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5381, 0x1c);
write_i2c(0x5382, 0x5a);
write_i2c(0x5383, 0x06);
write_i2c(0x5384, 0x1f);
write_i2c(0x5385, 0x7a);
write_i2c(0x5386, 0x9a);
write_i2c(0x5387, 0x9c);
write_i2c(0x5388, 0x9a);
write_i2c(0x5389, 0x02);
write_i2c(0x538b, 0x98);
write_i2c(0x538a, 0x01);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3
```

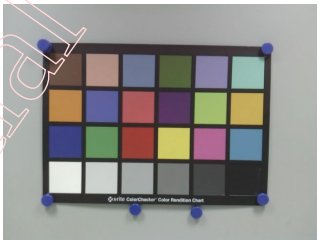
缺省色饱和度

```
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5381, 0x1c);
write_i2c(0x5382, 0x5a);
write_i2c(0x5383, 0x06);
write_i2c(0x5384, 0x1a);
write_i2c(0x5385, 0x66);
write_i2c(0x5386, 0x80);
write_i2c(0x5387, 0x82);
write_i2c(0x5388, 0x80);
write_i2c(0x5389, 0x02);
write_i2c(0x538b, 0x98);
write_i2c(0x538a, 0x01);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3
```

色饱和度 -1

```
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5381, 0x1c);
write_i2c(0x5382, 0x5a);
write_i2c(0x5383, 0x06);
write_i2c(0x5384, 0x15);
write_i2c(0x5385, 0x52);
write_i2c(0x5386, 0x66);
write_i2c(0x5387, 0x68);
write_i2c(0x5388, 0x66);
write_i2c(0x5389, 0x02);
write_i2c(0x538b, 0x98);
write_i2c(0x538a, 0x01);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3
```

色饱和度 -2



```
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5381, 0x1c);
write_i2c(0x5382, 0x5a);
write_i2c(0x5383, 0x06);
write_i2c(0x5384, 0x10);
write_i2c(0x5385, 0x3d);
write_i2c(0x5386, 0x4d);
write_i2c(0x5387, 0x4e);
write_i2c(0x5388, 0x4d);
write_i2c(0x5389, 0x01);
write_i2c(0x538b, 0x98);
write_i2c(0x538a, 0x01);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3
```

色饱和度 -3

```
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5381, 0x1c);
write_i2c(0x5382, 0x5a);
write_i2c(0x5383, 0x06);
write_i2c(0x5384, 0x0c);
write_i2c(0x5385, 0x30);
write_i2c(0x5386, 0x3d);
write_i2c(0x5387, 0x3e);
write_i2c(0x5388, 0x3d);
write_i2c(0x5389, 0x01);
write_i2c(0x538b, 0x98);
write_i2c(0x538a, 0x01);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3
```

4.10.4 EV 曝光补偿

EV +3
```
write_i2c(0x3a0f, 0x60);
write_i2c(0x3a10, 0x58);
write_i2c(0x3a11, 0xa0);
write_i2c(0x3a1b, 0x60);
write_i2c(0x3a1e, 0x58);
write_i2c(0x3a1f, 0x20);
```

EV +2
```
write_i2c(0x3a0f, 0x50);
write_i2c(0x3a10, 0x48);
write_i2c(0x3a11, 0x90);
write_i2c(0x3a1b, 0x50);
write_i2c(0x3a1e, 0x48);
write_i2c(0x3a1f, 0x20);
```

EV +1
```
write_i2c(0x3a0f, 0x40);
write_i2c(0x3a10, 0x38);
write_i2c(0x3a11, 0x71);
write_i2c(0x3a1b, 0x40);
write_i2c(0x3a1e, 0x38);
write_i2c(0x3a1f, 0x10);
```

缺省 EV
```
write_i2c(0x3a0f, 0x38);
write_i2c(0x3a10, 0x30);
write_i2c(0x3a11, 0x61);
write_i2c(0x3a1b, 0x38);
write_i2c(0x3a1e, 0x30);
write_i2c(0x3a1f, 0x10);
```

EV -1
```
write_i2c(0x3a0f, 0x30);
write_i2c(0x3a10, 0x28);
write_i2c(0x3a11, 0x61);
write_i2c(0x3a1b, 0x30);
write_i2c(0x3a1e, 0x28);
write_i2c(0x3a1f, 0x10);
```

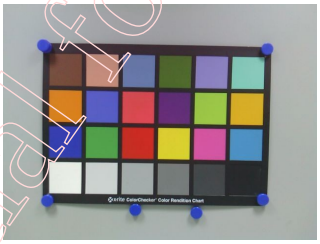
EV -2
```
write_i2c(0x3a0f, 0x20);
write_i2c(0x3a10, 0x18);
write_i2c(0x3a11, 0x41);
write_i2c(0x3a1b, 0x20);
write_i2c(0x3a1e, 0x18);
```

write_i2c(0x3a1f, 0x10);

EV -3
write_i2c(0x3a0f, 0x10);
write_i2c(0x3a10, 0x08);
write_i2c(0x3a1b, 0x10);
write_i2c(0x3a1e, 0x08);
write_i2c(0x3a11, 0x20);
write_i2c(0x3a1f, 0x10);

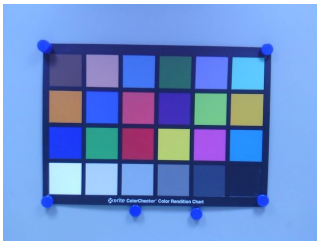
4.10.5 环境光模式

Auto 自动
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x3406, 0x00);
write_i2c(0x3400, 0x04);
write_i2c(0x3401, 0x00);
write_i2c(0x3402, 0x04);
write_i2c(0x3403, 0x00);
write_i2c(0x3404, 0x04);
write_i2c(0x3405, 0x00);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // lanuch group 3



Sunny 日光
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x3406, 0x01);
write_i2c(0x3400, 0x06);
write_i2c(0x3401, 0x1c);
write_i2c(0x3402, 0x04);
write_i2c(0x3403, 0x00);
write_i2c(0x3404, 0x04);
write_i2c(0x3405, 0xf3);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // lanuch group 3

Office 办公室
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x3406, 0x01);
write_i2c(0x3400, 0x05);
write_i2c(0x3401, 0x48);
write_i2c(0x3402, 0x04);
write_i2c(0x3403, 0x00);
write_i2c(0x3404, 0x07);
write_i2c(0x3405, 0xcf);
write_i2c(0x3212, 0x13); // end group 3

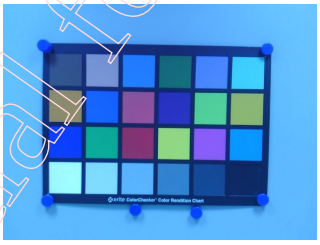


write_i2c(0x3212, 0xa3); // lanuch group 3

Cloudy 阴天

write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x3406, 0x01);
write_i2c(0x3400, 0x06);
write_i2c(0x3401, 0x48);
write_i2c(0x3402, 0x04);
write_i2c(0x3403, 0x00);
write_i2c(0x3404, 0x04);
write_i2c(0x3405, 0xd3);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // lanuch group 3

Home 室内



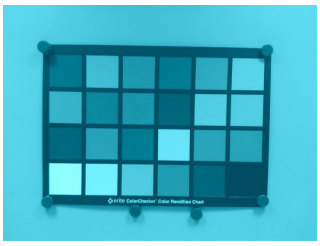
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x3406, 0x01);
write_i2c(0x3400, 0x04);
write_i2c(0x3401, 0x10);
write_i2c(0x3402, 0x04);
write_i2c(0x3403, 0x00);
write_i2c(0x3404, 0x08);
write_i2c(0x3405, 0x40);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // lanuch group 3

4.10.6 特效

Normal (off) 正常

write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5580, 0x06);
write_i2c(0x5583, 0x40); // sat U
write_i2c(0x5584, 0x10); // sat V
write_i2c(0x5003, 0x08);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3

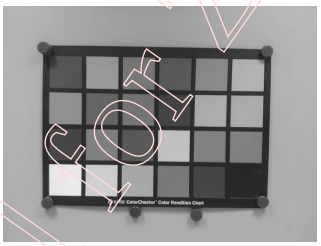
Blueish (cool light) 冷色



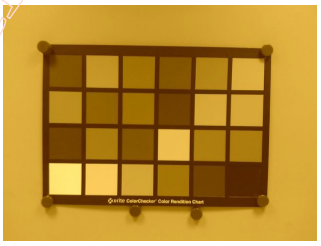
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5580, 0x1e);
write_i2c(0x5583, 0xa0);
write_i2c(0x5584, 0x40);
write_i2c(0x5003, 0x08);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3

Redish (warm) 暖色
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5580, 0x1e);
write_i2c(0x5583, 0x80);
write_i2c(0x5584, 0xc0);
write_i2c(0x5003, 0x08);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3

Black and white 黑白
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5580, 0x1e);
write_i2c(0x5583, 0x80);
write_i2c(0x5584, 0x80);
write_i2c(0x5003, 0x08);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3

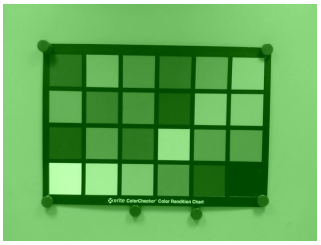


Sepia 泛黄
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5580, 0x1e);
write_i2c(0x5583, 0x40);
write_i2c(0x5584, 0xa0);
write_i2c(0x5003, 0x08);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3



Negative 反色
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5580, 0x40);
write_i2c(0x5003, 0x08);
write_i2c(0x5583, 0x40); // sat U
write_i2c(0x5584, 0x10); // sat V
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3

Greenish 偏绿
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5580, 0x1e);
write_i2c(0x5583, 0x60);
write_i2c(0x5584, 0x60);
write_i2c(0x5003, 0x08);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3

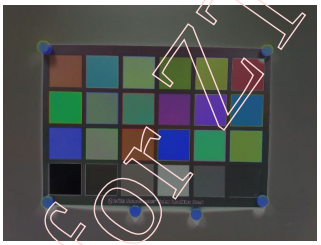


Overexposure 过曝

