OV5640 Auto Focus Camera Module Application Notes (with MIPI Interface)

Last Modified: Feb. 21nd, 2012

Document Revision: 2.15

OmniVision Technologies, Inc. reserves the right to make changes without further notice to any product herein to improve reliability, function or design. OmniVision does not assume any liability arising out of the application or use of any project, circuit described herein; neither does it convey any license under its patent nor the right of others.

This document contains information of a proprietary nature. None of this information shall be divulged to persons other than OmniVision Technologies, Inc. employee authorized by the nature of their duties to receive such information, or individuals or organizations authorized by OmniVision Technologies, Inc.

内容目录

1. Overview of OV5640 Application....4
2. Hardware Design....5
2.1 OV5640 Camera Module Reference Design....5
2.2 Host Interface....6
2.1.1 Pin Definition....6
2.2 Power Supply....6
2.3 Deal with Lens....7
2.3.1 Light fall off....7
2.3.2 Dark corner....7
2.3.3 Resolution....7
2.3.4 Optical contrast....7
2.3.5 Lens Cover....7
2.3.6 Lens Correction....7
2.3.6.1 Lens Correction:....7
2.3.6.2 Lens Correction: 7
3. Hardware Operation....8
3.1 Operation Modes....8
3.1.1 Power Up....8
3.1.2 Power Down....9
3.1.3 Wake up from Power down....9
3.1.4 Power OFF 9
3.1.5 Hardware Reset....10
3.2 Operations....10
3.2.1 Cut of power when not used....10
3.2.2 Power down when not used....10
3.2.3 OV5640 share I2C bus with other devices....10
4. Software Operation....11
4.1 1 Lane MIPI Interface....11
4.1.1 YCbCr Initial Setting....11
4.1.2 YCbCr VGA Preview @ 30fps....16
4.1.3 YCbCr 720p Video @ 30fps....17
4.1.4 YCbCr 5M Capture @ 7.5fps....18
4.2 2 Lane MIPI Interface....20
4.2.1 YCbCr Initial Setting....20
4.2.2 YCbCr VGA Preview @ 30fps....25
4.2.3 YCbCr 720p @ 60fps....26
4.2.4 YCbCr 5M Capture @ 15fps....27
4.3 Drive Capability....28
4.4 I/O Control....28
4.5 MIPI Stream Control....29
4.5.1 MIPI Stream on....29
4.5.2 MIPI Stream off....29
4.6 YUV Sequence 29
4.7 Mirror and Flip....30
4.8 Test Pattern....31
4.9 Remove Light Band....31

4.10 User Interface Functions....32
4.10.1 Brightness....32
4.10.2 Contrast....33
4.10.3 Saturation....34
4.10.4 EV....37
4.10.5 Light Mode....38
4.10.6 Special Effects....39
4.10.7 Night Mode....41
4.10.8 Banding Filter Selection....41
4.11 Auto Focus....42
4.11.1 Embedded Auto Focus Solution....42
4.11.2 I2C Commands for Auto Focus....42
4.11.3 AF Sequence....43
4.11.4 Download firmware....43
4.11.5 Auto focus....43
4.11.6 Release Focus....43
4.12 Capture Sequence....43
4.12.1 Shutter....43
4.12.2 Gain....44
4.12.3 Dummy Lines and Dummy Pixels....44
4.12.4 Capture Sequence....44
4.12.4.1 Auto Focus....44
4.12.4.2 Read Preview Registers....44
4.12.4.3 Change Resolution to Capture....44
4.12.4.4 Read Capture Register Values....44
4.12.4.5 Preview Gain/Exposure to Capture/Gain Exposure....45
4.12.4.6 Gain to Exposure and Capture Banding Filter....45
4.12.4.7 Write gain/exposure value....45
4.12.4.8 Capture....46
4.12.4.9 Back to Preview....46
4.13 Scale and Zoom....46
4.13.1 Scale....46
4.13.2 Digital Zoom....47
Appendix I Sample Code of Camera Driver for 2 Lane MIPI....49
Revision History....65

1. Overview of OV5640 Application

OV5640 is a 1/4 inch high performance 5M camera supporting both DVP and MIPI interface. This document focus on DVP interface application of OV5640. For MIPI interface application of OV5640, please read “OV5640 Camera Module Application Notes (with MIPI interface)”

