# OV3640 Camera Module

# Software Application Notes



Last Modified: Mar. 21th, 2008

Document Revision: 1.70

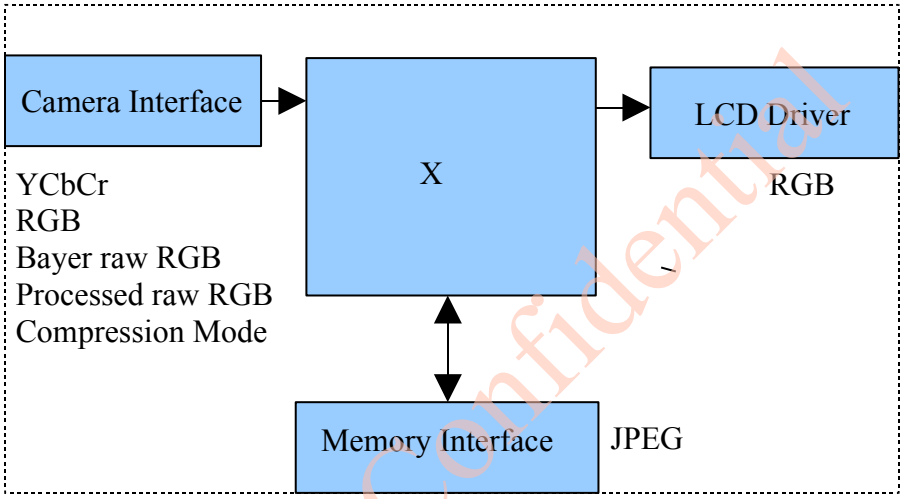
# Table of Contents

**NOTE: OV3640 use 16bits Register address and 8bits Register data.**

## *1. How to Select Output format?*

OV3640 support 5 output format: YcbCr422,YCbCr420, RGB565, Bayer raw RGB, YUV422 JPEG. How to choose the right output format for camera phone design or other applications? Let's look at the back-end chip first.

The general diagram of back-end chip is as below:




The data format at LCD driver are always RGB. For example, RGB444, RGB565, RGB555, RGB888 etc. The data format and memory interface are always JPEG. The JPEG data is compressed from YCbCr data. So Both RGB and YCbCr data are needed inside the back-end chip. The "X" block is different for different back-end chips.



### 1.1 Back-end with full ISP

This kind of back-end has full ISP. It takes raw RGB input, doing interpolation to generate RGB24 and doing translation to generate YCbCr. This kind of back-end could take Bayer raw RGB or processed raw RGB.

The advantage of processed raw RGB over Bayer raw RGB is the output data are processed. Sensor functions such as defect pixel correction, lens correction, gamma, color matrix, de-noise, sharpness, BLC etc. could be applied. Since the life time of back-end chip is longer than image sensor, sometimes back-end chips could not fix defects of new sensors if taken Bayer raw RGB. But the defects of new sensors could be fixed in processed raw RGB output.

If back-end take Bayer raw RGB format from sensor, all the image process operations such as defect pixel correction, lens correction, gamma, color matrix, de-noise, sharpness, BCL etc should

be done by back-end. If back-end take processed raw RGB format from sensor, the image process operations such as defect pixel correction, lens correction, gamma, color matrix, de-noise, sharpness, BCL etc could be done either inside sensor or by back-end chips. In other words, user could select the image process operation be done by which side.

## 1.2 Back-end with YCbCr ISP

This kind of back-end has ISP, but could take only YCbCr format. The ISP could convert YCbCr to RGB format for LCD display and compress YCbCr to JPEG for storage.

## 1.3 Back-end without ISP

This kind of back-end doesn't have ISP built-in. It can not convert from one format to another by hardware. Actually the format conversion is done by software. There are 3 possible solution for this kind of back-end chips.

a. Sensor output YCbCr. back-end chip convert YCbCr to RGB for display by software.
b. Sensor output RGB565 . Back-end chip convert RGB565 to YCbCr for JPEG compression.
c. Sensor output RGB565 for preview, output YCbCr for capture (JPEG compression).

Solution a. provide the best picture quality. Since the input data is 24-bit RGB equivalent. It could converted to RGB888 for LCD display. Solution b. provide the worst picture quality. Since the input data is only 16-bit RGB565, even it is converted to YCbCr, the color depth is still 16-bit. The solution c. provide similar picture quality as solution a. But since preview is RGB565, capture is YCbCr, preview picture may looks a little different than captured picture.



## 1.4 Equations to Convert from One Format to Another

YCbCr to RGB24

$Y = 0.299R + 0.587G + 0.114B$

$Cb = 0.568(B-Y) + 128 = -0.172R - 0.339G + 0.511B + 128$

$Cr = 0.713(R-Y) + 128 = 0.511R - 0.428G - 0.083B + 128$

```
Y = ((77 * R + 150 * G + 29 * B) >> 8);
Cb = ((-43 * R - 85 * G + 128 * B) >> 8) + 128;
Cr = ((128 * R - 107 * G - 21 * B) >> 8) + 128;
```

RGB24 to YCbCr

$R = Y + 1.371(Cr - 128)$

G = Y – 0.698(Cr – 128) – 0.336(Cb – 128)

B = Y + 1.732(Cb – 128)

R = Y + (351*(Cr – 128)) >> 8

G = Y – (179*(Cr – 128) + 86*(Cb – 128))>>8

B = Y + (443*(Cb – 128)) >> 8

## *2. How to Select Output Resolution?*

### 2.1 back-end with ISP

If back-end chip has built-in ISP (Full ISP or YCbCr ISP), the ISP could do image scale. So OV3640 outputs only XGA format for preview and QXGA for capture. ISP scaled XGA image to other resolution that mobile device needed for LCD display. And the ISP scaled QXGA image to other resolution that the mobile device needed for capture.

### 2.2 back-end without ISP

If back-end chip doesn't have image scale capability, then the LCD scaler of OV3640 must be used to scale output resolution exactly the LCD size. For example, if the LCD size is 176x220, then the LCD scaler will scale the output size to 176x220.

In this case, OV3640 output small resolution for preview, and several other resolution for capture. The resolution for capture may include: QQVGA, QVGA, QCIF, CIF, VGA, SVGA,SXGA,UXGA,QXGA. For best quality, all capture size are all downscaled from QXGA.

## *3. How to Adjust frame rate*

The recommended frame rates is 15fps preview for 60/50Hz light environment. The recommended frame rate for capture is 7.5fps for 60/50hz light environment. The frame rate for night mode is lower, we'll discuss night mode later.

Reference settings for above frame rates are listed below. VGA is DCW from QXGA.

### 3.1 VGA Preview, 15fps, 24 Mhz input clock,22Mhz PCLK

```
i2c_salve_Address = 0x78;
write_i2c(0x300e, 0x32);
write_i2c(0x3011, 0x00);
write_i2c(0x3010, 0x20);
write_i2c(0x300f, 0x21);
write_i2c(0x304c, 0x85);
write_i2c(0x302a, 0x06);
write_i2c(0x302b, 0x20);
write_i2c(0x302c, 0x00);
write_i2c(0x3014, 0x04);
```

write_i2c(0x302e, 0x00);
write_i2c(0x302d, 0x00);

### 3.2 QXGA Capture, 7.5fps, 24 Mhz input clock,56Mhz PCLK

i2c_salve_Address = 0x78;
write_i2c(0x300e, 0x39);
write_i2c(0x3011, 0x00);
write_i2c(0x3010, 0x20);
write_i2c(0x300f, 0x21);
write_i2c(0x304c, 0x81);
write_i2c(0x302a, 0x06);
write_i2c(0x302b, 0x20);
write_i2c(0x302c, 0x00);
write_i2c(0x3014, 0x04);
write_i2c(0x302e, 0x00);
write_i2c(0x302d, 0x00);

## *4. How to set Night Mode Preview*

There are 2 types of settings for night mode. One type is set to fixed low frame rate, for example 3.75fps. The other type is set to auto frame rate, for example from 15fps to 3.75fps. When environment is bright, the frame rate is increased to 15fps. When environment is dark, the frame rate is decreased to 3.75fps.

### 4.1 Night Mode VGA Preview with Fixed Frame Rate

3.75fps night mode for 60/50Hz light environment, 24Mhz clock input,6Mhz PCLK
i2c_salve_Address = 0x78;
write_i2c(0x300e, 0x32);
write_i2c(0x3011, 0x03);
write_i2c(0x3010, 0x20);
write_i2c(0x300f, 0x21);
write_i2c(0x304c, 0x84);
write_i2c(0x302a, 0x06);
write_i2c(0x302b, 0x20);
write_i2c(0x302c, 0x00);
write_i2c(0x3014, 0x04);
write_i2c(0x302e, 0x00);
write_i2c(0x302d, 0x00);

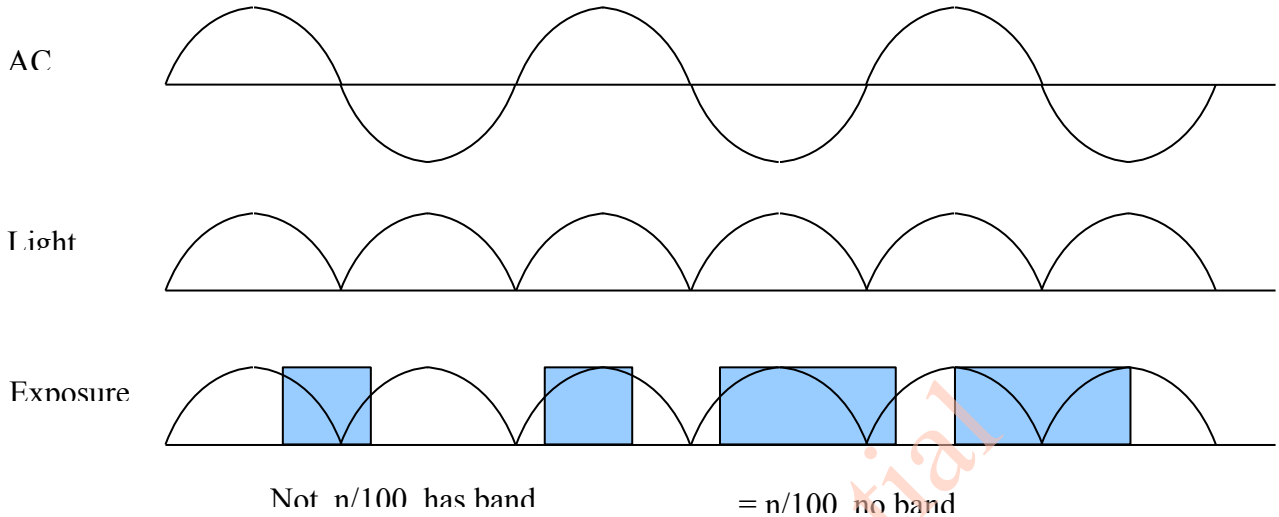
### 4.2 Night Mode VGA preview with Auto Frame Rate

15fps ~ 3.75fps night mode for 60/50Hz light environment, 24Mhz clock input,22Mhz PCLK
i2c_salve_Address = 0x78;
write_i2c(0x300e, 0x32);
write_i2c(0x3011, 0x00);

```
write_i2c(0x3010, 0x20);
write_i2c(0x300f, 0x21);
write_i2c(0x304c, 0x85);
write_i2c(0x302a, 0x06);
write_i2c(0x302b, 0x20);
write_i2c(0x302c, 0x00);
write_i2c(0x3014, 0x0c);
write_i2c(0x302e, 0x00);
write_i2c(0x302d, 0x00);
write_i2c(0x3015, 0x42);
```

# 5. How to Remove Light Band in Preview Mode

## 5.1 Light Band




The strength of office light is not even. It changes with AC frequency. For example, if the AC frequency is 50Hz, the light changes strength at 100hz.