```
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5580, 0x1e);
write_i2c(0x5583, 0xf0);
write_i2c(0x5584, 0xf0);
write_i2c(0x5003, 0x08);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3
```

Solarize 正负片叠加

```
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5580, 0x06);
write_i2c(0x5583, 0x40); // sat U
write_i2c(0x5584, 0x10); // sat V
write_i2c(0x5003, 0x09);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3
```



4.10.7 夜景模式

进入夜景模式时，将自动加入 dummy lines（虚拟曝光行）以降低帧率。插入 dummy line（虚拟曝光行）的总行数根据光源不同分别由以下寄存器控制：0x3a02, 0x3a03（60hz 光源），0x3a14, 0x3a15（50hz 光源）。寄存器 0x3a00 bit[2] 用来控制夜景模式的开/关切换。

开启夜景模式

```
temp = read_i2c(0x3a00);
temp = temp | 0x04;
write_i2c(0x3a00, temp);
```

关闭夜景模式

```
temp = read_i2c(0x3a00);
temp = temp & 0xfb;
write_i2c(0x3a00, temp);
```

4.10.8 去除灯光条纹

OV5640 相机驱动支持自动计算灯光条纹过滤参数（banding filter），所以不用再进行手动设置。有以下四种设置可供选择：

Off 关闭

```
temp = read_i2c(0x3a00);
temp = temp & 0xdf;       // turn off banding filter
write_i2c(0x3a00, temp);
```

Manual 50Hz 手动设置 50Hz 光源

```
write_i2c(0x3c00, 04);     // set to 50Hz
```

```
write_i2c(0x3c01, 80);    // manual banding filter
temp = read_i2c(0x3a00);
temp = temp | 0x20;       // turn on banding filter
write_i2c(0x3a00, temp);
```

Manual 60Hz 手动设置 60Hz 光源

```
write_i2c(0x3c00, 00);    // set to 60Hz
write_i2c(0x3c01, 80);    // manual banding filter
temp = read_i2c(0x3a00);
temp = temp | 0x20;       // turn on banding filter
write_i2c(0x3a00, temp);
```

Auto Detection 自动侦测