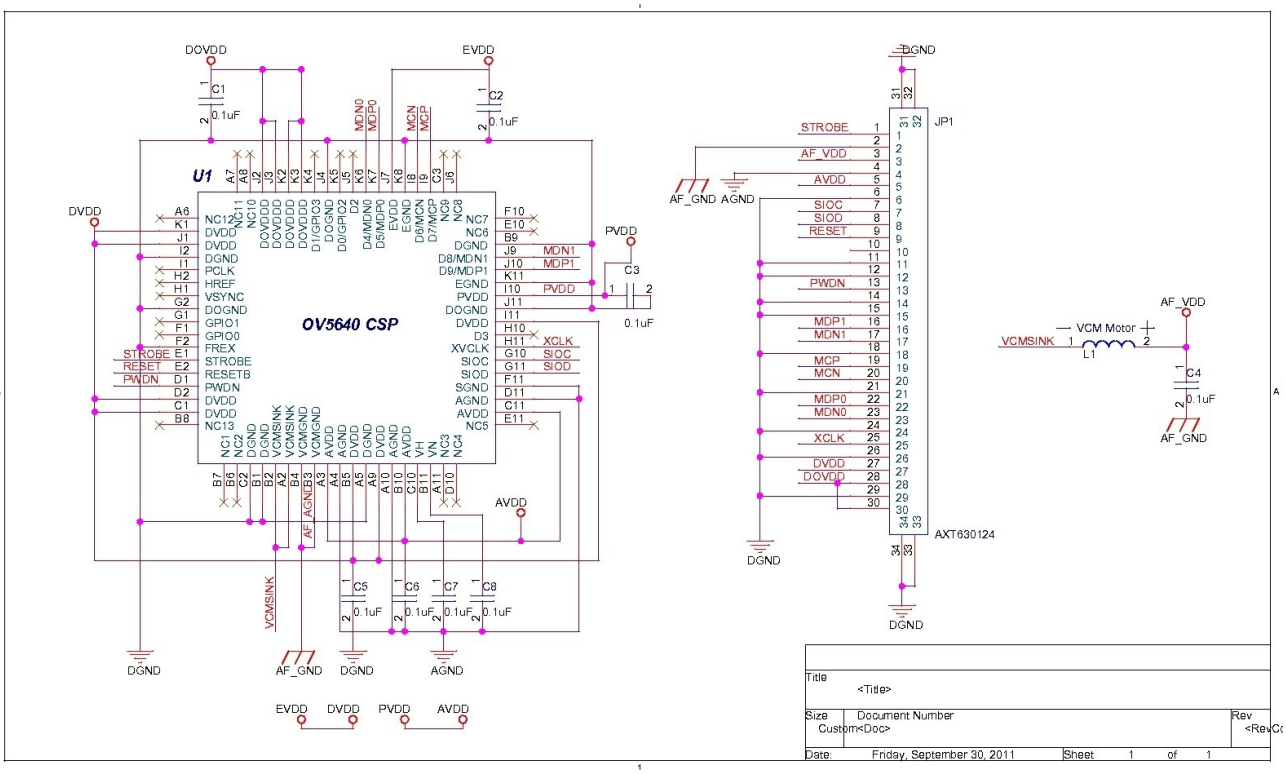
OV5640 could be used as

5M Main Camera for Cellular Phone Applications

5M Main Camera for Tablet Applications

2. Hardware Design

2.1 OV5640 Camera Module Reference Design



Note:

1. PWND, active HIGH as DOVDD to power down OV5640, should be connected to ground outside of module if unused
2. RESETB. Active LOW to reset OV5640, should be connected to DOVDD outside of module if unused
3. AVDD is 2.6-3.0V of sensor analog power (clean). 2.8V is recommended. AVDD must be 2.5V+-5% for OTP write, and OTP read does not have such requirement
4. DVDD is 1.5V±5% of sensor digital power(clean). Using the internal DVDD regulator is strongly recommended
5. DOVDD. 1.8V recommended is 1.7V-3.0V of sensor digital IO power(clean)
6. sensor AGND and DGND should be separated and connected to a single point outside module, Do not connect it inside module
7. Capacitors should be close to the related sensor pins
8. MCP, MCN, MDP0 and MDP1 are 1 lane MIPI interface. MCP, MCN, MDP0, MDN0, MDP1 and MCP1 are 2 lane MIPI interface.

2.2 Host Interface

2.1.1 Pin Definition

The video port of OV5640 support 2 lane MIPI and 1 lane MIPI. MCP, MCN, MDP0 and MDP1 are 1 lane MIPI interface. MCP, MCN, MDP0, MDN0, MDP1 and MCP1 are 2 lane MIPI interface.

The Href and Hsync signal is on the same pin – Href. The function of this pin could be selected by SCCB setting.