## 5.2 Remove Light band

Light band is removed by set exposure to n/100 (n/120 for 60Hz)seconds. The banding filter value tell OV3640 how many lines is 1/100 (1/120 for 60Hz) seconds.

## 5.3 Select Banding Filter by Region Information

The region information of mobile phone could be used to select banding filter values. A light frequency table is built to indicate which region uses 50Hz light and which region uses 60Hz light. When region information is got, the light frequency information could be get from the table.

Different frame rate could be used for different light frequency. So the frame rate is optimized for both 50hz light condition and 60hz light condition.

Banding filter setting for 15fps VGA preview, 24Mhz input clock
i2c_salve_Address = 0x78;
write_i2c(0x3014, 0x04);  bit[7]select 50/60hz banding, 0:60hz
write_i2c(0x3013, 0xe7); bit[5] banding filter selection on/off
write_i2c(0x3070, 0x00); //50Hz banding filter 8 MSB
write_i2c(0x3071, 0xeb); //50Hz banding filter value 8 LSB
write_i2c(0x3072, 0x00); //60Hz banding filter value 8MSB
write_i2c(0x3073, 0xc4); //60Hz banding filter value 8 LSB
write_i2c(0x301c, 0x05); 50Hz maximum banding step
write_i2c(0x301d, 0x06);  60Hz maximum banding step

## 5.4 Select Banding Filter by Automatic Light Frequency Detection

Set same frame rate for 50Hz and 60Hz light environment, set 50Hz and 60Hz banding filter value. OV3640 could automatic select 50Hz or 60Hz banding filter based on light frequency detection.

QXGA and any size DCW from QXGA
i2c_salve_Address = 0x78;
write_i2c(0x304d, 0x45);
write_i2c(0x30d7, 0x90);
write_i2c(0x302c, 0x00);
write_i2c(0x302d, 0x00);
write_i2c(0x302e, 0x00);
write_i2c(0x3014, 0x44);
write_i2c(0x3048, 0x1f);
write_i2c(0x3049, 0x4e);
write_i2c(0x304a, 0x20);
write_i2c(0x304b, 0x00);
write_i2c(0x304c, 0x82);
write_i2c(0x30a5, 0x03);
write_i2c(0x30ab, 0x03);

write_i2c(0x3070, 0xXX);Based on different frame rate
write_i2c(0x3071, 0xXX);
write_i2c(0x3072, 0xXX);
write_i2c(0x3073, 0xXX);

XGA and any size DCW from XGA
i2c_salve_Address = 0x78;
write_i2c(0x304d, 0x45);
write_i2c(0x30d7, 0x90);
write_i2c(0x302c, 0x00);
write_i2c(0x302d, 0x00);

```
write_i2c(0x302e, 0x00);
write_i2c(0x3014, 0x44);
write_i2c(0x3048, 0x1f);
write_i2c(0x3049, 0x4e);
write_i2c(0x304a, 0x20);
write_i2c(0x304b, 0x00);
write_i2c(0x304c, 0x82);
write_i2c(0x30a5, 0x03);
write_i2c(0x30ab, 0x03);

write_i2c(0x3070, 0xXX);Based on different frame rate
write_i2c(0x3071, 0xXX);
write_i2c(0x3072, 0xXX);
write_i2c(0x3073, 0xXX);
```

## 5.5 Remove Light Band In Capture

Refer to 14.

## 5.6 When Light Band can not be Removed

Normally the light band is removed by banding filter.

But there is some special conditions such as mix light of sun light and office light, take picture of florescent light, the light band can not removed. The reason is the exposure time is less than 1/100 second for 50hz light environment and less than 1/120 second for 60hz light environment, so the light band can not be removed.

The light band in this conditions could not be removed for all CMOS sensors, not only OV3640. So there is no way to remove light band in this condition.

# *6. White Balance*

OV3640 support simple white balance and advanced white balance.

## 6.1 Simple White Balance

Simple white balance assume “gray world”. Which means the average color of world is gray. It is true for most environment.

Advantage of simple AWB

Simple white balance is not depend on lens. A general setting for simple white balance could applied for all modules with different lens.

Disadvantage of simple AWB

The color is not accurate in conditions where “gray world” not true. For example the background has a huge red, blue or green etc. the color of the foreground is not accurate. If the camera target

single color such as red, blue, green, the simple white balance will make the single color gray.

Settings
i2c_salve_Address = 0x78;
write_i2c(0x3308, 0xa5); // Simple AWB

## 6.2 Advanced White Balance

Advanced white balance uses color temperature information to detect white area and do the white balance.

Advantage of Advanced AWB

Color is more accurate than simple white balance. Even the background is single color, the camera will not make the single color gray.

Disadvantage of Advanced AWB

Advanced white balance setting is depend on lens. The setting must be adjusted for every module with new lens. The adjustment must be done by OmniVision FAE in optical lab with some optical equipment such as light box, color checker etc.

Settings

Contact with OmniVision local FAE.

## 6.3 How to select?

Generally, for low resolution camera module such as CIF, VGA and 1.3M, simple AWB is selected. For high resolution camera module such as 2M, 3M, advanced AWB is selected.

# *7. Defect Pixel Correction*

Defect pixel includes dead pixel and wounded pixel.

Dead pixel include white dead pixel and black dead pixel. White dead pixel is always white no matter the actual picture is bright or dark. Black dead pixel is always black no matter the actual picture is bright or dark.

Wounded pixel may change with light, but not as much as normal pixel. White wounded pixels are much brighter then normal pixels, but not complete white. Black wounded pixels are much darker than normal pixels, but not complete black.

OV3640 has built-in defect pixel correction function. If OV3640 output YCbCr, RGB565, Processed raw RGB, the defect pixel correction function could be enabled to fix defect pixels. But if Bayer raw RGB is used, the defect pixel correction function of sensor could not be used. The defect pixel correction of back-end chip should be used instead.

Please pay attention to the defect pixel correction function of back-end chip. Some back-end chip may not be able to correct all defect pixels of OV3640.

Settings
i2c_salve_Address = 0x78;
write_i2c(0x3301, 0xde); // Pixel Correction ON,bit[2:1]: 11,select enable

## 8. BLC

The function of Black Level Calibration (BLC) is to product accurate color in the dark area of picture. There is automatic BLC function built-in OV3640. It should always be turned on.

## 9. Video Mode

Video mode need high frame rate, usually fixed 15fps. There is no night mode for video mode.

## 10. Digital zoom

If OV3640 output image smaller than XGA, it may support continuous digital zoom. For example

| | |
|---|---|
| QXGA | no digital zoom supported |
| XGA | 1-2x |
| VGA | 1-3.2x |
| QVGA | 1-6.4x |

If back-end chip support scale up, then more zoom level could be supported.

## 11. OV3640 Functions

### 11.1 Light Mode

Auto
i2c_salve_Address = 0x78;
write_i2c(0x332b, 0x00)//AWB auto, bit[3]:0,auto

Sunny
i2c_salve_Address = 0x78;
write_i2c(0x332b, 0x08); //AWB off
write_i2c(0x33a7, 0x5e);
write_i2c(0x33a8, 0x40);
write_i2c(0x33a9, 0x46);

Cloudy
i2c_salve_Address = 0x78;
write_i2c(0x332b, 0x08);
write_i2c(0x33a7, 0x68);
write_i2c(0x33a8, 0x40);
write_i2c(0x33a9, 0x4e);

Office
i2c_salve_Address = 0x78;
write_i2c(0x332b, 0x08);
write_i2c(0x33a7, 0x52);
write_i2c(0x33a8, 0x40);
write_i2c(0x33a9, 0x58);

Home

```
i2c_salve_Address = 0x78;
write_i2c(0x332b, 0x08);
write_i2c(0x33a7, 0x44);
write_i2c(0x33a8, 0x40);
write_i2c(0x33a9, 0x70);
```

## 11.2 Color Saturation

The color saturation of OV3640 could be adjusted. High color saturation would make the picture looks more vivid, but the side effect is the bigger noise and not accurate skin color.

```
Saturation + 2(1.75x)
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);bit[7]:1, enable SDE
write_i2c(0x3355, 0x02); enable color saturation
write_i2c(0x3358, 0x70);
write_i2c(0x3359, 0x70);

Saturation + 1(1.25x)
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x02);
write_i2c(0x3358, 0x50);
write_i2c(0x3359, 0x50);

Saturation 0
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x02);
write_i2c(0x3358, 0x40);
write_i2c(0x3359, 0x40);

Saturation -1(0.75x)
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x02);
write_i2c(0x3358, 0x30);
write_i2c(0x3359, 0x30);

Saturation – 2(0.5x)
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x02);
write_i2c(0x3358, 0x20);
write_i2c(0x3359, 0x20);
```

## 11.3 Brightness

The brightness of OV3640 could be adjusted. Higher brightness will make the picture more bright. The side effect of higher brightness is the picture looks foggy.

Brightness +3
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x04);  bit[2] enable
write_i2c(0x3354, 0x01);  bit[3] sign of brightness
write_i2c(0x335e, 0x30);

Brightness +2
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x04);
write_i2c(0x3354, 0x01);
write_i2c(0x335e, 0x20);

Brightness +1
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x04);
write_i2c(0x3354, 0x01);
write_i2c(0x335e, 0x10);

Brightness 0
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x04);
write_i2c(0x3354, 0x01);
write_i2c(0x335e, 0x00);

Brightness -1
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x04);
write_i2c(0x3354, 0x09);
write_i2c(0x335e, 0x10);

Brightness -2
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x04);
write_i2c(0x3354, 0x09);
write_i2c(0x335e, 0x20);

Brightness -3
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);

write_i2c(0x3355, 0x04);
write_i2c(0x3354, 0x09);
write_i2c(0x335e, 0x30);

## 11.4 Contrast

The contrast of OV3640 could be adjusted. Higher contrast will make the picture sharp. But the side effect is loosing dynamic range.

Contrast +3
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x04);  bit[2] enable contrast/brightness
write_i2c(0x3354, 0x01); bit[2] Yoffset sign
write_i2c(0x335c, 0x2c);
write_i2c(0x335d, 0x2c);

Contrast +2
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x04);
write_i2c(0x3354, 0x01);
write_i2c(0x335c, 0x28);
write_i2c(0x335d, 0x28);

Contrast +1
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x04);
write_i2c(0x3354, 0x01);
write_i2c(0x335c, 0x24);
write_i2c(0x335d, 0x24);

Contrast 0
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x04);
write_i2c(0x3354, 0x01);
write_i2c(0x335c, 0x20);
write_i2c(0x335d, 0x20);

Contrast -1
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x04);
write_i2c(0x3354, 0x01);
write_i2c(0x335c, 0x1c);
write_i2c(0x335d, 0x1c);

Contrast -2
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x04);
write_i2c(0x3354, 0x01);
write_i2c(0x335c, 0x18);
write_i2c(0x335d, 0x18);

Contrast -3
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x04);
write_i2c(0x3354, 0x01);
write_i2c(0x335c, 0x14);
write_i2c(0x335d, 0x14);

## 11.5 Special effects

OV3640 support some special effects such as B/W, negative, sepia, bluish, reddish, greenish,negative, etc. If users need other special effects, it should be supported by back-end chips.