```
write_i2c(0x3c01, 00);    // auto banding filter
temp = read_i2c(0x3a00);
temp = temp & 0xdf;       // turn off banding filter
write_i2c(0x3a00, temp);
```

自动侦测功能参数与输入时钟相关，如果输入时钟不是 24Mhz，请联络 OmniVision 当地的 FAE 取得正确的设置参数。

4.11 自动对焦

4.11.1 内置自动对焦

OV5640 由内置微型控制器完成自动对焦，并且 VCM 驱动器也已集成在传感器内部。微型控制器的控制固件（firmware）从主机下载。当固件运行后，内置微型控制器从 OV5640 传感器读得自动对焦所需的信息，计算并驱动 VCM 马达带动镜头到达正确的对焦位置。主机可以通过 I2C 命令控制微型控制器的各种功能。

4.11.2 I2C 自动对焦命令

寄存器名	地址	描述	值
CMD_MAIN	0x3022	AF 主命令寄存器	0x03 - 触发自动对焦过程 0x06 - 暂停对焦过程 0x08 - 释放马达回初始状态 0x12 - 重设对焦区域 0x00 – 命令完成
CMD_ACK	0x3023	ACK of command 命令确认	0x00 - 命令完成 0x01 - 命令运行中
FW_STATUS	0x3029	Status of focus	0x7F - S_FIRWARE

		对焦状态	固件已下载完毕但未运行 可能由以下两个原因 微控制器关闭 固件错误 0x7E - S_STARTUP 固件初始化中 0x70 - S_IDLEIdle 状态, 释放马达，镜头回到初始（对焦为无穷远处）位置 0x00 - S_FOCUSING 自动对焦中 0x10 S_FOCUSED 自动对焦完毕

注: 微程序控制器（MCU)收到自动对焦命令后会自动将 CMD_MAIN（0x3022）寄存器数据清零。当命令完成后会将 CMD_ACK（0x3023）寄存器数据清零。

4.11.3 AF 自动对焦过程

当使用 OV5640 自动对焦固件进行操作的流程，寄存器读写。 自动对焦流程如下:

1. 第一次进入图像预览时 下载固件（firmware）
2. 拍照时 自动对焦
3. 拍照完毕并回到图像预览时 释放马达至初始状态（对焦为无穷远处）

4.11.4 下载固件（download firmware）

AOV5640 初始化完成后，就可以下载 AF 自动对焦固件了，其操作和下载初始化参数是一样的。建议使用 I2C 多字节写入来加快固件的下载速度。

固件下载完成后，请检查以下寄存器：

MCU on:0x3000 BIT6=0 BIT5=0 0x3004 BIT6=1 BIT5=1

AFC on :0x3001 BIT6=00x3005 BIT6=1

4.11.5 自动对焦

在拍摄图片流程开始之前，自动对焦必须已经完成，才能得到清晰的图像。

1. 将 0x3022 寄存器写为 0x03，开始单点对焦过程。
2. 读取寄存器 0x3029 的状态，如果返回值为 0x10，代表对焦已完成。
3. 写寄存器 0x3022 为 0x06，暂停对焦过程，使镜头将保持在此对焦位置。

4.11.6 释放马达至初始状态 （对焦为无穷远处）

拍照完毕以后，写寄存器 0x3022 为 0x08，释放马达，镜头回到初始状态（对焦无穷远处）。

4.12 拍照流程

4.12.1 Shutter 快门时间

OV5640 传感器快门时间用来控制曝光时间，其单位为行频周期（line period），在每一种分辨率下曝光行都是有限制的，最大允许的快门时间值为 VTS(0x380e, 0x380f) - 4 。

快门时间对应寄存器的值为实际曝光行数的 16 倍，寄存器为：0x3500，0x3501，0x3502。

快门时间（以行频周期为单位）= (reg0x3500<<12) + (reg0x3501<<4) + (reg3502>>4);

4.12.2 Gain 增益

Gain 增益值设置在寄存器 0x350a 和 0x350b 中。

增益 = ((reg0x350a & 03)<<4) + (reg0x350b>>4);

4.12.3 虚拟曝光行（Dummy Lines）及虚拟像素（Dummy Pixels）

如果自动夜景模式开启，OV5640 会自动增加虚拟曝光行。插入 dummy line（虚拟曝光行）的总行数根据光源不同分别由以下寄存器控制：0x3a02[3:0], 0x3a03（60hz 光源），0x3a14[3:0], 0x3a15（50hz 光源）。

Dummy pixel（虚拟像素）用来调整时序，调整 HTS 寄存器 0x380c, 0x380d 可增加 dummy pixel（虚拟像素）。

4.12.4 拍照流程

4.12.4.1 自动对焦

触发单点自动对焦

暂停自动对焦过程 (保持镜头处在正确的对焦位置)

4.12.4.2 Read Preview Registers

Read preview_shutter 读取预览_shutter
Read preview_gain 读取预览_gain
Read preview_HTS 读取预览_HTS

Read preview_sysclk 读取预览_sysclk
Read preview_binning_factor B 读取预览_binning_factor B

4.12.4.3 改变为拍照图像分辨率

下载拍照设置参数。关闭 AGC/AEC(自动增益控制/自动曝光控制)。

4.12.4.4 读取拍照设置参数

Read capture _HTS 读取拍照_HTS
Read capture _VTS 读取拍照_VTS
Read capture_banding_filter 读取拍照_banding_filter
Read capture SYSCLK 读取拍照 SYSCLK

4.12.4.5 转换预览增益和曝光值为拍照增益和曝光值

预览和拍照的快门曝光时间必须一致。换算公式如下：

$$\text{拍照\_快门时间} = \text{预览\_快门时间} \times \frac{\text{拍照\_PCLK}}{\text{预览\_PCLK}} \times \frac{\text{预览\_HTS}}{\text{拍照\_HTS}} \times B$$

$$\text{拍照\_增益} = \text{预览\_增益}$$

其中 B 是像和并参数。OV5640 中 B 值为 1。

在光照较强的情况下，'当预览_快门时间'为 1，由于整数计算的误差，会得到'拍照_快门时间'等于 0。为避免这种情况，可以用以下公式计算：

$$\text{拍照\_快门时间} = \text{预览\_快门时间}$$

$$\text{拍照\_增益} = \text{预览\_增益} \times \frac{\text{拍照\_PCLK}}{\text{预览\_PCLK}} \times \frac{\text{预览\_HTS}}{\text{拍照\_HTS}} \times B$$

4.12.4.6 增益值转换为曝光值，拍照参数下去除灯光条纹

```
gain_exposure = capture_shutter*capture_gain;

If ( gain_exposure < capture_banding_filter) {
  // capture_shutter < 1/100
  capture_shutter = gain_exposure;
  capture_gain = gain_exposure/capture_shutter;
}
else {
  if ( gain_exposure > (capture_VTS -4)) {
      // exposure reach maximum
      capture_shutter = capture_VTS -4;
```

```
        capture_gain = gain_exposure/capture_shutter;
    }
    else {
    // 1/100 < exposure < max, capture_shutter = N * capture_banding_filter
    capture_shutter = int (gain_exposure/capture_banding_filter) * capture_banding_filter;
    capture_gain = gain_exposure/capture_shutter;
    }
}
```

4.12.4.7 写入增益（gain）/曝光值（shutter）

Write capture_gain 写入拍照_gain
Write capture_shutter 写入拍照_shutter

4.12.4.8 拍照

等待两个 Vsync 后。
抓取第三帧图像。

4.12.4.9 返回预览模式

// 释放马达，使镜头回到初始状态（对焦无穷远处）。

//写入预览设置参数，返回预览状态。

//开启 AGC/AEC。

write_i2c(0x3503, 0);　　写寄存器 0x3503 为 0。

4.13 Scale and Zoom 缩放和变焦

使用缩放和变焦功能，OV5640 ISP 寄存器 0x5001[5] 必须设置为 '1'。为保证缩放/变焦过程比较平滑，建议使用 I2C 群写入指令。

4.13.1 Scale 缩放

OV5640 ISP 支持图像缩小，可以得到任何小于 2592x1944 像素的图像尺寸。如果是按照与全尺寸图像宽高比（4:3）相同的比例进行缩小，那么调整寄存器 DVPHO 和 DVPVO 即可得到想要的尺寸。例如，要想得到分辨率为 2048x1536 的输出，其设置如下：

DVPHO = 2048

DVPVO = 1536

H offset = 16

Y offset = 4