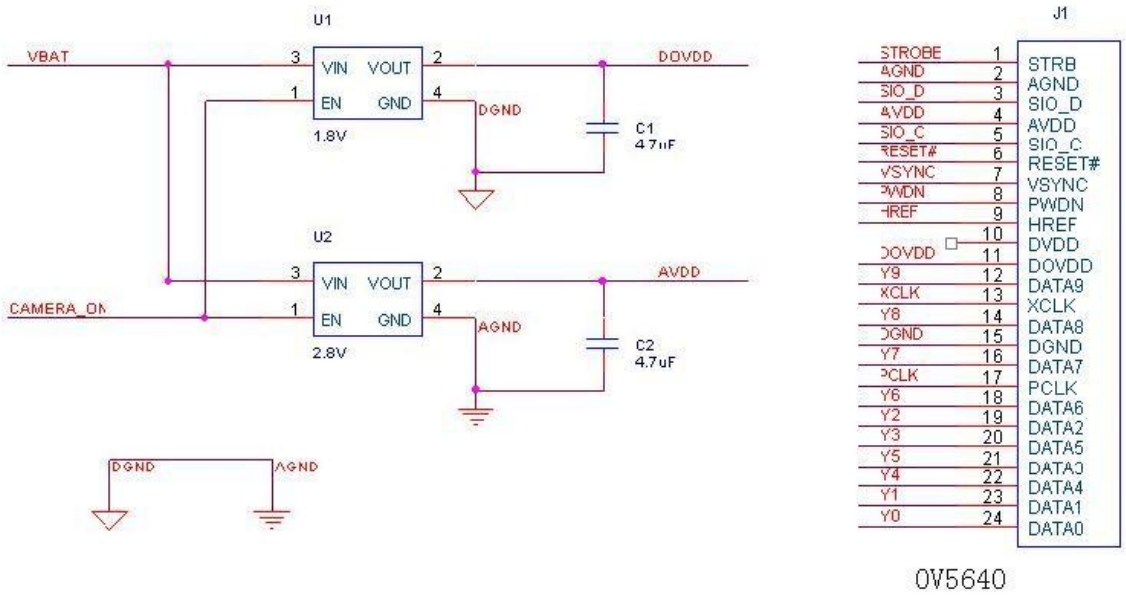
The SIO_C and SIO_D bus should have external pull up resistors, the typical value of the pull up resistors is 5.1K.

RESETB is active low with internal pull-up resistor. RESETB should be controlled by backend chip for proper power up sequence.

PWDN is active high with internal pull-down resistor. PWDN should be controlled by backend chip for proper power up sequence.

2.2 Power Supply

When OV5640 is used with back-end chip supporting MIPI interface, DOVDD of OV5650 should be 1.8V. DVDD is generated by internal regulator of OV5640. So 2 regulators should be used to supply powers to OV5640.



2.3 Deal with Lens

2.3.1 Light fall off

Light fall off means the corner of image is darker than center of image. It is caused by the lens. The lens shading correction function of OV5640 could be turned on to compensate the corner brightness and make the whole picture looks same bright.

2.3.2 Dark corner

Some lens may have dark corner. Dark corner means the color of picture looks almost black. It is not possible to correct dark corner with lens correction. So the module with dark corner is NG, it can not be used.

2.3.3 Resolution

The resolution of camera module depends on lens design, focus adjustment and sensor resolution as well. The focus adjustment is very important for camera module assembly.

2.3.4 Optical contrast

The optical contrast of lens is very important to picture quality. If the optical contrast of lens is not good, the picture would looks foggy. Though it could be improved by increase the sensor contrast to make the picture sharper, the higher sensor contrast would make the detail lost of dark area of the picture.

2.3.5 Lens Cover

The lens cover is the cheapest part in optical path. But it could affect picture quality very much. The lens cover should be made with optical glass with AR coating at both side. Otherwise, the lens cover may cause sensitivity loss and/or stronger lens flare.

2.3.6 Lens Correction

Lens Correction setting should be tuned for every module. Please find lens correction settings of modules tuned by OVT FAE.

2.3.6.1 Lens Correction:

2.3.6.2 Lens Correction:

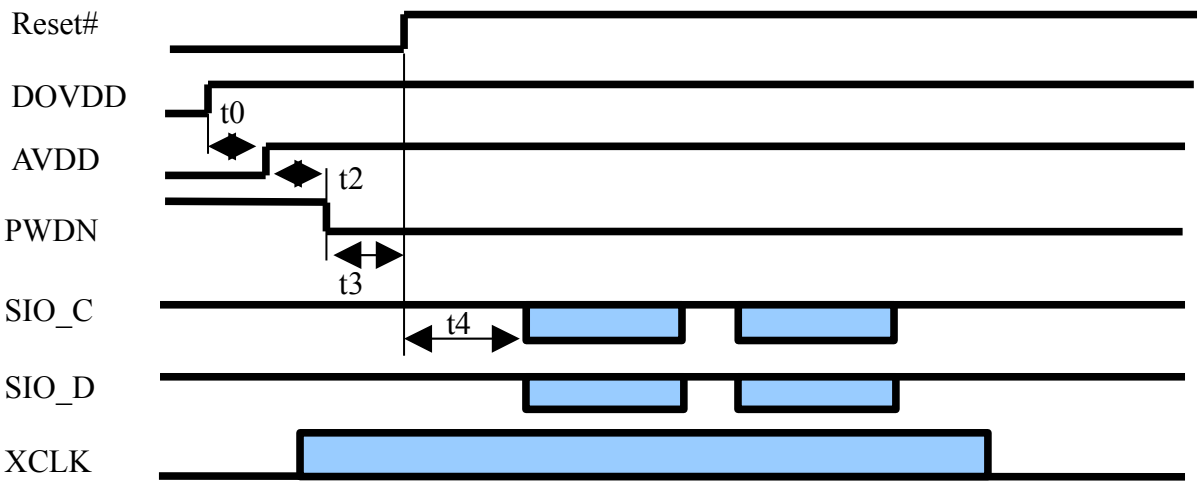
Note:

If module/lens you are using can not be found above, please contact with OmniVision local FAE for lens correction settings.

3. Hardware Operation

3.1 Operation Modes

3.1.1 Power Up



t0: >= 0ms. Delay from DOVDD stable to AVDD stable.
t2: >= 5ms. Delay from AVDD stable to sensor power up stable.
t3: >= 1ms. Delay from sensor power up stable to Reset# pull high.
t4: >=20ms. Delay from Reset pull high to SCCB initialization.

Step 1:
ResetB is applied to OV5640 camera module. PWDN is pulled high.

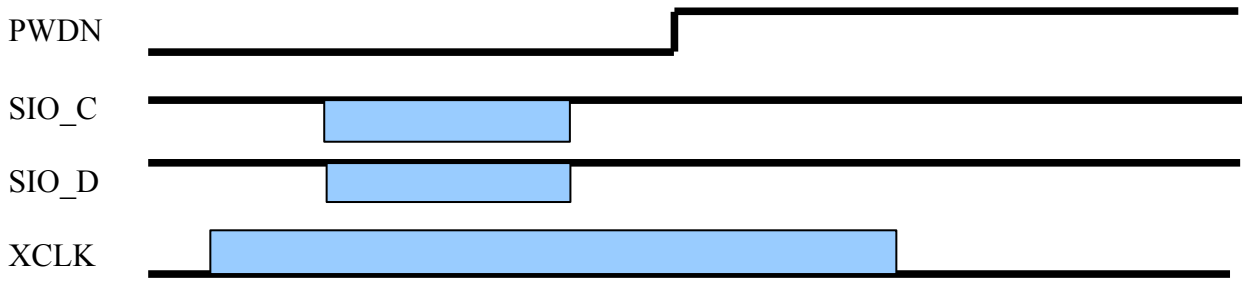
Step 2:
DOVDD and AVDD powers are applied. The 2 powers could be applied simultaneously. If applied separately, the power on sequence should be DOVDD first, and AVDD last.

Step 3:
after 5ms of AVDD reaching stable, pull PWDN to low.

Step 4:
after 1ms of PWDN go low, pull high ResetB.

Step 5:
After 20ms, initialize OV5640 by SCCB initialization. Please find the settings from section 4.

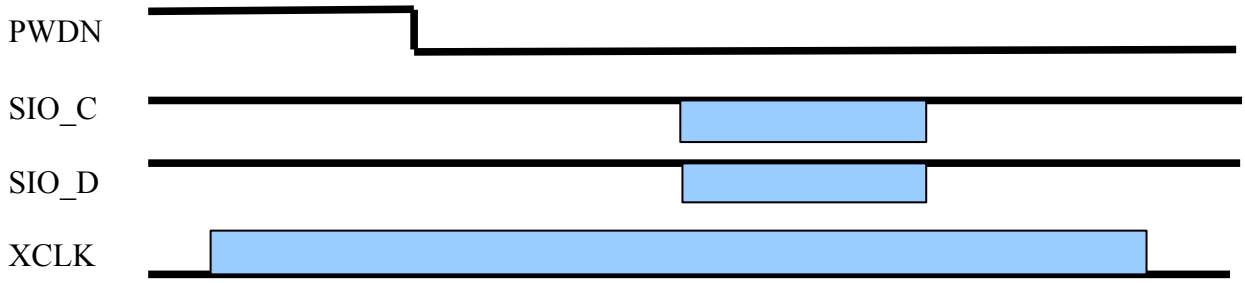
3.1.2 Power Down



Step 1:
Pull PWDN pin high.

Step 2:
Pull XCLK low. XCLK should be keep more than 0.1ms after PWDN pulled high.

3.1.3 Wake up from Power down



Step 1:
Apply XCLK

Step 2:
after 0.1ms, Pull low PWDN

Optional Step 3:
Full SCCB Initialization. Please find the settings from section 4.