Sepia(antique)
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x18);
write_i2c(0x335a, 0x40);
write_i2c(0x335b, 0xa6);

Bluish
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x18);
write_i2c(0x335a, 0xa0);
write_i2c(0x335b, 0x40);

Greenish
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x18);
write_i2c(0x335a, 0x60);
write_i2c(0x335b, 0x60);

Reddish
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x18);
write_i2c(0x335a, 0x80);
write_i2c(0x335b, 0xc0);

Yellowish
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x18);
write_i2c(0x335a, 0x30);
write_i2c(0x335b, 0x90);

B&W
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x18);bit[4]fix u enable, bit[3]fix v enable
write_i2c(0x335a, 0x80);
write_i2c(0x335b, 0x80);

Negative
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x40);bit[6] negative

Normal
i2c_salve_Address = 0x78;
write_i2c(0x3302, 0xef);
write_i2c(0x3355, 0x00);

## 11.6 Exposure

OV3640 support different exposure level. It can increase/decrease target brightness by change exposure/gain auto. OV3640 support two exposure algorithm, one is Average, other is Histogram.

### 11.6.1 Average-based Algorithm

**---Based on Target luminance**

i2c_salve_Address = 0x78;
write_i2c(0x3047,0x00);

-1.7EV
write_i2c(0x3018, 0x10)
write_i2c(0x3019, 0x08)
write_i2c(0x301a, 0x21)

-1.3EV
write_i2c(0x3018, 0x18)
write_i2c(0x3019, 0x10)
write_i2c(0x301a, 0x31)

-1.0EV
write_i2c(0x3018, 0x20)
write_i2c(0x3019, 0x18)

write_i2c(0x301a, 0x41)

-0.7EV
write_i2c(0x3018, 0x28)
write_i2c(0x3019, 0x20)
write_i2c(0x301a, 0x51)

-0.3EV
write_i2c(0x3018, 0x30)
write_i2c(0x3019, 0x28)
write_i2c(0x301a, 0x61)

default
write_i2c(0x3018, 0x38)
write_i2c(0x3019, 0x30)
write_i2c(0x301a, 0x61)

0.3EV
write_i2c(0x3018, 0x40)
write_i2c(0x3019, 0x38)
write_i2c(0x301a, 0x71)

0.7EV
write_i2c(0x3018, 0x48)
write_i2c(0x3019, 0x40)
write_i2c(0x301a, 0x81)

1.0EV
write_i2c(0x3018, 0x50)
write_i2c(0x3019, 0x48)
write_i2c(0x301a, 0x91)

1.3EV
write_i2c(0x3018, 0x58)
write_i2c(0x3019, 0x50)
write_i2c(0x301a, 0x91)

1.7EV
write_i2c(0x3018, 0x60)
write_i2c(0x3019, 0x58)
write_i2c(0x301a, 0xa1)

### 11.6.2 Histogram-based Algorithm

**---Based on Histogram and probability**.

i2c_salve_Address = 0x78;
write_i2c(0x3047,0x80);

-1.7EV
write_i2c(0x3018, 0x58)
write_i2c(0x3019, 0x38)

-1.3EV
write_i2c(0x3018, 0x60)
write_i2c(0x3019, 0x40)

-1.0EV
write_i2c(0x3018, 0x68)
write_i2c(0x3019, 0x48)

-0.7EV
write_i2c(0x3018, 0x70)
write_i2c(0x3019, 0x50)

-0.3EV
write_i2c(0x3018, 0x78)
write_i2c(0x3019, 0x58)

default
write_i2c(0x3018, 0x80)
write_i2c(0x3019, 0x60)

0.3EV
write_i2c(0x3018, 0x88)
write_i2c(0x3019, 0x68)

0.7EV
write_i2c(0x3018, 0x90)
write_i2c(0x3019, 0x70)

1.0EV
write_i2c(0x3018, 0x98)
write_i2c(0x3019, 0x78)

1.3EV
write_i2c(0x3018, 0xa0)
write_i2c(0x3019, 0x80)

1.7EV
write_i2c(0x3018, 0xa8)
write_i2c(0x3019, 0x88)

## 11.7 Sharpness

i2c_salve_Address = 0x78;

Sharpness 1
write_i2c(0x332d, 0x41)

Sharpness 2
write_i2c(0x332d, 0x42)

Sharpness 3
write_i2c(0x332d, 0x43)

Sharpness 4
write_i2c(0x332d, 0x44)

Sharpness 5
write_i2c(0x332d, 0x45)

Sharpness 6
write_i2c(0x332d, 0x46)

Sharpness 7
write_i2c(0x332d, 0x47)

Sharpness 8
write_i2c(0x332d, 0x48)

Sharpness auto
write_i2c(0x332d, 0x60)
write_i2c(0x332f, 0x03)

## 11.8 Mirror/Flip

i2c_salve_Address = 0x78;

MIRROR
write_i2c(0x307c, 0x12);mirror
write_i2c(0x3090, 0xc8);
write_i2c(0x3023, 0x0a);

FLIP
write_i2c(0x307c, 0x11);flip
write_i2c(0x3023, 0x09);
write_i2c(0x3090, 0xc0);

MIRROR&FLIP
write_i2c(0x307c, 0x13);flip/mirror
write_i2c(0x3023, 0x09);
write_i2c(0x3090, 0xc8);

NORML
write_i2c(0x307c, 0x10);no mirror/flip
write_i2c(0x3090, 0xc0);
write_i2c(0x3023, 0x0a);

## 11.9 YUV Sequence

0x3600[0:1] control YUV sequence
Y U Y V
write_i2c(0x3404, 0x00)

Y V Y U
write_i2c(0x3404, 0x01)

V Y U Y
write_i2c(0x3404, 0x03)

U Y V Y
write_i2c(0x3404, 0x02)

## 11.10 Clock Polarity

Data valid VSYNC high
0x3600[2] Control VSYNC polarity
1: Data valid VSYNC High
0: Data valid VSYNC low

0x3600[4] Control PCLK polarity
1: Data update at Falling-edge
0: Data update at Rising-edge

0x3600[3] Control HREF polarity
0:Data valid HREF high
1: Data valid HREF Low



# *12. Deal with Lens*

## 12.1 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 OV3640 could be turned on to compensate the corner brightness and make the whole picture looks same bright.
settings
i2c_salve_Address = 0x78;
write_i2c(0x3300,0x13);bit[0]: enable lens correction

## 12.2 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.

## 12.3 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.

For OV3640 the focus distance is about 140~160cm. The depth of field is about from 70~80cm to infinite. If checking resolution of camera module, the resolution chart should be placed 140~160 cm away.

## 12.4 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.

## 12.5 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.

# *13. Reference Settings*

## 13.1 YCbCr Reference Setting

OV3640 output maximum 15fps QXGA and maximum 30fps XGA. For other resolution, only downscale from QXGA or XGA. Say to, resolution from XGA can reach up maximum 30fps, resolution from QXGA can reach up maximum 15fps.

### *13.1.1 VGA Preview*

```
// OV3640_VGA_YUV 15 fps
// 24 MHz input clock, 22Mhz PCLK

write_i2c(0x3012, 0x80);
//delay 5ms
write_i2c(0x304d, 0x45);
write_i2c(0x30a7, 0x5e);
write_i2c(0x3087, 0x16);
write_i2c(0x309c, 0x1a);
write_i2c(0x30a2 0xe4);
write_i2c(0x30aa, 0x42);
write_i2c(0x30b0, 0xff);
write_i2c(0x30b1, 0xff);
```

```
write_i2c(0x30b2, 0x10);
write_i2c(0x300e, 0x32);
write_i2c(0x300f, 0x21);
write_i2c(0x3010, 0x20);
write_i2c(0x3011, 0x04);
write_i2c(0x304c, 0x81);
write_i2c(0x30d7, 0x10);
write_i2c(0x30d9, 0x0d);
write_i2c(0x30db, 0x08);
write_i2c(0x3016, 0x82);
write_i2c(0x3018, 0x38);
write_i2c(0x3019, 0x30);
write_i2c(0x301a, 0x61);
write_i2c(0x307d, 0x00);
write_i2c(0x3087, 0x02);
write_i2c(0x3082, 0x20);
write_i2c(0x3015, 0x12);8x
write_i2c(0x3014, 0x04);
write_i2c(0x3013, 0xf7);

write_i2c(0x303c, 0x08);
write_i2c(0x303d, 0x18);
write_i2c(0x303e, 0x06);
write_i2c(0x303f, 0x0c);
write_i2c(0x3030, 0x62);
write_i2c(0x3031, 0x26);
write_i2c(0x3032, 0xe6);
write_i2c(0x3033, 0x6e);
write_i2c(0x3034, 0xea);
write_i2c(0x3035, 0xae);
write_i2c(0x3036, 0xa6);
write_i2c(0x3037, 0x6a);

write_i2c(0x3104, 0x02);
write_i2c(0x3105, 0xfd);
write_i2c(0x3106, 0x00);
write_i2c(0x3107, 0xff);
write_i2c(0x3300, 0x12);
write_i2c(0x3301, 0xde);
write_i2c(0x3302, 0xef);

write_i2c(0x3316, 0xff);
write_i2c(0x3317, 0x00);
write_i2c(0x3312, 0x26);
write_i2c(0x3314, 0x42);
write_i2c(0x3313, 0x2b);
write_i2c(0x3315, 0x42);
write_i2c(0x3310, 0xd0);
```

```
write_i2c(0x3311, 0xbd);
write_i2c(0x330c, 0x18);
write_i2c(0x330d, 0x18);
write_i2c(0x330e, 0x56);
write_i2c(0x330f, 0x5c);
write_i2c(0x330b, 0x1c);
write_i2c(0x3306, 0x5c);
write_i2c(0x3307, 0x11);

write_i2c(0x336a, 0x52);
write_i2c(0x3370, 0x46);
write_i2c(0x3376, 0x38);
write_i2c(0x3300, 0x13);
write_i2c(0x30b8, 0x20);
write_i2c(0x30b9, 0x17);
write_i2c(0x30ba, 0x04);
write_i2c(0x30bb, 0x08);
write_i2c(0x3507, 0x06);
write_i2c(0x350a, 0x4f);

write_i2c(0x3100, 0x02);
write_i2c(0x3301, 0xde);
write_i2c(0x3304, 0x00);
write_i2c(0x3400, 0x00);
write_i2c(0x3404, 0x00);

write_i2c(0x335f, 0x68);
write_i2c(0x3360, 0x18);
write_i2c(0x3361, 0x0c);
write_i2c(0x3362, 0x12);
write_i2c(0x3363, 0x88);
write_i2c(0x3364, 0xe4);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x02);
write_i2c(0x3089, 0x80);
write_i2c(0x308a, 0x01);
write_i2c(0x308b, 0xe0);

write_i2c(0x308d, 0x04);
write_i2c(0x3086, 0x03);
write_i2c(0x3086, 0x00);

write_i2c(0x3011, 0x00);
write_i2c(0x304c, 0x85);
```

### *13.1.2 Other Preview Resolution DCW from QXGA*

After make VGA preview, add below settings for related size preview:

CIF Preview

```
write_i2c(0x3302, 0xef);
write_i2c(0x335f, 0x68);
write_i2c(0x3360, 0x18);
write_i2c(0x3361, 0x0c);
write_i2c(0x3362, 0x11);
write_i2c(0x3363, 0x68);
write_i2c(0x3364, 0x24);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x01);
write_i2c(0x3089, 0x60);
write_i2c(0x308a, 0x01);
write_i2c(0x308b, 0x20);
```