```
write_i2c(0x3212, 0x03);        // start group 3
write_i2c(0x3808, 0x08);        // DVPHO = 2048
write_i2c(0x3809, 0x00);        // DVP HO
write_i2c(0x380a, 0x06);        // DVPVO = 1536
write_i2c(0x380b, 0x00);        // DVPVO
write_i2c(0x3810, 0x00);        // H offset = 16
write_i2c(0x3811, 0x10);        // H offset
write_i2c(0x3812, 0x00);        // V offset = 4
write_i2c(0x3813, 0x04);        // V offset
write_i2c(0x3212, 0x13);        // end group 3
write_i2c(0x3212, 0xa3);        // launch group 3
```

如果缩小尺寸与全尺寸图像宽高比（4:3）不同，那么还需调整 X offset 和 Y offset，对初始图像进行修剪以得到想要的图像尺寸。比如，要得到分辨率为 1280x1024 的输出：

DVPHO = 1280

DVPVO = 1024

input image height = 1944

input image width = 1944 * 1280/1024 = 2430

X offset = 16

Y offset = 4 + (2592 - 2430)/2 = 85

```
write_i2c(0x3212, 0x03);        // start group 3
write_i2c(0x3808, 0x05);        // DVPHO = 1280
write_i2c(0x3809, 0x00);        // DVP HO
write_i2c(0x380a, 0x04);        // DVPVO = 1024
write_i2c(0x380b, 0x00);        // DVPVO
write_i2c(0x3810, 0x00);        // H offset = 16
write_i2c(0x3811, 0x10);        // H offset
write_i2c(0x3812, 0x00);        // V offset = 85
write_i2c(0x3813, 0x55);        // V offset
write_i2c(0x3212, 0x13);        // end group 3
write_i2c(0x3212, 0xa3);        // launch group 3
```

4.13.2 Digital Zoom 数码变焦

对于任何小于 2592x1944 尺寸的图像，在保持图像宽高比与当前一致的情况下增加 X offset 和 Y offset，即可得到数码变焦的效果。比如当输出图像尺寸为 1600x1200 像素时：

Digital Zoom 1x （1 倍）

DVPHO = 1600

DVPVO = 1200

X offset = 16

Y offset = 4

```
write_i2c(0x3212, 0x03);        // start group 3
```

```
write_i2c(0x3808, 0x06);      // DVPHO = 1600
write_i2c(0x3809, 0x40);      // DVP HO
write_i2c(0x380a, 0x04);      // DVPVO = 1200
write_i2c(0x380b, 0xb0);      // DVPVO
write_i2c(0x3810, 0x00);      // H offset = 16
write_i2c(0x3811, 0x10);      // H offset
write_i2c(0x3812, 0x00);      // V offset = 4
write_i2c(0x3813, 0x04);      // V offset
write_i2c(0x3212, 0x13);      // end group 3
write_i2c(0x3212, 0xa3);      // launch group 3
```

Digital Zoom 1.5x （1.5 倍）

DVPHO = 1600

DVPVO = 1200

input image width = 2592/1.5 = 1728 > 1600

input image height = 1944/1.5 = 1296 > 1200

X offset = 16 + (2592 – 1728)/2 = 448

Y offset = 4 + (1944 – 1296)/2 = 328

```
write_i2c(0x3212, 0x03);      // start group 3
write_i2c(0x3808, 0x06);      // DVPHO = 1600
write_i2c(0x3809, 0x40);      // DVP HO
write_i2c(0x380a, 0x04);      // DVPVO = 1200
write_i2c(0x380b, 0xb0);      // DVPVO
write_i2c(0x3810, 0x01);      // H offset = 448
write_i2c(0x3811, 0xc0);      // H offset
write_i2c(0x3812, 0x01);      // V offset = 328
write_i2c(0x3813, 0x48);      // V offset
write_i2c(0x3212, 0x13);      // end group 3
write_i2c(0x3212, 0xa3);      // launch group 3
```

附录 I 双通道 MIPI 接口摄像头驱动程序示例

```
int m_iCombo_NightMode = 0;

int XVCLK = 2400;    // real clock/10000
int preview_sysclk, preview_HTS, preview_VTS;
int AE_Target = 52;
int AE_high, AE_low;

int OV5640_init_setting()
{
        // initialize OV5640

        int regInit[] =
        {

                //OV5640 setting Version History
                //dated 04/08/2010 A02
                //--Based on v08 release
                //
                //dated 04/20/2010 A03
                //--Based on V10 release
                //
                //dated 04/22/2010 A04
                //--Based on V10 release
                //--updated ccr & awb setting
                //
                //dated 04/22/2010 A06
                //--Based on A05 release
                //--Add pg setting
                //
                //dated 05/19/2011 A09
                //--changed pchg 3708 setting

                0x3008, 0x42,    // software power down
                0x3103, 0x03,    // SCCB system control
                0x3017, 0x00,    // set Frex, Vsync, Href, PCLK, D[9:6] input
                0x3018, 0x00,    // set d[5:0], GPIO[1:0] input
                0x3034, 0x18,    // MIPI 8-bit mode
                0x3037, 0x13,    // PLL
                0x3108, 0x01,    // system divider
                0x3630, 0x36,
                0x3631, 0x0e,
                0x3632, 0xe2,
                0x3633, 0x12,
                0x3621, 0xe0,
                0x3704, 0xa0,
                0x3703, 0x5a,
                0x3715, 0x78,
                0x3717, 0x01,
                0x370b, 0x60,
                0x3705, 0x1a,
                0x3905, 0x02,
                0x3906, 0x10,
                0x3901, 0x0a,
                0x3731, 0x12,
                0x3600, 0x08,    // VCM debug mode
                0x3601, 0x33,    // VCM debug mode
                0x302d, 0x60,    // system control
                0x3620, 0x52,
                0x371b, 0x20,
```

```
0x471c, 0x50,
0x3a13, 0x43,    // AGC pre-gain, 0x40 = 1x
0x3a18, 0x00,    // gain ceiling
0x3a19, 0xf8,    // gain ceiling
0x3635, 0x13,
0x3636, 0x03,
0x3634, 0x40,
0x3622, 0x01,

// 50Hz/60Hz
0x3c01, 0x34,    // 50/60Hz
0x3c04, 0x28,    // threshold for low sum
0x3c05, 0x98,    // threshold for high sum
0x3c06, 0x00,    // light meter 1 threshold high
0x3c08, 0x00,    // light meter 2 threshold high
0x3c09, 0x1c,    // light meter 2 threshold low
0x3c0a, 0x9c,    // sample number high
0x3c0b, 0x40,    // sample number low