3.1.4 Power OFF

Step 1.
Pull low XCLK,

Step 2.
Turn off AVDD, DVDD and DOVDD. The 3 powers could be turned off simultaneously. If turned off separately, DVDD should be turned off first, AVDD second and DOVDD third.

Step 3.

Pull Low PWDN and RESETB.

3.1.5 Hardware Reset

OV5640 sensor includes a RESETB pin that forces a complete hardware reset when it is pulled low(GND). OV5640 clears all registers and resets them to their default values when a hardware reset occurred. A reset can also be initiated through the SCCB interface by setting register 0x12[7] to high.

The whole chip will be reset during power up. Manually applying a hard reset after power up is recommended even through the on-chip power up reset is included. The hard reset is active low with an asynchronous design. The reset pulse width should be greater than or equal to 1ms.

3.2 Operations

3.2.1 Cut of power when not used

Mode	Operation
Battery On	No
Enter Camera	Power On Hardware Reset Initialization
Exit Camera	Power Off

3.2.2 Power down when not used

Mode	
Battery On	Power Up Hardware Reset Power Down
Enter Camera	Wake up from Power Down Initialization
Exit From Camera	Power Down

3.2.3 OV5640 share I2C bus with other devices

States other devices access I^2C	States other devices can not access I^2C
Power Off, Power Down, Wake up from Power Down Power up	None

4. Software Operation

4.1 1 Lane MIPI Interface

4.1.1 YCbCr Initial Setting

```
//
//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
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
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
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
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
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
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
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
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 Preview @ 30fps

```
//input 24M
//output  vga 30fps  bit rate 224M bps
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
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 Video @ 30fps

```
//input 24M
//ourput 720p 30fps  bit rate 336
write_i2c(0x3108, 0x02); // system divider
write_i2c(0x3035, 0x21); // 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
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
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 5M Capture @ 7.5fps

```
//input 24M
//output  5M 7.5fps  bit rate 672M bps
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
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
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 2 Lane MIPI Interface

4.2.1 YCbCr Initial Setting

```
//
//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
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
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
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
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
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
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
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
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 Preview @ 30fps

```
//input 24M
//output  vga 30fps  bit rate 224M bps
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
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 @ 60fps

```
//input 24M
//output  720p 60fps  bit rate 672M bps
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
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
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 5M Capture @ 15fps

```
//input 24M
//output  5M 15fps  bit rate 672M bps
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
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
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 Drive Capability

Register bit	Function
0x302c[7:6]	00 – 1x 01 – 2x 10 – 3x 11 – 4x

4.4 I/O Control

Signal	Input/High-z	Output data path	Output - 0	Output - 1
D3	0x3018[5]=0	0x3018[5]=1 0x301e[5]=0 (Output DVP)	0x3018[5]=1 0x301e[5]=1 0x301b[5]= 0	0x3018[5]=1 0x301e[5]=1 0x301b[5]= 1
D2	0x3018[4]=0	0x3018[4]=1 0x301e[4]=0 (Output DVP)	0x3018[4]=1 0x301e[4]=1 0x301b[4]= 0	0x3018[4]=1 0x301e[4]=1 0x301b[4]= 1
D1	0x3018[3]=0	0x3018[3]=1 0x301e[3]=0 (Output DVP)	0x3018[3]=1 0x301e[3]=1 0x301b[3]= 0	0x3018[3]=1 0x301e[3]=1 0x301b[3]= 1
D0	0x3018[2]=0	0x3018[2]=1	0x3018[2]=1	0x3018[2]=1

		0x301e[2]=0 (Output DVP)	0x301e[2]=1 0x301b[2]= 0	0x301e[2]=1 0x301b[2]= 1
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 0x301c[1]=0	0x3016[1]=1 0x301c[1]=1 0x3019[1]=0	0x3016[1]=1 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]=1 0x301b[1]= 0	0x3018[1]=1 0x301e[1]=1 0x301b[1]= 0
GPIO0	0x3018[0]=0		0x3018[0]=1 0x301e[0]=1 0x301b[0]= 0	0x3018[0]=1 0x301e[0]=1 0x301b[0]= 0

Note:

1. If any pin is open in camera module or open in system, the pin should be configured as output.

2. If any pin shares same bus with other devices, when OV5640 is set to standby mode and other device is driving the bus, the pin should be set to input/High-z. If any pin shares same bus with other devices, when both OV5640 and other devices are set to standby mode, one pin should set to output to drive the bus and other pin should set to input/high-Z.

4.5 MIPI Stream Control

4.5.1 MIPI Stream on

write_i2c(0x4202, 0x00);

4.5.2 MIPI Stream off

write_i2c(0x4202, 0x0f);

4.6 YUV Sequence

The YUV sequence of OV5640 output image should match with baseband chip/ISP to get correct image.

