ArduCAM Camera Shield Series

SPI Camera Software Application Note

Rev 2.0, Oct 2016

Table of Contents

1 Introduction

This application note describes the detail software operation of ArduCAM camera shield. The latest source code library and examples can be downloaded from the https://github.com/arducam.

2 Software Library Structure

The ArdCAM library is designed for Arduino platform,which is composed by two sub-libraries one is ArduCAM and the other is UTFT4ArduCAM_SPI. These two libraries should be copied right under the libraries of Arduino directory in order to be recognized by the Arduino IDE. The ArduCAM libraries structure is shown as Figure 1.

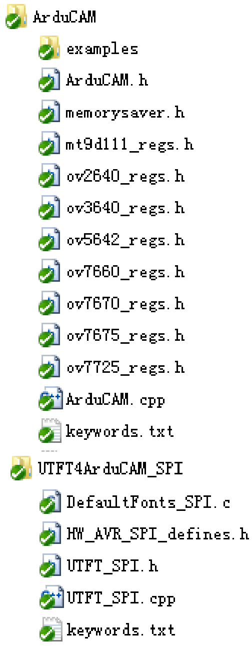

Figure 1 ArduCAM Libraries Structure

The ArduCAM library is the core library for ArduCAM shields. It contains supported image sensor drivers and user land API functions which issue capture or image data read commands .There is also an example directory inside the ArduCAM library which illustrates most function of the ArduCAM shields. The existing examples are plug and play without need to write a single line of code.

The UTFT4ArduCAM_SPI library is modified version of UTFT which is written by Henning Karlsen from http://www.henningkarlsen.com/electronics. We ported it to support ArduCAM shield with LCD screen. So the UTFT4ArduCAM_SPI library is only needed when using the ArduCAM-LF model.

3 Quick Start Guide

The libraries should be configured before running any examples, or else you will get a compilation error message. Open the memorysaver.h file in the ArduCAM folder and enable the hardware platform and camera module which matches to your hardware by comment or uncomment the macro definition in the file. For example, if you got a ArduCAM-Mini-2MP you should uncomment the line " #define OV2640_MINI_2MP " and comment all the other lines. And if you got a ArduCAM-Shield-V2 and a OV5642 camera module, you should uncomment the line

" #define ARDUCAM_SHIELD_V2" and the line " #define OV5642_CAM" then comment other lines.

After that open the Arduino IDE, the ArduCAM examples can be found from the menu File->Examples->ArduCAM as the Figure 2 shown.

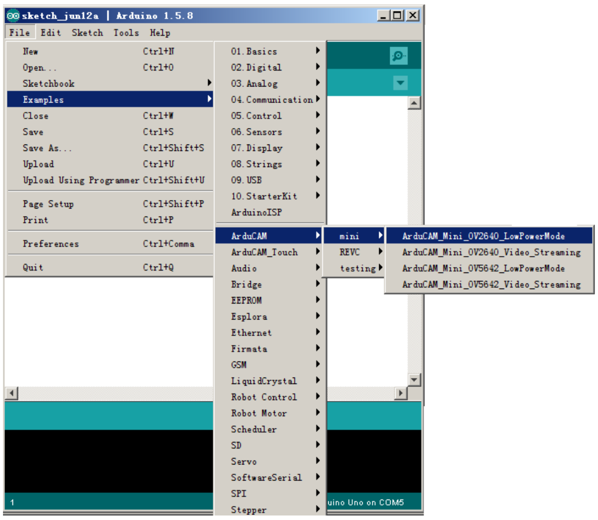

Figure 2 Arduino IDE examples

Open one of the examples, wiring SPI and I2C interface especially CS pins to ArduCAM shield according to the examples. More information about the wiring can be found from ArduCAM hardware application note. Selecting correct COM port and Arduino boards then upload the sketches as the Figure 3 shown.