// timing
0x3800, 0x00,    // HS
0x3801, 0x00,    // HS
0x3802, 0x00,    // VS
0x3804, 0x0a,    // HW
0x3805, 0x3f,    // HW
0x3810, 0x00,    // H offset high
0x3811, 0x10,    // H offset low
0x3812, 0x00,    // V offset high
0x3708, 0x64,
0x3a08, 0x01,    // B50
0x4001, 0x02,    // BLC start line
0x4005, 0x1a, // BLC always update
0x3000, 0x00,    // system reset 0
0x3002, 0x1c,    // system reset 2
0x3004, 0xff,    // clock enable 00
0x3006, 0xc3,    // clock enable 2
0x300e, 0x45,    // MIPI control, 2 lane, MIPI on
0x302e, 0x08,
0x4300, 0x30,    // YUV 422, YUYV
0x501f, 0x00,    // ISP YUV 422
0x4407, 0x04,    // JPEG QS
0x440e, 0x00,
0x5000, 0xa7,    // ISP control, Lenc on, gamma on, BPC on, WPC on, CIP on

// AWB
0x5180, 0xff,
0x5181, 0xf2,
0x5182, 0x00,
0x5183, 0x14,
0x5184, 0x25,
0x5185, 0x24,
0x5186, 0x09,
0x5187, 0x09,
0x5188, 0x09,
0x5189, 0x75,
0x518a, 0x54,
0x518b, 0xe0,
0x518c, 0xb2,
0x518d, 0x42,
0x518e, 0x3d,
0x518f, 0x56,
0x5190, 0x46,
0x5191, 0xf8,
0x5192, 0x04,
```

```
0x5193, 0x70,
0x5194, 0xf0,
0x5195, 0xf0,
0x5196, 0x03,
0x5197, 0x01,
0x5198, 0x04,
0x5199, 0x12,
0x519a, 0x04,
0x519b, 0x00,
0x519c, 0x06,
0x519d, 0x82,
0x519e, 0x38,

// color matrix
0x5381, 0x1e,
0x5382, 0x5b,
0x5383, 0x08,
0x5384, 0x0a,
0x5385, 0x7e,
0x5386, 0x88,
0x5387, 0x7c,
0x5388, 0x6c,
0x5389, 0x10,
0x538a, 0x01,
0x538b, 0x98,

// CIP
0x5300, 0x08,    // sharpen MT th1
0x5301, 0x30,    // sharpen MT th2
0x5302, 0x10,    // sharpen MT offset 1
0x5303, 0x00,    // sharpen MT offset 2
0x5304, 0x08,    // DNS threshold 1
0x5305, 0x30,    // DNS threshold 2
0x5306, 0x08,    // DNS offset 1
0x5307, 0x16,    // DNS offset 2
0x5309, 0x08,    // sharpen TH th1
0x530a, 0x30,    // sharpen TH th2
0x530b, 0x04,    // sharpen TH offset 1
0x530c, 0x06,    // sharpen Th offset 2

// gamma
0x5480, 0x01,
0x5481, 0x08,
0x5482, 0x14,
0x5483, 0x28,
0x5484, 0x51,
0x5485, 0x65,
0x5486, 0x71,
0x5487, 0x7d,
0x5488, 0x87,
0x5489, 0x91,
0x548a, 0x9a,
0x548b, 0xaa,
0x548c, 0xb8,
0x548d, 0xcd,
0x548e, 0xdd,
0x548f, 0xea,
0x5490, 0x1d,

// UV adjust
0x5580, 0x06,    // sat on, contrast on
0x5583, 0x40,    // sat U
0x5584, 0x10,    // sat VV
0x5589, 0x10,    // UV adjust th1
```

```
0x558a, 0x00,    // UV adjust th2[8]
0x558b, 0xf8,    // UV adjust th2[7:0]
0x501d, 0x40,    // enable manual offset of contrast

// lens correction
0x5800, 0x23,
0x5801, 0x14,
0x5802, 0x0f,
0x5803, 0x0f,
0x5804, 0x12,
0x5805, 0x26,
0x5806, 0x0c,
0x5807, 0x08,
0x5808, 0x05,
0x5809, 0x05,
0x580a, 0x08,
0x580b, 0x0d,
0x580c, 0x08,
0x580d, 0x03,
0x580e, 0x00,
0x580f, 0x00,
0x5810, 0x03,
0x5811, 0x09,
0x5812, 0x07,
0x5813, 0x03,
0x5814, 0x00,
0x5815, 0x01,
0x5816, 0x03,
0x5817, 0x08,
0x5818, 0x0d,
0x5819, 0x08,
0x581a, 0x05,
0x581b, 0x06,
0x581c, 0x08,
0x581d, 0x0e,
0x581e, 0x29,
0x581f, 0x17,
0x5820, 0x11,
0x5821, 0x11,
0x5822, 0x15,
0x5823, 0x28,
0x5824, 0x46,
0x5825, 0x26,
0x5826, 0x08,
0x5827, 0x26,
0x5828, 0x64,
0x5829, 0x26,
0x582a, 0x24,
0x582b, 0x22,
0x582c, 0x24,
0x582d, 0x24,
0x582e, 0x06,
0x582f, 0x22,
0x5830, 0x40,
0x5831, 0x42,
0x5832, 0x24,
0x5833, 0x26,
0x5834, 0x24,
0x5835, 0x22,
0x5836, 0x22,
0x5837, 0x26,
0x5838, 0x44,
0x5839, 0x24,
0x583a, 0x26,
```

```
            0x583b, 0x28,
            0x583c, 0x42,
            0x583d, 0xce,

            0x5025, 0x00,
            0x3a0f, 0x30,    // stable in high
            0x3a10, 0x28,    // stable in low
            0x3a1b, 0x30,    // stable out high
            0x3a1e, 0x26,    // stable out low
            0x3a11, 0x60,    // fast zone high
            0x3a1f, 0x14,    // fast zone low
            0x4202, 0x0f,    // stream off
            0x3008, 0x02,    // wake up
        };

        OV5640_write_i2c(0x3103, 0x11);      // sysclk from pad
        OV5640_write_i2c(0x3008, 0x82);      // software reset

        // delay 5ms
        Delay(5);

        // Write initialization table
        for (int i=0; i<sizeof(regInit)/sizeof(int); i+=2)
        {
            OV5640_write_i2c(regInit[i], regInit[i+1]);
        }

        return 0;
}

int OV5640_preview_setting()
{
        // set OV5640 to preview mode

        int regPreview[] =
        {
            // 640x480 15fps, night mode 5fps
            // Input CLock = 24Mhz
            // PCLK = 17Mhz
            0x3035, 0x14,    // pll
            0x3036, 0x38,    // pll
            0x3c07, 0x08,    // light meter 1 threshold
            0x3820, 0x41,    // ISP flip off, sensor flip off
            0x3821, 0x07,    // ISP mirror on, sensor mirror on
            // timing
            0x3814, 0x31,    // X inc
            0x3815, 0x31,    // Y inc
            0x3803, 0x04,    // VS
            0x3806, 0x07,    // VH
            0x3807, 0x9b,    // VH
            0x3808, 0x02,    // DVPHO
            0x3809, 0x80,    // DVPHO
            0x380a, 0x01,    // DVPVO
            0x380b, 0xe0,    // DVPVO
            0x380c, 0x07,    // HTS
            0x380d, 0x68,    // HTS
            0x380e, 0x03,    // VTS
            0x380f, 0xd8,    // VTS
            0x3813, 0x06,    // V offset