0x4300[0:1] control YUV sequence

```
//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);
```

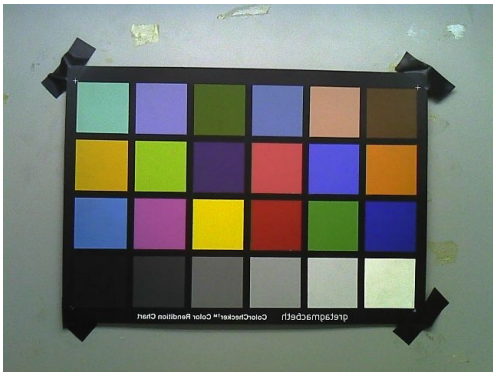
4.7 Mirror and Flip

Since OV5640 is a BSI sensor, the light go to sensor from backside of sensor. So the image without mirror/flip looks mirror. The image with mirror looks normal.

i2c_salve_Address = 0x78;

MIRROR (default)

```
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

```
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);
```

MIRROR&FLIP

```
reg3820 = read_i2c(0x3820);
```

MIRROR&FLIP

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

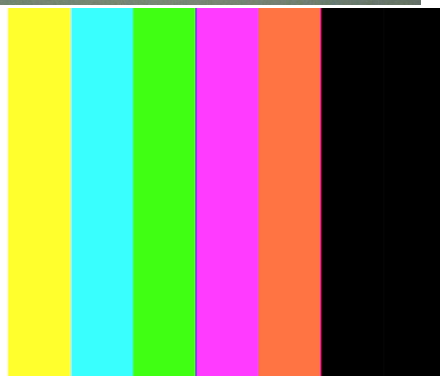
Normal

```
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);
```

4.8 Test Pattern

Color bar

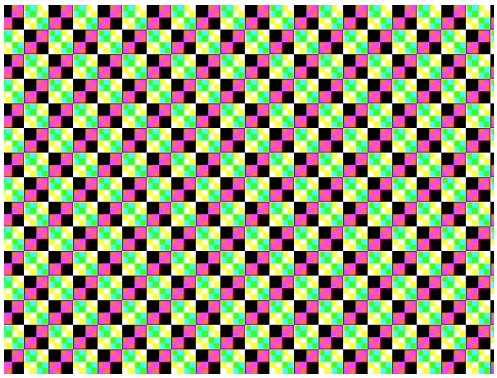
```
write_i2c(0x503d, 0x80);
write_i2c(0x4741, 0x00);
```



Color bar

Color square

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



Color square

4.9 Remove Light Band

OV5640_set_bandingfilter() function set banding filters automatically in camera driver.

Light band is removed by set exposure to $\frac{n}{100}$ ($\frac{n}{120}$ for 60Hz) seconds. The banding filter value tell OV5640 how many lines is $\frac{n}{100}$ ($\frac{n}{120}$ for 60Hz) seconds.

$$Banding_filter_50hz = \frac{\frac{1}{100}}{line_period} = \frac{\frac{1}{100}}{HTS \times sysclk} = \frac{sysclk}{HTS \times 100}$$

$$Banding_filter_60hz = \frac{\frac{1}{120}}{line_period} = \frac{\frac{1}{120}}{HTS \times sysclk} = \frac{sysclk}{HTS \times 120}$$

HTS = register {0x380c, 0x380d}

sysclk could be calculated by the function OV5640_get_sysclk().

4.10 User Interface Functions

4.10.1 Brightness

Brightness +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
```

Brightness +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
```

Brightness +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
```

Brightness +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

Default Brightness
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

Brightness -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

Brightness -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

Brightness -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

Brightness -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 Contrast

Contrast +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

Contrast +2
write_i2c(0x3212, 0x03); // start group 3

ny Confidential

```
write_i2c(0x5586, 0x28);
write_i2c(0x5585, 0x18);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3
```

Contrast +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
```

Contrast Standard

```
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
```

Contrast -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
```

Contrast -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
```

Contrast -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 Saturation

Saturation +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
```

Saturation +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
```

Saturation +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
```

Saturation Standard

```
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
```

Saturation -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
```

Saturation -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
```

Saturation -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

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);

EV Standard
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 Light Mode

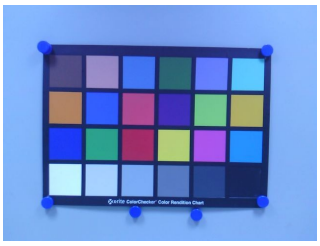
Auto

```
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