QXGA->QVGA

```
write_i2c(0x335f, 0x68);
write_i2c(0x3360, 0x18);
write_i2c(0x3361, 0x0c);
write_i2c(0x3362, 0x01);
write_i2c(0x3363, 0x48);
write_i2c(0x3364, 0xf4);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x01);
write_i2c(0x3089, 0x40);
write_i2c(0x308a, 0x00);
write_i2c(0x308b, 0xf0);
```

### *13.1.3 QXGA Capture*

```
; OV3640_QXGA_YUV7.5 fps
; 24 MHz input clock, 56Mhz pclk

write_i2c(0x300e, 0x39);
write_i2c(0x300f, 0x21);
write_i2c(0x3010, 0x20);
write_i2c(0x304c, 0x81);
write_i2c(0x3302, 0xef);

write_i2c(0x335f, 0x68);
write_i2c(0x3360, 0x18);
write_i2c(0x3361, 0x0C);
write_i2c(0x3362, 0x68);
write_i2c(0x3363, 0x08);
write_i2c(0x3364, 0x04);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x08);
write_i2c(0x3089, 0x00);
```

write_i2c(0x308a, 0x06);
write_i2c(0x308b, 0x00);

### *13.1.4 Other Capture size DCW from QXGA*

; 24 MHz input clock, 7.5fps
write_i2c(0x300e, 0x39);
write_i2c(0x300f, 0x21);
write_i2c(0x3010, 0x20);

QXGA->UXGA

write_i2c(0x3302, 0xef);

write_i2c(0x335f, 0x68);
write_i2c(0x3360, 0x18);
write_i2c(0x3361, 0x0C);
write_i2c(0x3362, 0x46);
write_i2c(0x3363, 0x48);
write_i2c(0x3364, 0xb4);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x06);
write_i2c(0x3089, 0x40);
write_i2c(0x308a, 0x04);
write_i2c(0x308b, 0xb0);
write_i2c(0x304c, 0x81);//56Mhz PCLK output

QXGA->SXGA ;1280*960

write_i2c(0x3302, 0xef);

write_i2c(0x335f, 0x68);
write_i2c(0x3360, 0x18);
write_i2c(0x3361, 0x0c);
write_i2c(0x3362, 0x35);
write_i2c(0x3363, 0x08);
write_i2c(0x3364, 0xc4);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x05);
write_i2c(0x3089, 0x00);
write_i2c(0x308a, 0x03);
write_i2c(0x308b, 0xc0);
write_i2c(0x304c, 0x81);//56Mhz PCLK output

QXGA->XGA

write_i2c(0x3302, 0xef);

```
write_i2c(0x335f, 0x68);
write_i2c(0x3360, 0x18);
write_i2c(0x3361, 0x0c);
write_i2c(0x3362, 0x34);
write_i2c(0x3363, 0x08);
write_i2c(0x3364, 0x06);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x04);
write_i2c(0x3089, 0x00);
write_i2c(0x308a, 0x03);
write_i2c(0x308b, 0x00);
write_i2c(0x304c, 0x82);//28Mhz PCLK output
```

QXGA->SVGA

```
write_i2c(0x3302, 0xef);

write_i2c(0x335f, 0x68);
write_i2c(0x3360, 0x18);
write_i2c(0x3361, 0x0c);
write_i2c(0x3362, 0x23);
write_i2c(0x3363, 0x28);
write_i2c(0x3364, 0x5c);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x03);
write_i2c(0x3089, 0x20);
write_i2c(0x308a, 0x02);
write_i2c(0x308b, 0x58);
write_i2c(0x304c, 0x82);//28Mhz PCLK output
```

QXGA->VGA

```
write_i2c(0x3302, 0xef);

write_i2c(0x335f, 0x68);
write_i2c(0x3360, 0x18);
write_i2c(0x3361, 0x0c);
write_i2c(0x3362, 0x12);
write_i2c(0x3363, 0x88);
write_i2c(0x3364, 0xe4);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x02);
write_i2c(0x3089, 0x80);
write_i2c(0x308a, 0x01);
write_i2c(0x308b, 0xe0);
write_i2c(0x304c, 0x84);//14Mhz PCLK output
```

QXGA->CIF

write_i2c(0x3302, 0xef);

write_i2c(0x335f, 0x68);
write_i2c(0x3360, 0x18);
write_i2c(0x3361, 0x0c);
write_i2c(0x3362, 0x11);
write_i2c(0x3363, 0x68);
write_i2c(0x3364, 0x24);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x01);
write_i2c(0x3089, 0x60);
write_i2c(0x308a, 0x01);
write_i2c(0x308b, 0x20);
write_i2c(0x304c, 0x84);//14Mhz PCLK output

QXGA->QVGA

write_i2c(0x3302, 0xef);

write_i2c(0x335f, 0x68);
write_i2c(0x3360, 0x18);
write_i2c(0x3361, 0x0c);
write_i2c(0x3362, 0x01);
write_i2c(0x3363, 0x48);
write_i2c(0x3364, 0xf4);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x01);
write_i2c(0x3089, 0x40);
write_i2c(0x308a, 0x00);
write_i2c(0x308b, 0xf0);
write_i2c(0x304c, 0x84);//14Mhz PCLK output

QXGA->QCIF

write_i2c(0x3302, 0xef);

write_i2c(0x335f, 0x68);
write_i2c(0x3360, 0x18);
write_i2c(0x3361, 0x0c);
write_i2c(0x3362, 0x00);
write_i2c(0x3363, 0xb8);
write_i2c(0x3364, 0x94);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x00);
write_i2c(0x3089, 0xb0);
write_i2c(0x308a, 0x00);

write_i2c(0x308b, 0x90);
write_i2c(0x304c, 0x84);//14Mhz PCLK output

### *13.1.5 Zoom*

There have Zoom function which can auto calculate sensor array size and output size. Below we will introduce two zoom way: one is using zoom function which can support continuous zoom; other is using register settings directly, which is easy for customers who only use 3 level or 4 level zoom.
Image scaling circuit mainly include Windowing(Cropping) , ISP Scaling input , and ISP Output. Image first enter into Windowing circuit which decide horizontal and vertical start point and length of sensor array. Then enter into ISP Scaling input control which decide zoom out size. ISP output decide last output size, say to, what you need when preview and capture.
Windowing circuit is involved in 0x3020—0x3027 registers.
ISP Scaling input is involved in 0x335f---0x3361 registers
ISP Output is involved in 0x3362—0x3364 registers, 0x3088—0x308b registers.

| | | | | |
|---|---|---|---|---|
| 0x3020 | HS[15:8] | 0x01 | RW | Horizontal Window Start 8 MSBs<br>HS[15:0]:Horizontal start point of array, each bit represents 1 pixel |
| 0x3021 | HS[7:0] | 0x0D | RW | Horizontal Window Start 8 LSBs<br>HS[15:0]:Horizontal start point of array, each bit represents 1 pixel |
| 0x3022 | VS[15:8] | 0x00 | RW | Vertical Window Start 8 MSBs<br>VS[15:0]:Vertical start point of array, each bit represents 1 scan line |
| 0x3023 | VS[7:0] | 0x0A | RW | Vertical Window Start 8 LSBs<br>VS[15:0]:Vertical start point of array, each bit represents 1 scan line |
| 0x3024 | HW[15:8] | 0x18 | RW | Horizontal Width 8 MSBs<br>HW[15:0]:Output raw image pixels are from HS[15:0] to HS[15:0] + HW[15:0] |
| 0x3025 | HW[7:0] | 0x00 | RW | Horizontal Width 8 LSBs<br>HW[15:0]:Output raw image pixels are from HS[15:0] to HS[15:0] + HW[15:0] |
| 0x3026 | VH[15:8] | 0x06 | RW | Vertical Height 8 MSBs<br>VH[15:0]:Output raw image pixels are from VS[15:0] to VS[15:0] + VH[15:0] |
| 0x3027 | VH[7:0] | 0x0C | RW | Vertical Height 8 LSBs<br>VH[15:0]:Output raw image pixels are from VS[15:0] to VS[15:0] + VH[15:0] |

| | | | | |
|---|---|---|---|---|
| 0x3088 | ISP_XOUT[15:8] | 0x80 | RW | ISP X-direction Output Size [15:8]<br>Bit[7:4]: Not used<br>Bit[3:0]: X_size_in[11:8] |
| 0x3089 | ISP_XOUT[7:0] | 0x00 | RW | ISP X-direction Output Size [7:0] |
| 0x308A | ISP_YOUT[15:8] | 0x06 | RW | ISP Y-direction Output Size [15:8]<br>Bit[7:3]: Not used<br>Bit[2:0]: X_size_in[10:8] |
| 0x308B | ISP_YOUT[7:0] | 0x00 | RW | ISP Y-direction Output Size [7:0] |

| | | |
|---|---|---|
| 0x335F | SIZE_IN_MISC | SIZE_IN_MISC<br>Bit[6:4]: Vsize_In[10:8] (used with 0x3361[7:0])<br>Bit[3:0]: Hsize_In[11:8] (used with 0x3362[7:0]) |
| 0x3360 | HSIZE_IN_L | HSIZE_IN_L<br>Bit[7:0]: Hsize_In[7:0] (used with 0x335F[3:0]) |
| 0x3361 | VSIZE_IN_L | VSIZE_IN_L<br>Bit[7:0]: Vsize_In[7:0] (used with 0x335F[6:4]) |
| 0x3362 | SIZE_OUT_MISC | SIZE_OUT_MISC<br>Bit[6:4]: Vsize_Out[10:8] (used with 0x3364[7:0])<br>Vsize_Out must be smaller than Vsize_In in 0x335F and 0x3361 due to zoom out function limit.<br>Bit[3:0]: Hsize_Out[11:8] (used with 0x3363[7:0])<br>Hsize_Out must be smaller than Hsize_In in 0x335F and 0x3360 due to zoom out function limit. |
| 0x3363 | HSIZE_OUT_L | HSIZE_OUT_L<br>Bit[7:0]: Hsize_Out[7:0] for zoom out<br>Zoom out output horizontal size low 8-bits in hex (used with 0x3362[3:0]) |
| 0x3364 | VSIZE_OUT_L | VSIZE_OUT_L<br>Bit[7:0]: Vsize_Out[7:0] for zoom out<br>Zoom out output vertical size low 8-bits in hex (used with 0x3362[6:4]) |