            0x3618, 0x00,
            0x3612, 0x29,
            0x3709, 0x52,
            0x370c, 0x03,
```

```
            0x3a02, 0x0b,    // 60Hz max exposure, 10fps
            0x3a03, 0x88,    // 60Hz max exposure
            0x3a14, 0x0b,    // 50Hz max exposure, 10fps
            0x3a15, 0x88,    // 50Hz max exposure

            0x4004, 0x02,    // BLC line number
            0x4713, 0x03,    // JPEG mode 3
            0x460b, 0x35,    // debug
            0x460c, 0x22,    // VFIFO, PCLK manual
            0x4837, 0x44,    // MIPI global timing
            0x3824, 0x02,    // PCLK divider
            0x5001, 0xa3,    // SDE on, scale on, UV average off, CMX on, AWB on

            0x3503, 0x00,    // AGC on, AEC on
        };

        // Write preview table
        for (int i=0; i<sizeof(regPreview)/sizeof(int); i+=2)
        {
            OV5640_write_i2c(regPreview[i], regPreview[i+1]);
        }

        return 0;
}

int OV5640_video_setting()
{
        // set OV5640 to video mode

        int regVideo[] =
        {
            // input clock 24Mhz
            // PCLK 42Mhz
            0x3035, 0x11,    // pll
            0x3036, 0x54,    // pll
            0x3c07, 0x07,    // light meter 1 threshold
            0x3820, 0x41,    // ISP flip off, sensor flip off
            0x3821, 0x07,    // ISP mirror on, sensor mirror on
            // timing
            0x3814, 0x31,    // X inc
            0x3815, 0x31,    // Y inc
            0x3803, 0xfa,    // VS
            0x3806, 0x06,    // VH
            0x3807, 0xa9,    // VH
            0x3808, 0x05,    // DVPHO
            0x3809, 0x00,    // DVPHO
            0x380a, 0x02,    // DVPVO
            0x380b, 0xd0,    // DVPVO
            0x380c, 0x07,    // HTS
            0x380d, 0x64,    // HTS
            0x380e, 0x02,    // VTS
            0x380f, 0xe4,    // VTS
            0x3813, 0x04,    // V offset

            0x3618, 0x00,
            0x3612, 0x29,
            0x3709, 0x52,
            0x370c, 0x03,
            // banding filter
            0x3a02, 0x02,    // 60Hz max exposure, fixed frame rate
            0x3a03, 0xe4,    // 60Hz max exposure
            0x3a14, 0x02,    // 50Hz max exposure, fixed frame rate
            0x3a15, 0xe4,    // 50Hz max exposure
```

```
            0x4004, 0x02,    // BLC line number
            0x4713, 0x02,    // JPEG mode 2
            0x460b, 0x37,
            0x460c, 0x20,    // VFIFO, PCLK auto
            0x4837, 0x16,    // MIPI global timing
            0x3824, 0x04,    // PCLK divider
            0x5001, 0x83,    // SDE on, scale off, UV average off, CMX on, AWB on

            0x3503, 0x00,    // AGC on, AEC on
        };

        // Write video table
        for (int i=0; i<sizeof(regVideo)/sizeof(int); i+=2)
        {
            OV5640_write_i2c(regVideo[i], regVideo[i+1]);
        }

        return 0;
}

int OV5640_capture_setting()
{
        // set OV5640 to capture mode

        int regCapture[] =
        {
            // YUV Capture
            // 2592 x 1944 15fps
            // 24 MHz input clock, 42Mhz PCLK


            0x3212, 0x00,    // start group 0

            0x3035, 0x11,    // pll
            0x3036, 0x54,    // pll
            0x3c07, 0x07,    // light meter 1 threshold
            0x3820, 0x40,    // ISP flip off, sensor flip off
            0x3821, 0x06,    // ISP mirror on, sensor mirror on
            // timing
            0x3814, 0x11,    // X inc
            0x3815, 0x11,    // Y inc
            0x3803, 0x00,    // VS
            0x3806, 0x07,    // VH
            0x3807, 0x9f,    // VH
            0x3808, 0x0a,    // DVPHO
            0x3809, 0x20,    // DVPHO
            0x380a, 0x07,    // DVPVO
            0x380b, 0x98,    // DVPVO
            0x380c, 0x0b,    // HTS
            0x380d, 0x1c,    // HTS
            0x380e, 0x07,    // VTS
            0x380f, 0xb0,    // VTS
            0x3813, 0x04,    // V offset

            0x3618, 0x04,
            0x3612, 0x2b,
            0x3709, 0x12,
            0x370c, 0x00,

            0x4004, 0x06,    // BLC line number
            0x4713, 0x00,    // JPEG mode
            0x460b, 0x35,
            0x460c, 0x22,    // VFIFO, PCLK manual
```

```
            0x4837, 0x0a,     // MIPI global timing
            0x3824, 0x01,     // PCLK divider
            0x5001, 0x83,     // SDE on, scale off, UV average off, CMX on, AWB on
            0x3212, 0x10,     // end group 0
            0x3212, 0xa0,     // launch group 0

            0x3503, 0x03,     // AGC off, AEC off
        };

        // Write capture table
        for (int i=0; i<sizeof(regCapture)/sizeof(int); i+=2)
        {
            OV5640_write_i2c(regCapture[i], regCapture[i+1]);
        }

        return 0;
}

int  OV5640_af_init()
{
        // download firmware
        // if supported, multiple bytes I2C writes are highly recommended.
        for (int i=0; i<sizeof(af_firmware)/sizeof(int); i+=2)
        {
                OVPantherDemo::WriteSCCB(0x78, af_firmware[i], af_firmware[i+1]);
        }

        return 0;
}

int  OV5640_auto_focus()
{
        int temp;
        // focus
                        OV5640_write_i2c(0x3022, 0x03);

        while(1)
        {
            // check status
            temp = OV5640_read_i2c(0x3029);
            if (temp ==0x10) return 0;   // focus completed
        }
         return 1;

        Delay(100);