```
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



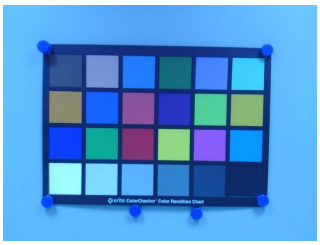
```
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x3406, 0x01);
write_i2c(0x3400, 0x05);
write_i2c(0x3401, 0x48);
write_i2c(0x3402, 0x04);
write_i2c(0x3403, 0x00);
write_i2c(0x3404, 0x07);
write_i2c(0x3405, 0xcf);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // lanuch group 3
```

Cloudy

```
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

```
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
```



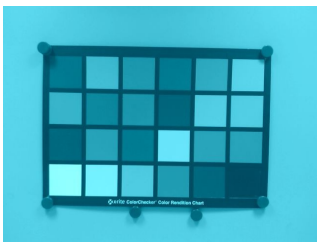
4.10.6 Special Effects

Normal (off)

```
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)

```
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
```



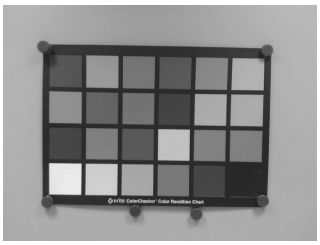
Redish (warm)

```
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

```
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

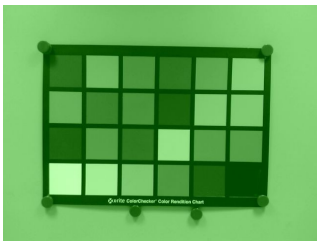
```
write_i2c(0x3212, 0x03); // start group 3
write_i2c(0x5580, 0x1e);
write_i2c(0x5583, 0x40);
write_i2c(0x5584, 0xa0);
write_i2c(0x5003, 0x08);
write_i2c(0x3212, 0x13); // end group 3
write_i2c(0x3212, 0xa3); // launch group 3
```

Negative

```
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

```
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
```

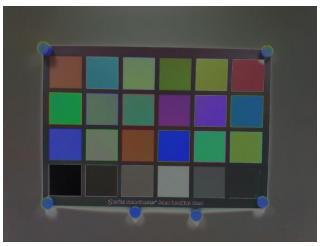


Overexposure

```
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

```
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 Night Mode

When night mode is turned on, dummy lines is inserted automatically and the frame rate is decreased. The total dummy line inserted in controlled by register 0x3a02, 0x3a03 for 50hz light and 0x3a14, 0x3a15 for 60hz light. The night mode on/off switch is controlled by register 0x3a00 bit[2].

Night Mode On

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

Night Mode Off

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

4.10.8 Banding Filter Selection

The camera driver of OV5640 supports banding filter calculation. So there is no need to set banding filter manually. There are 4 options for banding filter, they are

Off

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

Manual 50Hz