### *13.1.5.1 Zoom Function*

```
/*** Digital Continue Zoom Function ***/
static int digital_zoom(unsigned char ucZoomStep, char bZoomIn)
{//calculate sensor array, horizontal and vertical start point of array, scaling input size
        unsigned char ucRegValue;
        unsigned char ucArrayMode;
        unsigned int varH = 8;
        unsigned int varV = 6;
        unsigned int uiVStart;
        unsigned int uiHStart;
        unsigned int uiVWidth;
```

```
unsigned int uiHWidth;
unsigned int uiVArray;
unsigned int uiHArray;
unsigned int uiHSize_In;
unsigned int uiVSize_In;
unsigned int uiHSize_Out;
unsigned int uiVSize_Out;

read_i2c(0x3012, &ucArrayMode);//
ucArrayMode = (ucArrayMode >>4) & 0x07;
switch(ucArrayMode)
{//select sensor output which array size
        case 0:
                varH *=2;
                varV *=2;
                uiVArray = 1548;
                uiHArray = 2072;
                break;
        case 1:
                varH *=2;
                uiVArray = 779;
                uiHArray = 2376;
                break;
        case 7:
        default:
                return -1;
}

varH *= ucZoomStep;
varV *= ucZoomStep;

read_i2c(0x3024, &ucRegValue);
uiHWidth = ucRegValue;
uiHWidth <<=8;
read_i2c(0x3025, &ucRegValue);
uiHWidth += ucRegValue;

read_i2c(0x3026, &ucRegValue);
uiVWidth = ucRegValue;
uiVWidth <<=8;
read_i2c(0x3027, &ucRegValue);
uiVWidth += ucRegValue;


uiHWidth = bZoomIn ? (uiHWidth - 2*varH) : (uiHWidth + 2*varH);
uiVWidth = bZoomIn ? (uiVWidth - 2*varV) : (uiVWidth + 2*varV);

if((uiHWidth > uiHArray)||(uiVWidth>uiVArray))
```

```
{
        return -1;
}

read_i2c(0x335F, &ucRegValue);
uiHSize_In = (ucRegValue & 0x0F);
uiVSize_In = (ucRegValue & 0x70) >> 4;
uiHSize_In <<=8;
uiVSize_In <<=8;
read_i2c(0x3360, &ucRegValue);
uiHSize_In += ucRegValue;
read_i2c(0x3361, &ucRegValue);
uiVSize_In += ucRegValue;

if(0 == ucArrayMode)
{
        uiHSize_In = bZoomIn ? (uiHSize_In - 2*varH) : (uiHSize_In + 2* varH);
}
else
{
        uiHSize_In = bZoomIn ? (uiHSize_In - varH) : (uiHSize_In + varH);
}

uiVSize_In = bZoomIn ? (uiVSize_In - 2*varV) : (uiVSize_In + 2* varV);

if((uiHSize_In > uiHWidth) || (uiVSize_In > uiVWidth))
{
        return -1;
}

read_i2c(0x3362, &ucRegValue);
uiHSize_Out = (ucRegValue & 0x0F);
uiVSize_Out = (ucRegValue & 0x70) >> 4;
uiHSize_Out <<=8;
uiVSize_Out <<=8;
read_i2c(0x3363, &ucRegValue);
uiHSize_Out += ucRegValue;
read_i2c(0x3364, &ucRegValue);
uiVSize_Out += ucRegValue;

if((uiHSize_Out > uiHWidth) || (uiVSize_Out > uiVWidth))
{
        return -1;
}

if((uiHSize_Out > uiHSize_In) || (uiVSize_Out > uiVSize_In))
{
        return -1;
```

```
        }

        read_i2c(0x3020, &ucRegValue);
        uiHStart = ucRegValue;
        uiHStart <<=8;
        read_i2c(0x3021, &ucRegValue);
        uiHStart += ucRegValue;

        read_i2c(0x3022, &ucRegValue);
        uiVStart = ucRegValue;
        uiVStart <<=8;
        read_i2c(0x3023, &ucRegValue);
        uiVStart += ucRegValue;

        uiHStart = bZoomIn ? ( uiHStart + varH ) : (uiHStart - varH);
        uiVStart = bZoomIn ? ( uiVStart + varV) : (uiVStart - varV);

        write_i2c(0x3020, (unsigned char) ((uiHStart>>8)&0xFF));// sensor array start horizontal
register 8MSB
        write_i2c(0x3021, (unsigned char) (uiHStart & 0xFF));// /sensor array start horizontal
register 8LSB
        write_i2c(0x3022, (unsigned char) ((uiVStart>>8)&0xFF));// sensor array start vertical
register 8MSB
        write_i2c(0x3023, (unsigned char) (uiVStart & 0xFF));// sensor array start vertical register
8LSB

        write_i2c(0x3024, (unsigned char) ((uiHWidth>>8)&0xFF));
        write_i2c(0x3025, (unsigned char) (uiHWidth & 0xFF));
        write_i2c(0x3026, (unsigned char) ((uiVWidth>>8)&0xFF));
        write_i2c(0x3027, (unsigned char) (uiVWidth & 0xFF));

        write_i2c(0x335F, (unsigned char)( ((uiHSize_In>>8) & 0x0F) +
((uiVSize_In>>4)&0x70)));
        write_i2c(0x3360, (unsigned char) (uiHSize_In & 0xFF));
        write_i2c(0x3361, (unsigned char) (uiVSize_In & 0xFF));

        return 0;
}

/*** Digital Continue Zoom Function ***/
static void resolution(unsigned int width, unsigned int height)
{// calculate zoom(scaling) out size
        unsigned char ucRegValue;
        unsigned int uiHSize_In;
        unsigned int uiVSize_In;
        unsigned int uiHSize_Out;
        unsigned int uiVSize_Out;
        unsigned int uiX_Out;
```

```
        unsigned int uiY_Out;
        unsigned char ucXStart;
        unsigned char ucYStart;

        read_i2c(0x335F, &ucRegValue);
        uiHSize_In = (ucRegValue & 0x0F);
        uiVSize_In = (ucRegValue & 0x70) >> 4;
        uiHSize_In <<=8;
        uiVSize_In <<=8;
        read_i2c(0x3360, &ucRegValue);
        uiHSize_In += ucRegValue;
        read_i2c(0x3361, &ucRegValue);
        uiVSize_In += ucRegValue;

        read_i2c(0x3403, &ucRegValue);
        ucXStart = (ucRegValue & 0xF0) >> 4;
        ucYStart = ucRegValue & 0x0F;

        uiX_Out = width;
        uiY_Out = height;
        uiHSize_Out = width;
        uiVSize_Out = height;

        if((uiHSize_Out > uiHSize_In) || (uiVSize_Out > uiVSize_In))
        {
                return;
        }

        write_i2c(0x3088,  (unsigned char) ((uiX_Out>>8) & 0x0F) );
        write_i2c(0x3089,  (unsigned char) (uiX_Out & 0xFF) );
        write_i2c(0x308A,  (unsigned char) ((uiY_Out>>8) & 0x07) );
        write_i2c(0x308B,  (unsigned char) (uiY_Out & 0xFF) );

        write_i2c(0x3403,  0x42 );

        write_i2c(0x3362, (unsigned char)( ((uiHSize_Out>>8) & 0x0F) +
((uiVSize_Out>>4)&0x70)));
        write_i2c(0x3363, (unsigned char) (uiHSize_Out & 0xFF));
        write_i2c(0x3364, (unsigned char) (uiVSize_Out & 0xFF));
}
```

#### *13.1.5.2 Zoom Settings*

```
//
//Preview Zoom
//Source size is XGA, preview size all from XGA
```

**1.**

QVGA 1x

```
write_i2c(0x3302, 0xef);
write_i2c(0x335f, 0x34);
write_i2c(0x3360, 0x0c);
write_i2c(0x3361, 0x04);
write_i2c(0x3062, 0x01);
write_i2c(0x3063, 0x48);
write_i2c(0x3064, 0xf4);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x01);
write_i2c(0x3089, 0x40);
write_i2c(0x308a, 0x00);
write_i2c(0x308b, 0xf0);
write_i2c(0x3020, 0x01);
write_i2c(0x3021, 0x1d);
write_i2c(0x3022, 0x00);
write_i2c(0x3023, 0x06);
write_i2c(0x3024, 0x08);
write_i2c(0x3025, 0x18);
write_i2c(0x3026, 0x03);
write_i2c(0x3027, 0x04);
```

QVGA 2x

```
write_i2c(0x3302, 0xef);
write_i2c(0x335f, 0x12);
write_i2c(0x3360, 0x0c);
write_i2c(0x3361, 0x84);
write_i2c(0x3062, 0x01);
write_i2c(0x3063, 0x48);
write_i2c(0x3064, 0xf4);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x01);
write_i2c(0x3089, 0x40);
write_i2c(0x308a, 0x00);
write_i2c(0x308b, 0xf0);
write_i2c(0x3020, 0x03);
write_i2c(0x3021, 0x1d);
write_i2c(0x3022, 0x00);
write_i2c(0x3023, 0xc6);
write_i2c(0x3024, 0x04);
write_i2c(0x3025, 0x18);
write_i2c(0x3026, 0x01);
write_i2c(0x3027, 0x84);
```

QVGA 3x

```
write_i2c(0x3302, 0xef);
write_i2c(0x335f, 0x11);
write_i2c(0x3360, 0x61);
write_i2c(0x3361, 0x04);
```

write_i2c(0x3062, 0x01);
write_i2c(0x3063, 0x48);
write_i2c(0x3064, 0xf4);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x01);
write_i2c(0x3089, 0x40);
write_i2c(0x308a, 0x00);
write_i2c(0x308b, 0xf0);
write_i2c(0x3020, 0x03);
write_i2c(0x3021, 0xc8);
write_i2c(0x3022, 0x01);
write_i2c(0x3023, 0x06);
write_i2c(0x3024, 0x02);
write_i2c(0x3025, 0xc2);
write_i2c(0x3026, 0x01);
write_i2c(0x3027, 0x04);

**2.**
QCIF 1x
write_i2c(0x3302, 0xef);
write_i2c(0x335f, 0x34);
write_i2c(0x3360, 0x0c);
write_i2c(0x3361, 0x04);
write_i2c(0x3062, 0x00);
write_i2c(0x3063, 0xb8);
write_i2c(0x3064, 0x94);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x00);
write_i2c(0x3089, 0xb0);
write_i2c(0x308a, 0x00);
write_i2c(0x308b, 0x90);
write_i2c(0x3020, 0x01);
write_i2c(0x3021, 0x1d);
write_i2c(0x3022, 0x00);
write_i2c(0x3023, 0x06);
write_i2c(0x3024, 0x08);
write_i2c(0x3025, 0x18);
write_i2c(0x3026, 0x03);
write_i2c(0x3027, 0x04);

QCIF 2x
write_i2c(0x3302, 0xef);
write_i2c(0x335f, 0x12);
write_i2c(0x3360, 0x0c);
write_i2c(0x3361, 0x84);
write_i2c(0x3062, 0x00);
write_i2c(0x3063, 0xb8);
write_i2c(0x3064, 0x94);

write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x00);
write_i2c(0x3089, 0xb0);
write_i2c(0x308a, 0x00);
write_i2c(0x308b, 0x90);
write_i2c(0x3020, 0x03);
write_i2c(0x3021, 0x1d);
write_i2c(0x3022, 0x00);
write_i2c(0x3023, 0xc6);
write_i2c(0x3024, 0x04);
write_i2c(0x3025, 0x18);
write_i2c(0x3026, 0x01);
write_i2c(0x3027, 0x84);

QCIF 4x
write_i2c(0x3302, 0xef);
write_i2c(0x335f, 0x01);
write_i2c(0x3360, 0x0c);
write_i2c(0x3361, 0xc4);
write_i2c(0x3062, 0x00);
write_i2c(0x3063, 0xb8);
write_i2c(0x3064, 0x94);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x00);
write_i2c(0x3089, 0xb0);
write_i2c(0x308a, 0x00);
write_i2c(0x308b, 0x90);
write_i2c(0x3020, 0x04);
write_i2c(0x3021, 0x1d);
write_i2c(0x3022, 0x01);
write_i2c(0x3023, 0x26);
write_i2c(0x3024, 0x02);
write_i2c(0x3025, 0x18);
write_i2c(0x3026, 0x00);
write_i2c(0x3027, 0xc4);