}

int OV5640_get_sysclk()
{
        // calculate sysclk
        int temp1, temp2;
        int Multiplier, PreDiv, VCO, SysDiv, Pll_rdiv, Bit_div2x, sclk_rdiv, sysclk;

        int sclk_rdiv_map[] = {
            1, 2, 4, 8};

        temp1 = OV5640_read_i2c(0x3034);
        temp2 = temp1 & 0x0f;
        if (temp2 == 8 || temp2 == 10) {
            Bit_div2x = temp2 / 2;
        }

        temp1 = OV5640_read_i2c(0x3035);
```

```
        SysDiv = temp1>>4;
        if(SysDiv == 0) {
                SysDiv = 16;
        }

        temp1 = OV5640_read_i2c(0x3036);
        Multiplier = temp1;

        temp1 = OV5640_read_i2c(0x3037);
        PreDiv = temp1 & 0x0f;
        Pll_rdiv = ((temp1 >> 4) & 0x01) + 1;

        temp1 = OV5640_read_i2c(0x3108);
        temp2 = temp1 & 0x03;
        sclk_rdiv = sclk_rdiv_map[temp2];

        VCO = XVCLK * Multiplier / PreDiv;

        sysclk = VCO / SysDiv / Pll_rdiv * 2 / Bit_div2x / sclk_rdiv;

        return sysclk;
}

int OV5640_get_HTS()
{
        // read HTS from register settings
        int HTS;

        HTS = OV5640_read_i2c(0x380c);
        HTS = (HTS<<8) + OV5640_read_i2c(0x380d);

        return HTS;
}

int OV5640_get_VTS()
{
        // read VTS from register settings
        int VTS;

        VTS = OV5640_read_i2c(0x380e);
        VTS = (VTS<<8) + OV5640_read_i2c(0x380f);

        return VTS;
}

int OV5640_set_VTS(int VTS)
{
        // write VTS to registers
        int temp;

        temp = VTS & 0xff;
        OV5640_write_i2c(0x380f, temp);

        temp = VTS>>8;
        OV5640_write_i2c(0x380e, temp);

        return 0;
}

int OV5640_get_shutter()
{
        // read shutter, in number of line period
        int shutter;
```

```
	shutter = (OV5640_read_i2c(0x03500) & 0x0f);
	shutter = (shutter<<8) + OV5640_read_i2c(0x3501);
	shutter = (shutter<<4) + (OV5640_read_i2c(0x3502)>>4);

	return shutter;
}

int OV5640_set_shutter(int shutter)
{
	// write shutter, in number of line period
	int temp;

	shutter = shutter & 0xffff;

	temp = shutter & 0x0f;
	temp = temp<<4;
	OV5640_write_i2c(0x3502, temp);

	temp = shutter & 0xfff;
	temp = temp>>4;
	OV5640_write_i2c(0x3501, temp);

	temp = shutter>>12;
	OV5640_write_i2c(0x3500, temp);

	return 0;
}

int OV5640_get_gain16()
{
	// read gain, 16 = 1x
	int gain16;

	gain16 = OV5640_read_i2c(0x350a) & 0x03;
	gain16 = (gain16<<8) + OV5640_read_i2c(0x350b);

	return gain16;
}

int OV5640_set_gain16(int gain16)
{
	// write gain, 16 = 1x
	int temp;
	gain16 = gain16 & 0x3ff;

	temp = gain16 & 0xff;
	OV5640_write_i2c(0x350b, temp);

	temp = gain16>>8;
	OV5640_write_i2c(0x350a, temp);

	return 0;
}

int OV5640_get_light_frequency()
{
	// get banding filter value
	int temp, temp1, light_frequency;

	temp = OV5640_read_i2c(0x3c01);

	if (temp & 0x80) {
		// manual
		temp1 = OV5640_read_i2c(0x3c00);
```

```
            if (temp1 & 0x04) {
                // 50Hz
                light_frequency = 50;
            }
            else {
                // 60Hz
                light_frequency = 60;
            }
        }
        else {
            // auto
            temp1 = OV5640_read_i2c(0x3c0c);
            if (temp1 & 0x01) {
                // 50Hz
                light_frequency = 50;
            }
            else {
                // 60Hz
            }
        }
        return light_frequency;
}

void OV5640_set_bandingfilter()
{
        int preview_VTS;
        int band_step60, max_band60, band_step50, max_band50;

        // read preview PCLK
        preview_sysclk = OV5640_get_sysclk();

        // read preview HTS
        preview_HTS = OV5640_get_HTS();

        // read preview VTS
        preview_VTS = OV5640_get_VTS();

        // calculate banding filter
        // 60Hz
        band_step60 = preview_sysclk * 100/preview_HTS * 100/120;
        OV5640_write_i2c(0x3a0a, (band_step60 >> 8));
        OV5640_write_i2c(0x3a0b, (band_step60 & 0xff));

        max_band60 = int((preview_VTS-4)/band_step60);
        OV5640_write_i2c(0x3a0d, max_band60);

        // 50Hz
        band_step50 = preview_sysclk * 100/preview_HTS;
        OV5640_write_i2c(0x3a08, (band_step50 >> 8));
        OV5640_write_i2c(0x3a09, (band_step50 & 0xff));

        max_band50 = int((preview_VTS-4)/band_step50);
        OV5640_write_i2c(0x3a0e, max_band50);
}

int OV5640_set_AE_target(int target)
{
        // stable in high
        int fast_high, fast_low;
        AE_low = target * 23 / 25;   // 0.92
        AE_high = target * 27 / 25;  // 1.08

        fast_high = AE_high<<1;
        if(fast_high>255)
```

```
            fast_high = 255;

        fast_low = AE_low>>1;

        OV5640_write_i2c(0x3a0f, AE_high);
        OV5640_write_i2c(0x3a10, AE_low);
        OV5640_write_i2c(0x3a1b, AE_high);
        OV5640_write_i2c(0x3a1e, AE_low);
        OV5640_write_i2c(0x3a11, fast_high);
        OV5640_write_i2c(0x3a1f, fast_low);

        return 0;
}

void OV5640_MIPI_stream_on()
{
        OV5640_write_i2c(0x4202, 0x00);
}

void OV5640_MIPI_stream_off()
{
        OV5640_write_i2c(0x4202, 0x0f);
}

int OV5640_init()
{
        // initialize OV5640
        OV5640_init_setting();


        return 0;
}

int OV5640_preview()
{
        // MIPI stream off
        OV5640_MIPI_stream_off();

        // set OV5640 to preview mode
        OV5640_preview_setting();

        // calculate banding filter
        OV5640_set_bandingfilter();

        // set ae_target
        OV5640_set_AE_target(AE_Target);

        // update night mode setting
        OV5640_set_night_mode(m_iCombo_NightMode);

        // MIPI stream on
        OV5640_MIPI_stream_on();

        // download auto focus firmware
        OV5640_af_init();

        return 0;
}

int OV5640_return_to_preview()
{
        // release focus
        OV5640_write_i2c(0x3022, 0x08);
```

```
        // MIPI stream off
        OV5640_MIPI_stream_off();

        // set OV5640 to preview mode
        OV5640_preview_setting();

        // calculate banding filter
        OV5640_set_bandingfilter();