```
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

```
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

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

The auto detection function depends on input clock. If input clock is not 24Mhz, please contact with OmniVision local FAE for setting update.

4.11 Auto Focus

4.11.1 Embedded Auto Focus Solution

The auto focus function of OV5640 is controlled by built-in micro-controller, and the VCM driver is also built in OV5640 sensor. The firmware of micro-controller is downloaded from host. After firmware is running, the built-in micro-controller of OV5647 read auto focus information from sensor, calculate the focus position and driver VCM to the position. The host controls the function of built-in micro-controller by I2C commends.

4.11.2 I2C Commands for Auto Focus

Register name	Address	Description	Value
CMD_MAIN	0x3022	Auto focus main commend register	0x03 - Trig Single Auto Focus 0x06 - Pause Auto Focus 0x08 - Release Focus 0x12 - Re-launch Focus Zones 0x00 – command is finished
CMD_ACK	0x3023	ACK of commend	0x00 - Commend is finished 0x01 - Commend is running
FW_STATUS	0x3029	Status of focus	0x7F - S_FIRWARE Firmware is downloaded and not run. Cause 1 MCU is off Cause 2 Firmware is not correct. 0x7E - S_STARTUP Firmware is initializing. 0x70 - S_IDLEIdle state, focus is released. lens is located at the furthest position. 0x00 - S_FOCUSING Auto Focus is running. 0x10 S_FOCUSED Auto Focus is completed.

Note: The MCU will auto clear CMD_MAIN to zero after the command is receipt, and auto clear CMD_ACK to zero when the command is completed.

4.11.3 AF Sequence

There is registers read and written by using OV5640 auto focus firmware sequence,

The auto focus steps as below:

1. Download firmware

   when enter preview for the first time

2. Auto focus

   when capture

3. Release focus

   when finished capture and return to prreview

4.11.4 Download firmware

After initial OV5640, the AF firmware can be download. It is same as download the initial setting. To speedup firmware download, I2c multiple bytes write is highly recommended.

After download the firmware, please check the registers below:

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

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

4.11.5 Auto focus

Auto focus should be completed before capture sequence started.

1. Write 0x03 to register 0x3022 to start single focus
2. read back register 0x3029, if the read back value is 0x10, then focus is finished.
3. Write 0x06 to register 0x3022 to pause auto focus. Lens will stay in current focus position.

4.11.6 Release Focus

After capture , Write 0x08 to register 0x3022 to release the position of lens, let the lens move back infinite.

4.12 Capture Sequence

4.12.1 Shutter

The shutter of OV5640 controls exposure time. The unit of shutter is line period. Shutter value has limitation for each resolution. The maximum shutter value is VTS(0x380e, 0x380f) - 4.

Shutter register value is 16 multiple exposure lines. The shutter value are stored in 3 registers, reg0x3500, reg0x3501,reg3502 .

Shutter = (reg0x3500<<12) + (reg0x3501<<4) + (reg3502>>4);

4.12.2 Gain

Gain is stored in reg0x350a and Reg0x350b.

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

4.12.3 Dummy Lines and Dummy Pixels

If enable auto-night mode of OV5640, the dummy lines are automatic inserted. The number of dummy lines are controled by register {0x3a02[3:0], 0x3a03} for 60Hz and {0x3a14[3:0], 0x3a15} for 50Hz.

Dummy pixels are used to adjust timing. Please adjust HTS register {0x380c, 0x380d} to insert dummy pixels.

4.12.4 Capture Sequence

4.12.4.1 Auto Focus

Trigger single auto focus

Pause auto focus (hold lens at focused position)

4.12.4.2 Read Preview Registers

Read preview_shutter
Read preview_gain
Read preview_HTS
Read preview_sysclk
Read preview_bining_factor B

4.12.4.3 Change Resolution to Capture

Download Capture setting. Turn off AGC/AEC.

4.12.4.4 Read Capture Register Values

Read capture _HTS
Read capture _VTS
Read capture_banding_filter
Read capture SYSCLK

4.12.4.5 Preview Gain/Exposure to Capture/Gain Exposure

The shutter of capture should have same time as preview.

$$capture_shutter = preview_shutter \times \frac{capture_PCLK}{preview_PCLK} \times \frac{preview_HTS}{capture_HTS} \times B$$

$$capture_gain = preview_gain$$

Where B is binning factor. B=1 for OV5640.

Under strong light, when preview_shutter = 1, after calculation capture_shutter=0 due to error of integer calculation. To avoid such issue, we may calculate like this

$$capture_shutter = preview_shutter$$

$$capture_gain = preview_gain \times \frac{capture_PCLK}{preview_PCLK} \times \frac{preview_HTS}{capture_HTS} \times B$$

4.12.4.6 Gain to Exposure and Capture Banding Filter

```
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 Write gain/exposure value

Write capture_gain
Write capture_shutter

4.12.4.8 Capture

Wait for 2 Vsync
Capture the 3rd frame.

4.12.4.9 Back to Preview

// release auto focus, release lens back to infinite.

//Write Registers, Change to preview

//Start AG/AE

write_i2c(0x3503, 0);

4.13 Scale and Zoom

To use scale/zoom function, the ISP scale 0x5001[5] must be set to '1'. To make sure zoom/scale changes smoothly, I2c group write is highly recommended.

4.13.1 Scale

OV5640 ISP support down scale. Images could be scaled down to any size less than 2592x1944. If the aspect ratio of output image is same as sensor image (4:3), adjust register DVPHO and DVPVO to get the desired image size. For example, to get 2048x1536 output

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
```

If the aspect ratio of output image is different as sensor image (not 4:3), then need to adjust X offset and Y offset to clip input image to same aspect ratio as output image. For example, to get 1280x1024 output

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

For any image output size less than 2592x1944, increase X offset and Y offset in such way that the aspect ratio of input image match with aspect ratio of output image, digital zoom function is implemented by this way. For example, when output image is 1600x1200

Digital Zoom 1x

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

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
```

Appendix I Sample Code of Camera Driver for 2 Lane MIPI

```
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;

        }
}
```

Revision History

Rev 2.01

Merge from “OV5640 Camera Module Software Application Notes” and “OV5640 Camera Module Hardware Application Notes”.

Rev 2.10

Add auto focus support.

Rev 2.11

Update contrast setting(as DB version A10/AM10).

Add 4.11 Scale.

Update demo driver. Move driver code from “Form1.h” to “PantherDemo.cpp”

R 2.12

Fixed capture exposure bug in driver code.

Add “write_i2c(0x4005, 0x1a); // BLC always update” in initial setting.

Change from DVP interface to MIPI interface.

Update initial setting

R2.13

Add “4.3 Drive Capability” and “4.4 I/O control”

R2.14

Update “4.4 I/O Control”.

R2.15

Add Bit rate in setting