//capture Zoom
//source size is QXGA, capture size all from QXGA, keep view angle with preview
**1.**
QVGA 1x

write_i2c(0x3302, 0xef);
write_i2c(0x335f, 0x68);
write_i2c(0x3360, 0x18);
write_i2c(0x3361, 0x0c);
write_i2c(0x3062, 0x01);
write_i2c(0x3063, 0x48);
write_i2c(0x3064, 0xf4);

```
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x01);
write_i2c(0x3089, 0x40);
write_i2c(0x308a, 0x00);
write_i2c(0x308b, 0xf0);
write_i2c(0x3020, 0x01);
write_i2c(0x3021, 0x1d);
write_i2c(0x3022, 0x00);
write_i2c(0x3023, 0x0a);
write_i2c(0x3024, 0x08);
write_i2c(0x3025, 0x18);
write_i2c(0x3026, 0x06);
write_i2c(0x3027, 0x0c);
```

QVGA 2x

```
write_i2c(0x3302, 0xef);
write_i2c(0x335f, 0x34);
write_i2c(0x3360, 0x18);
write_i2c(0x3361, 0x0c);
write_i2c(0x3062, 0x01);
write_i2c(0x3063, 0x48);
write_i2c(0x3064, 0xf4);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x01);
write_i2c(0x3089, 0x40);
write_i2c(0x308a, 0x00);
write_i2c(0x308b, 0xf0);
write_i2c(0x3020, 0x03);
write_i2c(0x3021, 0x1d);
write_i2c(0x3022, 0x01);
write_i2c(0x3023, 0x8a);
write_i2c(0x3024, 0x04);
write_i2c(0x3025, 0x18);
write_i2c(0x3026, 0x03);
write_i2c(0x3027, 0x0c);
```

QVGA 3x

```
write_i2c(0x3302, 0xef);
write_i2c(0x335f, 0x22);
write_i2c(0x3360, 0xc2);
write_i2c(0x3361, 0x0c);
write_i2c(0x3062, 0x01);
write_i2c(0x3063, 0x48);
write_i2c(0x3064, 0xf4);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x01);
write_i2c(0x3089, 0x40);
```

write_i2c(0x308a, 0x00);
write_i2c(0x308b, 0xf0);
write_i2c(0x3020, 0x03);
write_i2c(0x3021, 0xc8);
write_i2c(0x3022, 0x02);
write_i2c(0x3023, 0x0a);
write_i2c(0x3024, 0x02);
write_i2c(0x3025, 0xc2);
write_i2c(0x3026, 0x02);
write_i2c(0x3027, 0x0c);

**2.**
VGA 1x
write_i2c(0x3302, 0xef);
write_i2c(0x335f, 0x68);
write_i2c(0x3360, 0x18);
write_i2c(0x3361, 0x0c);
write_i2c(0x3062, 0x12);
write_i2c(0x3063, 0x88);
write_i2c(0x3064, 0xe4);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x02);
write_i2c(0x3089, 0x80);
write_i2c(0x308a, 0x01);
write_i2c(0x308b, 0xe0);
write_i2c(0x3020, 0x01);
write_i2c(0x3021, 0x1d);
write_i2c(0x3022, 0x00);
write_i2c(0x3023, 0x0a);
write_i2c(0x3024, 0x08);
write_i2c(0x3025, 0x18);
write_i2c(0x3026, 0x06);
write_i2c(0x3027, 0x0c);

VGA 2x
write_i2c(0x3302, 0xef);
write_i2c(0x335f, 0x34);
write_i2c(0x3360, 0x18);
write_i2c(0x3361, 0x0c);
write_i2c(0x3062, 0x12);
write_i2c(0x3063, 0x88);
write_i2c(0x3064, 0xe4);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x02);
write_i2c(0x3089, 0x80);
write_i2c(0x308a, 0x01);
write_i2c(0x308b, 0xe0);
write_i2c(0x3020, 0x03);

write_i2c(0x3021, 0x1d);
write_i2c(0x3022, 0x01);
write_i2c(0x3023, 0x8a);
write_i2c(0x3024, 0x04);
write_i2c(0x3025, 0x18);
write_i2c(0x3026, 0x03);
write_i2c(0x3027, 0x0c);

VGA 3x

write_i2c(0x3302, 0xef);
write_i2c(0x335f, 0x22);
write_i2c(0x3360, 0xc2);
write_i2c(0x3361, 0x0c);
write_i2c(0x3062, 0x12);
write_i2c(0x3063, 0x88);
write_i2c(0x3064, 0xe4);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x02);
write_i2c(0x3089, 0x80);
write_i2c(0x308a, 0x01);
write_i2c(0x308b, 0xe0);
write_i2c(0x3020, 0x03);
write_i2c(0x3021, 0xc8);
write_i2c(0x3022, 0x02);
write_i2c(0x3023, 0x0a);
write_i2c(0x3024, 0x02);
write_i2c(0x3025, 0xc2);
write_i2c(0x3026, 0x02);
write_i2c(0x3027, 0x0c);

**3.**
XGA 1x

write_i2c(0x3302, 0xef);
write_i2c(0x335f, 0x68);
write_i2c(0x3360, 0x18);
write_i2c(0x3361, 0x0c);
write_i2c(0x3062, 0x34);
write_i2c(0x3063, 0x08);
write_i2c(0x3064, 0x04);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x04);
write_i2c(0x3089, 0x00);
write_i2c(0x308a, 0x03);
write_i2c(0x308b, 0x00);
write_i2c(0x3020, 0x01);
write_i2c(0x3021, 0x1d);

write_i2c(0x3022, 0x00);
write_i2c(0x3023, 0x0a);
write_i2c(0x3024, 0x08);
write_i2c(0x3025, 0x18);
write_i2c(0x3026, 0x06);
write_i2c(0x3027, 0x0c);

XGA 2x

write_i2c(0x3302, 0xef);
write_i2c(0x335f, 0x34);
write_i2c(0x3360, 0x18);
write_i2c(0x3361, 0x0c);
write_i2c(0x3062, 0x34);
write_i2c(0x3063, 0x08);
write_i2c(0x3064, 0x04);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x04);
write_i2c(0x3089, 0x00);
write_i2c(0x308a, 0x03);
write_i2c(0x308b, 0x00);
write_i2c(0x3020, 0x03);
write_i2c(0x3021, 0x1d);
write_i2c(0x3022, 0x01);
write_i2c(0x3023, 0x8a);
write_i2c(0x3024, 0x04);
write_i2c(0x3025, 0x18);
write_i2c(0x3026, 0x03);
write_i2c(0x3027, 0x0c);

**4.**

SXGA 1x
write_i2c(0x3302, 0xef);
write_i2c(0x335f, 0x68);
write_i2c(0x3360, 0x18);
write_i2c(0x3361, 0x0c);
write_i2c(0x3062, 0x45);
write_i2c(0x3063, 0x08);
write_i2c(0x3064, 0x04);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x05);
write_i2c(0x3089, 0x00);
write_i2c(0x308a, 0x04);
write_i2c(0x308b, 0x00);
write_i2c(0x3020, 0x01);
write_i2c(0x3021, 0x1d);
write_i2c(0x3022, 0x00);

write_i2c(0x3023, 0x0a);
write_i2c(0x3024, 0x08);
write_i2c(0x3025, 0x18);
write_i2c(0x3026, 0x06);
write_i2c(0x3027, 0x0c);

SXGA 1.5x

write_i2c(0x3302, 0xef);
write_i2c(0x335f, 0x45);
write_i2c(0x3360, 0x6d);
write_i2c(0x3361, 0x0c);
write_i2c(0x3062, 0x45);
write_i2c(0x3063, 0x08);
write_i2c(0x3064, 0x04);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x05);
write_i2c(0x3089, 0x00);
write_i2c(0x308a, 0x04);
write_i2c(0x308b, 0x00);
write_i2c(0x3020, 0x02);
write_i2c(0x3021, 0x72);
write_i2c(0x3022, 0x01);
write_i2c(0x3023, 0x0a);
write_i2c(0x3024, 0x05);
write_i2c(0x3025, 0x6d);
write_i2c(0x3026, 0x04);
write_i2c(0x3027, 0x0c);

**5.**
UXGA 1x

write_i2c(0x3302, 0xef);
write_i2c(0x335f, 0x68);
write_i2c(0x3360, 0x18);
write_i2c(0x3361, 0x0c);
write_i2c(0x3062, 0x46);
write_i2c(0x3063, 0x48);
write_i2c(0x3064, 0xb4);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x06);
write_i2c(0x3089, 0x40);
write_i2c(0x308a, 0x04);
write_i2c(0x308b, 0xb0);
write_i2c(0x3020, 0x01);
write_i2c(0x3021, 0x1d);
write_i2c(0x3022, 0x00);
write_i2c(0x3023, 0x0a);

write_i2c(0x3024, 0x08);
write_i2c(0x3025, 0x18);
write_i2c(0x3026, 0x06);
write_i2c(0x3027, 0x0c);

UXGA 1.2x

write_i2c(0x3302, 0xef);
write_i2c(0x335f, 0x56);
write_i2c(0x3360, 0xc2);
write_i2c(0x3361, 0x0c);
write_i2c(0x3062, 0x46);
write_i2c(0x3063, 0x48);
write_i2c(0x3064, 0xb4);
write_i2c(0x3403, 0x42);
write_i2c(0x3088, 0x06);
write_i2c(0x3089, 0x40);
write_i2c(0x308a, 0x04);
write_i2c(0x308b, 0xb0);
write_i2c(0x3020, 0x01);
write_i2c(0x3021, 0xc8);
write_i2c(0x3022, 0x00);
write_i2c(0x3023, 0x8a);
write_i2c(0x3024, 0x06);
write_i2c(0x3025, 0xc2);
write_i2c(0x3026, 0x05);
write_i2c(0x3027, 0x0c);

**6.**
QXGA no zoom

## 13.2 Format Transfer Reference Setting

YUV422
write_i2c(0x3100, 0x02);
write_i2c(0x3301, 0xde);
write_i2c(0x3304, 0x00);
write_i2c(0x3400, 0x00);
write_i2c(0x3404, 0x00);
write_i2c(0x3600, 0xc4);

RGB565
write_i2c(0x3100, 0x02);
write_i2c(0x3301, 0xde);
write_i2c(0x3304, 0x00);
write_i2c(0x3400, 0x01);
write_i2c(0x3404, 0x11);

write_i2c(0x3600, 0xc4);

Processed raw
write_i2c(0x3100, 0x02);
write_i2c(0x3301, 0xde);
write_i2c(0x3304, 0x00);
write_i2c(0x3400, 0x01);
write_i2c(0x3404, 0x18);
write_i2c(0x3600, 0xc4);

Sensor raw
write_i2c(0x3100, 0x22);
write_i2c(0x3400, 0x04);
write_i2c(0x3600, 0xc4);

JPEG
write_i2c(0x3100, 0x32);
write_i2c(0x3304, 0x00);
write_i2c(0x3400, 0x02);
write_i2c(0x3404, 0x22);
write_i2c(0x3500, 0x00);
write_i2c(0x3600, 0x42);Vsync mode2 select
write_i2c(0x3610, 0x00);
write_i2c(0x3611, 0x20);fixed width 1024 pclk
write_i2c(0x3507, 0x04); QScale
write_i2c(0x350a, 0x4f);
write_i2c(0x304c, 0x82);

## *14. Capture Sequence*

### 14.1 Shutter

The shutter of OV3640 controls exposure time. The unit of shutter is line period.