        // set ae_target
        OV5640_set_AE_target(AE_Target);

        // update night mode setting
        OV5640_set_night_mode(m_iCombo_NightMode);

        // MIPI stream on
        OV5640_MIPI_stream_on();

        // re-launch auto focus zones
        OV5640_write_i2c(0x3022, 0x12);

        return 0;
}

int OV5640_video()
{
        // MIPI stream off
        OV5640_MIPI_stream_off();

        // set OV5640 to video mode
        OV5640_video_setting();

        // calculate banding filter
        OV5640_set_bandingfilter();

        // set ae_target
        OV5640_set_AE_target(AE_Target);

        // turn off night mmode
        OV5640_set_night_mode(0);

        // MIPI stream on
        OV5640_MIPI_stream_on();

        return 0;
}

int OV5640_capture()
{
        // set OV5640 to capture mode

        int preview_shutter, preview_gain16, average;
        int capture_sysclk, capture_HTS, capture_VTS;
        int capture_shutter, capture_gain16;
        int light_frequency, capture_bandingfilter, capture_max_band;
        long capture_gain16_shutter;

        //auto focus
        OV5640_auto_focus();

        // read preview shutter
        preview_shutter = OV5640_get_shutter();

        // read preview gain
```

```
        preview_gain16 = OV5640_get_gain16();

        // get average
        average = OV5640_read_i2c(0x56a1);

        // turn off night mode for capture
        OV5640_set_night_mode(0);

        // turn off overlay
         OV5640_write_i2c(0x3022, 0x06);

        // MIPI stream off
        OV5640_MIPI_stream_off();

        // Write capture setting
        OV5640_capture_setting();

        // read capture VTS
        capture_VTS = OV5640_get_VTS();
        capture_HTS = OV5640_get_HTS();
        capture_sysclk = OV5640_get_sysclk();

        // calculate capture banding filter
        light_frequency = OV5640_get_light_frequency();
        if (light_frequency == 60) {
                // 60Hz
                capture_bandingfilter = capture_sysclk * 100 / capture_HTS * 100 / 120;
        }
        else {
                // 50Hz
                capture_bandingfilter = capture_sysclk * 100 / capture_HTS;
        }
        capture_max_band = int((capture_VTS - 4)/capture_bandingfilter);

        // calculate capture shutter/gain16
        if (average > AE_low && average < AE_high) {
                // in stable range
                capture_gain16_shutter = preview_gain16 * preview_shutter * capture_sysclk/preview_sysclk *
preview_HTS/capture_HTS * AE_Target / average;
        }
        else {
                capture_gain16_shutter = preview_gain16 * preview_shutter * capture_sysclk/preview_sysclk *
preview_HTS/capture_HTS;
        }

        // gain to shutter
        if(capture_gain16_shutter < (capture_bandingfilter * 16)) {
                // shutter < 1/100
                capture_shutter = capture_gain16_shutter/16;
                if(capture_shutter <1)
                        capture_shutter = 1;

                capture_gain16 = capture_gain16_shutter/capture_shutter;
                if(capture_gain16 < 16)
                        capture_gain16 = 16;
        }
        else {
                if(capture_gain16_shutter > (capture_bandingfilter*capture_max_band*16)) {
                        // exposure reach max
                        capture_shutter = capture_bandingfilter*capture_max_band;
                        capture_gain16 = capture_gain16_shutter / capture_shutter;
                }
                else {
                        // 1/100 < capture_shutter =< max, capture_shutter = n/100
```

```
                capture_shutter = (int (capture_gain16_shutter/16/capture_bandingfilter)) * capture_bandingfilter;
                capture_gain16 = capture_gain16_shutter / capture_shutter;
            }
        }


        // write capture gain
        OV5640_set_gain16(capture_gain16);

        // write capture shutter
        if (capture_shutter > (capture_VTS - 4)) {
            capture_VTS = capture_shutter + 4;
            OV5640_set_VTS(capture_VTS);
        }
        OV5640_set_shutter(capture_shutter);

        // skip 2 vysnc

        // start capture at 3rd vsync

        // start capture at 3rd vsync
        OV5640_MIPI_stream_on();

        return 0;
}

void OV5640_set_light_frequency(int LightFrequency)
{

        int temp;

        switch (LightFrequency)
        {

        case 0://Off
            temp = OV5640_read_i2c(0x3a00);
            temp = temp & 0xdf;                 // turn off band filter, bit[5] = 0
            OV5640_write_i2c(0x3a00, temp);

            break;

        case 1:// 50Hz
            OV5640_write_i2c(0x3c00, 0x04);     // set manual banding 50Hz
            OV5640_write_i2c(0x3c01, 0x80);     // enable manual banding

            temp = OV5640_read_i2c(0x3a00);
            temp = temp | 0x20;                 // turn on band filter, bit[5] = 1
            OV5640_write_i2c(0x3a00, temp);
            break;

        case 2:// 60Hz
            OV5640_write_i2c(0x3c00, 0x00);     // set manual banding 60Hz
            OV5640_write_i2c(0x3c01, 0x80);     // enable manual banding

            temp = OV5640_read_i2c(0x3a00);
            temp = temp | 0x20;                 // turn on band filter, bit[5] = 1
            OV5640_write_i2c(0x3a00, temp);
            break;

        case 3:// auto
            OV5640_write_i2c(0x3c01, 0x00);     // enable auto banding

            temp = OV5640_read_i2c(0x3a00);
            temp = temp | 0x20;                 // turn on band filter, bit[5] = 1
```

```
            OV5640_write_i2c(0x3a00, temp);
            break;

        default:
            break;

        }
}

void OV5640_set_night_mode(int NightMode)
{

        int temp;

        switch (NightMode)
        {

        case 0://Off
            temp = OV5640_read_i2c(0x3a00);
            temp = temp & 0xfb;                 // night mode off, bit[2] = 0
            OV5640_write_i2c(0x3a00, temp);

            break;

        case 1:// On
            temp = OV5640_read_i2c(0x3a00);
            temp = temp | 0x04;                 // night mode on, bit[2] = 1
            OV5640_write_i2c(0x3a00, temp);

            break;

        default:
            break;

        }
}
```

文档版本历史

Rev 2.01

合并“OV5640 模组软件应用指南”和“OV5640 模组硬件应用指南”。

Rev 2.10

加入自动对焦支持。

Rev 2.11

更新对比度设置参数 (基于 DB 文件 A10 或者 AM10)。

增加 “4.10 自动对焦”章节。

升级演示驱动。将驱动代码从“Form1.h” 移动至“PantherDemo.cpp”。

R 2.12

修正驱动代码中拍照曝光部分的程序 bug。

在初始化参数中加入“write_i2c(0x4005, 0x1a); // BLC 总是更新”。

由 DVP 接口修改为 MIPI 接口。

初始化参数更新。

R2.13

增加“4.3 驱动能力”和“4.4 I/O 控制”。