Shutter value has limitation for each output resolution. If no dummy lines are inserted, the maximum shutter value for QXGA resolution is 1563. The maximum shutter value for XGA resolution is 779.

Default_XGA_maximum_shutter = 779;
Default_QXGA_maximum_shutter = 1563;

The shutter value are stored in 2 registers, reg0x3002, reg0x3003 .

Shutter = reg0x3002 << 8 + reg0x3003;

### 14. 2 Dummy Lines

The exposure could be increased further by insert dummy lines. When dummy lines are inserted,

frame rate also changes.

The are 2 kinds of dummy lines could be inserted. Dummy line before data output and dummy line after data output.

### 14.2.1 Extra Line

If dummy lines are inserted before data output, which is called extra line, the actual exposure time is increased. The extra line is controlled by register 0x302d and 0x302e.

Exposure = Shutter + Extra_lines

Extra_lines = reg0x302d + (reg0x302e << 8);

the maximum shutter value is not changed.

So even shutter value is 0, the minimum exposure time is Extra_lines.

Usually, extra lines should be inserted automatically. If the environment is dark, longer exposure time is required, extra line number increased. If the environment is bright, shorter exposure time is required, extra line number decreased. If extra line is inserted manually and the value is fixed, the total exposure time can not be less then extra_line in bright environment, the image would be over exposed.

The extra lines are inserted inside the active period of Vsync, the timing of output period in which Vsync is inactive is not changed.

### 14.2.2 Dummy Line

If dummy lines are inserted after data output, which is called dummy line, the maximum shutter value is changed. The dummy lines are inserted between two Vsync. The number of dummy lines is controlled by register 0x302a and 0x302b. Different with other sensor, there have default values in such two register, Default_Reg0x302a, Default_Reg0x302b.

XGA_maximum_shutter = Default_XGA_Maximum_Shutter + Dummy_line
QXGA_maximum_shutter = Default_QXGA_Maximum_Shutter + Dummy_line

The exposure time is

Exposure = Shutter

Dummy_line = (Reg0x302a-Default_Reg0x302a) <<8 + Reg0x302b-Default_Reg0x302b;

For XGA: Default_Reg0x302a = 0x03; Default_Reg0x302b = 0x10;
For QXGA: Default_Reg0x302a = 0x06; Default_Reg0x302b = 0x20;

| | Extra line | Dummy Line |
|---|---|---|
| Registers | 0x302d, 0x302e | 0x302a, 0x302b |
| Minimum Shutter Value | 0 | 0 |

| | | |
|---|---|---|
| Maximum Shutter Value | 1563 for QXGA<br>779 for XGA | 1563 + Dummy_line for QXGA<br>799 + Dummy_line for XGA |
| Minimum Exposure | Extra_lines | 0 |
| Maximum Exposure | Maximum_Shutter + Extra_lines | Maximum_Shutter |

So if both dummy line and extra line are inserted, the exposure time is

Exposure = Shutter + Extra_Lines

And the maximum shutter value is

XGA_maximum_shutter = Default_XGA_Maximum_Shutter + Dummy_line
QXGA_maximum_shutter = Default_QXGA_Maximum_Shutter + Dummy_line

## 14. 3 Dummy Pixels

If no dummy pixel is inserted, the line width is called default line width. Dummy pixel register is 0x3028, 0x3029.
Default_XGA_Line_Width = 1188;
Default_QXGA_Line_Width = 2376;

When dummy pixel is inserted, the line width changes and frame rate also changes.

XGA_Line_Width = Default_XGA_Line_Width + Dummy_pixel
QXGA_Line_Width = Default_QXGA_Line_Width + Dummy_pixel

## 14.4 Gain

Gain is stored in reg0x3000 and Reg0x3001. If only use the gain of Reg0x3001, maximum gain of 32x could be reached. It is enough for camera phone. So we don't discuss reg0x3000 here.

Gain = (((reg0x3001 & 0xf0)>>4) + 1)*(1 + (reg0x3001 & 0x0f)/16)

## 14.5 Banding Filter

### *14.5.1 Preview*

Automatic Banding filter is used for preview.

### *14.5.2 Capture*

Manual banding filter is used for capture.
For 50Hz, the banding filter calculation is

Banding_Filter = Capture_PCLK_Frequency /100 /capture_line_width

For 60Hz, the banding filter calculation is

Banding_Filter = Capture_PCLK_Frequency /120 /capture_line_width

Capture_Exposure = n*Banding_Filter

n is an integer.

## 14.6 Auto frame rate

Auto frame rate could be enabled by turn on night mode. When night mode is enabled, the extra line are adjusted automatically.

## 14.7 Capture Sequence

### *14.7.1 Preview*

// Initialize OV3640 for preview

Different with other sensor, there have default values in dummy pixel, Default_Reg0x3028, Default_Reg0x3029,dummy lines register, Default_Reg0x302a, Default_Reg0x302b. And XGA and QXGA have different default values. So dummy pixel values and dummy line values are new values minus default values of such registers.
In common, Preview_dummy_pixel,Preview_dummy_line,Capture_dummy_pixel and Capture_dummy_line can be set to zero.

```
// Dummy pixel and Dummy line could be inserted for preview
Preview_dummy_pixel =
Preview_dummy_line =

If (Resolution == XGA) {
        Preview_dummy_pixel_val = Preview_dummy_pixel * 2;
}
else {
        Preview_dummy_pixel_val = Preview_dummy_pixel;
}
Reg0x3029 = Preview_dummy_pixel_val & 0x00ff + Default_Reg0x3029;
Reg0x3028 = Preview_dummy_pixel_val >>8 + Default_Reg0x3028;

write_i2c(0x3029, Reg0x3029);
write_i2c(0x3028, Reg0x3028);

// update dummy line
Reg0x302b = Preview_dummy_line & 0x00ff + Default_Reg0x302b;
Reg0x302a = Preview_dummy_line >>8 + Default_Reg0x302a;

write_i2c(0x302a, Reg0x302a);
write_i2c(0x302b, Reg0x302b);
```

### *14.7.2 Single Focus for AF Module*

Step 1: Read out state register value state_current,

if(state_current == STATE_INF)goto step2;
else
go to step1;
Step 2: Write cmd_Capture(0x03) to command register(0x3f00);Detailed information please refer to 16.2.2

### 14.7.3 Stop Preview

```
//Stop AE/AG

reg0x3013 = read_i2c(0x3013);
Reg0x3013 = reg0x3013 & 0xfa;
write_i2c(0x3013,Reg0x3013);

//Read back preview shutter
reg0x3002 = read_i2c(0x3002);
reg0x3003 = read_i2c(0x3003);

Shutter = reg0x3002<<8 + reg0x3003;

//Read back extra line
reg0x302d = read_i2c(0x302d);
reg0x302e = read_i2c(0x302e);
Extra_lines = reg0x302e + (reg0x302d<<8);

Preview_Exposure = Shutter + Extra_lines;

//Read Back Gain for preview
reg0x3001 = read_i2c(0x3001);
Preview_Gain16 = (((Reg0x3001 & 0xf0)>>4) + 1) * (16 + reg0x3001 & 0x0f);

//Read back dummy pixels
reg0x3028 = read_i2c(0x3028);
reg0x3029 = read_i2c(0x3029);
Preview_dummy_pixels = ((reg0x3028-Default_Reg0x3028) & 0xf0)<<8 + reg0x3029-
Default_Reg0x3029;

If (Resolution == XGA) {
        Preview_dummy_pixels = Preview_dummy_pixels/2 ;
}
```

### 14.7.4 Calculate Capture Exposure

```
// Dummy Pixel and Dummy Line could be inserted for capture
Capture_dummy_pixel =
Capture_dummy_line =
Preview_PCLK_frequency =
Capture_PCLK_frequency =

// Capture maximum gain could be defined.
```

```
// Capture_max_gain16 = capture_max_gain * 16
//
Preview_line_width = Default_XGA_Line_Width + Preview_dummy_pixel ;
If (resolution ==XGA) {
        Capture_line_width = Default_XGA_Line_Width + capture_Dummy_pixel;
}
else {
        Capture_line_width = Default_QXGA_Line_Width + capture_Dummy_pixel;
}

If (resolution ==XGA) {
        Capture_maximum_shutter = Default_XGA_maximum_shutter + capture_dummy_lines;
}
else {
        Capture_maximum_shutter = Default_QXGA_maximum_shutter + capture_dummy_lines;
}

Capture_Exposure = 2*
        Preview_Exposure * Capture_PCLK_Frequency/Preview_PCLK_Frequency *
        Preview_Line_width/Capture_Line_Width;

//Calculate banding filter value
If (50Hz) {
        If (format == RGB) {//RGB indicates raw RGB
                Capture_banding_Filter = Capture_PCLK_Frequency /100 /capture_line_width;
        }
        else {
                Capture_banding_Filter = Capture_PCLK_Frequency/ 100/ (2*capture_line_width);
        }
else {(60Hz)
        If (format == RGB) {
                Capture_banding_Filter = Capture_PCLK_Frequency /120 /capture_line_width;
        }
        else {
                Capture_banding_Filter = Capture_PCLK_frequency /120 /(2*capture_line_width);
        }
}

//redistribute gain and exposure
Gain_Exposure = Preview_Gain16 * Capture_Exposure;
If (Gain_Exposure <  Capture_Banding_Filter * 16) {
        // Exposure < 1/100
        Capture_Exposure = Gain_Exposure /16;
        Capture_Gain16 = (Gain_Exposure*2 + 1)/Capture_Exposure/2;
}
else {
        If (Gain_Exposure >  Capture_Maximum_Shutter * 16) {
                // Exposure > Capture_Maximum_Shutter
```

```
                Capture_Exposure = Capture_Maximum_Shutter;
                Capture_Gain16 = (Gain_Exposure*2 + 1)/Capture_Maximum_Shutter/2;

                If (Capture_Gain16 > Capture_Max_Gain16) {
                        // gain reach maximum, insert extra line
                        Capture_Exposure = Gain_Exposure*1.1/Capture_Max_Gain16;
                        // For 50Hz, Exposure = n/100; For 60Hz, Exposure = n/120
                        Capture_Exposure = Capture_Exposure/Capture_banding_filter;
                        Capture_Exposure =
                                Capture_Exposure * Capture_banding_filter;
                        Capture_Gain16 = (Gain_Exposure *2+1)/ Capture_Exposure/2;
                }
        }
        else {
                // 1/100(120) < Exposure < Capture_Maximum_Shutter, Exposure = n/100(120)
                Capture_Exposure = Gain_Exposure/16/Capture_banding_filter;
                Capture_Exposure =  Capture_Exposure * Capture_banding_filter;
                Capture_Gain16 = (Gain_Exposure*2 +1) / Capture_Exposure/2;
        }
}
```

### 14.7.5 Switch to QXGA

```
//  Write registers, change to QXGA resolution.
```

### 14.7.6 Write Registers

```
//write dummy pixels
reg0x3029 = Capture_dummy_pixels & 0x00ff;
reg0x3028 = read_i2c(0x3028);
reg0x3028 = (reg0x3028 & 0x0f) | ((Capture_dummy_pixels & 0x0f00)>>4);
write_i2c(0x3028, reg0x3028+Default_Reg0x3028);
write_i2c(0x302b, reg0x3029+Default_Reg0x3029);

//Write Dummy Lines
Reg0x302b = Capture_dummy_line & 0x00ff + Default_Reg0x302b;
Reg0x302a = Capture_dummy_line >>8 + Default_Reg0x302a;
write_i2c(0x302a, Reg0x302a);
write_i2c(0x302b, Reg0x302b);

//Write Exposure
If (Capture_Exposure > Capture_maximum_shutter) {
        Shutter = Capture_maximum_shutter;
        Extra_lines = Capture_Exposure – Capture_maximum_shutter;
}
else {
        Shutter = Capture_Exposure;
        Extra_lines = 0;
}
```

```
Reg0x3003 = Shutter  & 0x00ff;
Reg0x3002 = (Shutter >>8) & 0x00ff;

write_i2c(0x3003, Reg0x3003);
write_i2c(0x3002, Reg0x3002);

// Write extra line
reg0x302e = Extra_lines & 0x00ff;
reg0x302d = Extra_lines >> 8;
write_i2c(0x302d, reg0x302d);
write_i2c(0x302e, reg0x302e);

// Write Gain
Gain = 0;
If (Capture_Gain16 > 31) {
        Capture_Gain16 = Capture_Gain16 /2;
        Gain = 0x10;
}
If (Capture_Gain16 > 31) {
        Capture_Gain16 = Capture_Gain16 /2;
        Gain = Gain | 0x20;
}
If (Capture_Gain16 > 31) {
        Capture_Gain16 = Capture_Gain16 /2;
        Gain = Gain | 0x40;
}
If (Capture_Gain16 > 31) {
        Capture_Gain16 = Capture_Gain16 /2;
        Gain = Gain | 0x80;
}
If (Capture_Gain16 > 16) {
        Gain = Gain | (Capture_Gain16 -16);
}
write_i2c(0x3001, Gain);
```



### *14.7.7 Capture*

// Wait for 2 Vsync
// Capture the 3rd frame.

### *14.7.8 Send finish command for AF module*

Refer to 16.2.2 for detailed operation.

### *14.7.9 Back to preview*

//Write Registers, Change to XGA

...

```
//Start AG/AE
Reg0x3013 = read_i2c(0x3013);
Reg0x3013 = Reg0x3013 | 0x05;
write_i2c(0x3013, Reg0x3013);
```

## 15. Strobe Flash Control

To achieve best image quality in low light condition, strobe flash is recommended.

OV3640 can output one programmable signal from strobe pin.

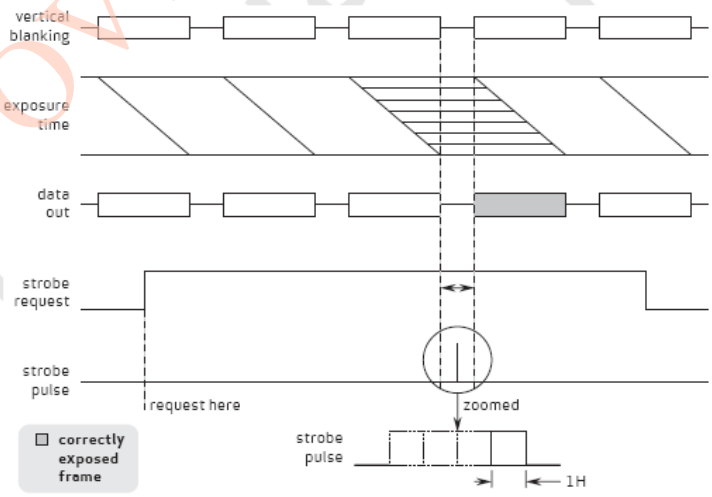
| | | |
|---|---|---|
| strobe function enable | 0x307A[7] | strobe function enable<br>0: strobe disable<br>1: start strobe enable |
| strobe output pulse polarity control | 0x307A[6]<br>(TMC4[6]) | strobe output polarity control<br>0: positive pulse<br>1: negative pulse |
| xenon mode strobe pulse width | 0x307A[3:2]<br>(TMC4[3:2]) | xenon mode pulse width<br>00: 1 line<br>01: 2 lines<br>10: 3 lines<br>11: 4 lines |
| strobe mode | 0x307A[1:0]<br>(TMC4[1:0]) | strobe mode select<br>00: xenon mode<br>01: LED 1 & 2 mode<br>10: LED 1 & 2 mode<br>11: LED 3 mode |

### 15.1 Strobe Pulse

The strobe signal is programmable. It support LED and Xenon mode.

### 15.2 Xenon flash control

After a strobe request is submitted, the strobe pulse will be activated at the beginning of the third frame . The third frame will be correctly exposed. The pulse width can be changed in Xenon mode between 1H and 4H, where H is one horizontal period.




### 15.3 Application for Xenon flash

Xenon control through STROBE pin of OV3640, STROBE will go high one-four lines period at the VSYNC blanking period when active.

Turn on strobe pin (to high) in the 3rd frame Vsync blanking period

78 307a 00 80;clear bit[7]

78 307a 80 80;set bit[7]

Turn off strobe pin

78 307a 00 80; bit[7],from 1 to 0, turn off strobe pin

### 15.4 Capture flow with Xenon flash

OV3640 Recommended capturing sequence with using Xenon Flash.

Step 01 Stop frame out

Step 02 Stop AEC/AGC and set AEC value to maximum if it is not maximum

Step 03 Change the resolution for capture mode

Step 04 Resume frame out

Step 05 Issue Xenon flash start

Step 06 Still Image capture at 3rd frame while Xenon flash will be light on

Step 07 Stop frame out and to step 9 if capture is successful, else back to step 6

Step 08 Change back the resolution for preview

Step 09 Start AEC/AGC

Step 10 Resume frame out



## *16. Auto Focus Application Solution*

### 16.1 Auto Focus function Introduce

OmniVision Had built-in Auto Focus Control(AFC) both in chip and in firmware of OV3640. Currently the auto focus firmware support below three types of auto focus camera module:

1. OV3640+VCM+AD5820
2. OV3640+VCM+AD5398
3. OV3640+VCM+DW9710

#### *16.1.1 General Auto Focus Control Flow*

Step1. OV3640 Initialization. Refer to 13.1 VGA preview settings.
Step2. Download firmware to built-in memory of OV3640. Confirm VCM type and contact OmniVision local FAE to get related auto focus firmware.
Step3. Send Firmware Commands to OV3640 to control Auto Focus Functions
Detailed command is in below.
Step4. If reset or cut off power of OV3640, go to Step1.

### 16.1.2 AF Firmware Download

The format of AF Firmware is same as register settings file. It should be downloaded to OV3640 one by one byte.

### 16.1.3 General Auto Focus Control Flow

The firmware command register is 0x3f00. The host can control AF by sending commands to this register. The firmware command register will auto clear to zero after command is processed.

| 0x3F00 | Command Description |
|---|---|
| 0x01 (cmd_EnOverlay) | Enable overlay window |
| 0x02 (cmd_DisOverlay) | Disable Overlay window |
| 0x03 (cmd_SingleMode) | Single focus mode |
| 0x06 (cmd_Capture) | Capture Command after focus success |
| 0x08 (cmd_Finish) | Finish Command after focused |
| 0x10*~0x15* | Reserved |
| *Reserved function commands. | |



### 16.1.4 Auto Focus Control Firmware State

The firmware state register is 0x3f01, by which the host can get current running state of firmware. The following is the bit description of state register.

| BIT Index | Bit Description |
|---|---|
| BIT 7 | Step State --- 1 : Failed 0 : Success |
| BIT 6 | 1 Mode Change Mask |
| BIT 5 | 1 Capture Mask |
| BIT 4 | 0 Reserved |
| BIT 3,2 | Mode |
| | 00 Idle Mode |
| | 01 Single Focus Mode |
| | 10 Continue Focus Mode |
| | 11 Step Mode |
| BIT 1,0 | Mode Step |
| | 00 Instruction |
| | 01 Focusing |
| | 10 Focused |
| | 11 Capture |



Based on the above bit description, the following firmware states are defined:

```
#define STEP_STATE_NO              0x80
#define MASK_MODECHANGE            0x40
#define MASK_CAPTURECMD            0x20
#define RESERVED_INVALID           0x10

#define MODE_IDLE                  0x00
#define MODE_SINGLE                0x04
#define MODE_CONTINUE              0x08
#define MODE_STEP                  0x0c

#define MODE_STEP_INSTRUCTION      0x00
#define MODE_STEP_FOCUSING         0x01
#define MODE_STEP_FOCUSED          0x02
#define MODE_STEP_CAPTURE          0x03

STATE_INF          =MODE_IDLE|MODE_STEP_INSTRUCTION,
STATE_SINGLE       =MODE_SINGLE|MODE_STEP_FOCUSING|MASK_MODECHANGE|
MASK_CAPTURECMD,
```

STATE_SUCCESS_S=MODE_SINGLE|MODE_STEP_FOCUSED|MASK_MODECHANGE,
STATE_FAIL_S =MODE_SINGLE|MODE_STEP_FOCUSED|MASK_MODECHANGE|
STEP_STATE_NO,
STATE_CAPTURE_S=MODE_SINGLE|MODE_STEP_CAPTURE|MASK_MODECHANGE,

## 16.2 Auto Focus Module Application Flow

### *16.2.1 Initiation*

After send initial preview settings and firmware download successfully, AF will be initialized and the state value read from state register(0x3f01) will be state STATE_INF.

### *16.2.2 Single Focus*

1.Check current firmware running state.
Read out state register value state_current
if(state_current == STATE_INF)goto next step;
2. Send single focus command.
Step1: write cmd_SingleMode(0x03)to command register(0x3f00);
Step2: Read out state register value state_current.
if(state_current == STATE_SINGLE)goto step2;
else if(state_current == STATE_FAIL_S)single focus failed;
else if(state_current == STATE_SUCCESS_S)single focus success;
3. Send finish command.
First Write cmd_Finish(0x08) to command register(0x3f00);
Second Read out state register value state_current.
If(state_current == STATE_INF)finish OK and focus on infinity.

## 16.3 Customer develop AF code by themselves

Please refer to OV3640 auto focus application notes document.

# *17. Power Down*

## 17.1 Hardware Power down

Set PWDN pin as high. It will halt internal device clock and counter .Detailed information, please refer to 3640 Hardware Application Notes

## 17.2 Software Power down

Software power down not halt internal device clock, only halt internal circuit activity.

**Enter into Sleep(software power down)mode**

```
write_i2c(0x361e, 0x00);
write_i2c(0x308d, 0x06);
write_i2c(0x30ad, 0x82);
write_i2c(0x308d, 0x0f);
```

**Wake up from Sleep mode**

```
write_i2c(0x308d, 0x00);
write_i2c(0x30ad, 0x02);
write_i2c(0x308d, 0x00);
write_i2c(0x361e, 0x00);
```

Revision History

**Rev1.10**

Update ZOOM and Power down

**Rev1.20**

Update PCLK output when output small size

Update YUV Sequence.

Update Sharpness

**Rev1.21**

Update Gain/Exposure Algorithm

**Rev1.22**

Update Auto Night Mode settings.

**Rev1.3**

Update Initial settings.

**Rev1.4**

Update Initial settings and DCW settings for R2C.

**Rev1.5**

Add AF function and Strobe Flash function,update banding filter register

**Rev1.6**

Preview  settings are all DCW from QXGA

Del 14.3 fps settings.

**Rev1.7**

Update Register initial settings

Update auto focus module solution and capture sequence

**Rev1.7**

Update QXGA maximum exposure lines