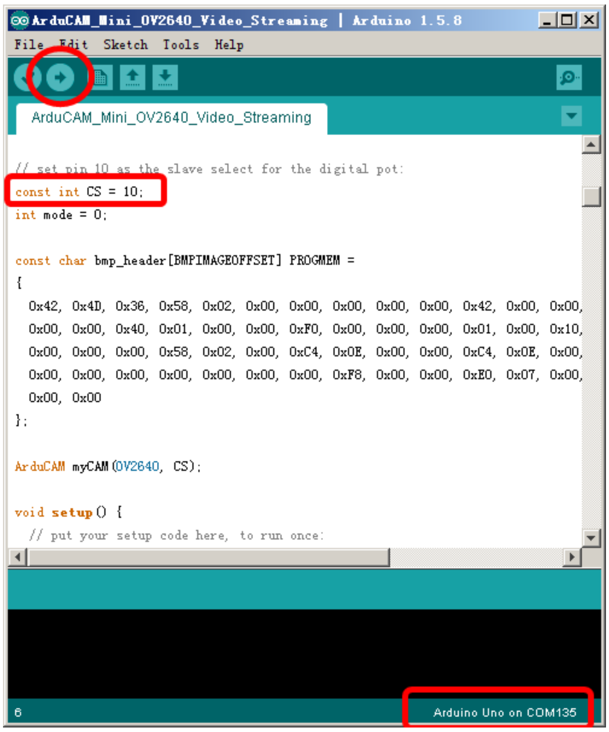

Figure 3 Example Sketch

After uploading the example sketch, user can preview the live video on LCD screen if using ArduCAM-LF model. Or downloading Windows host application here to capture image if using ArduCAM Mini model.

4 Example Sketches

In the example folder there are six sub directories for different ArduCAM models and the host application. Directories structure lists as Figure 4 shown. The **ESP8266** folder is for **ArduCAM-ESP8266-UNO** board examples. The **Mini** folder is for ArduCAM-Mini-2MP and ArduCAM-Mini-5MP modules. The **Mini_5MP_Plus** folder is for ArduCAM-Mini-5MP-Plus (OV5640/OV5642) modules. The **RevC** folder is for ArduCAM-Shield-RevC or ArduCAM-Shield-RevC+ shields. The **Shield_V2** folder is for ArduCAM-Shield-V2 shield. The **host_app** folder is host capture and display application for all of ArduCAM modules.

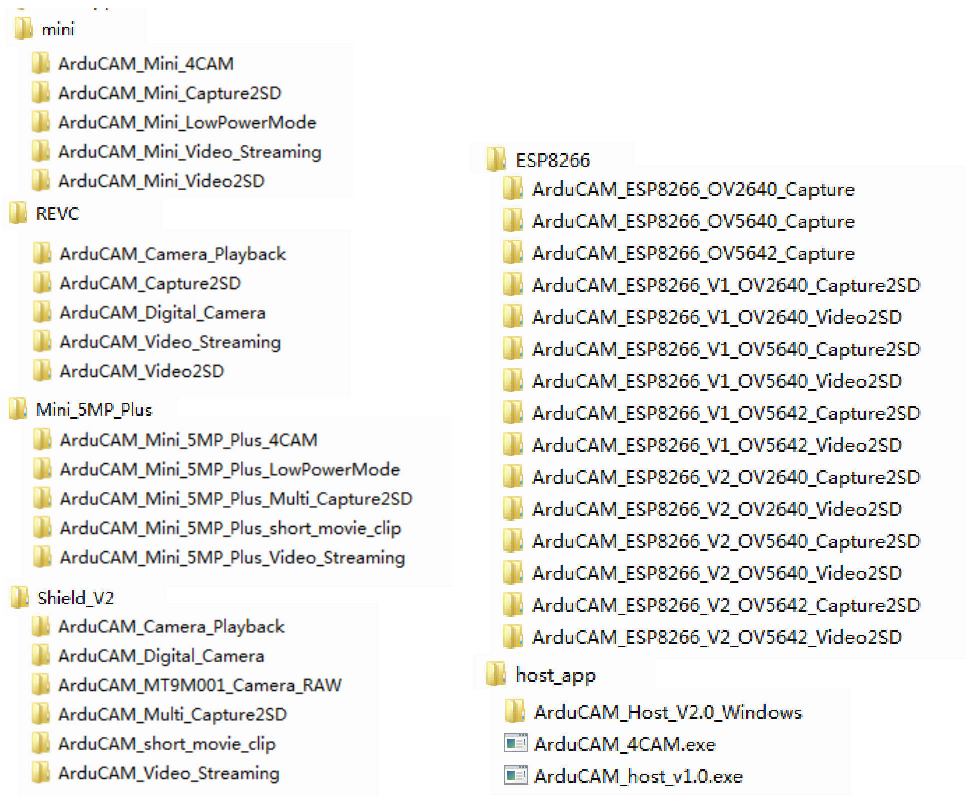

Figure 4 Example Folder Structure

4.1 ArduCAM Mini Examples

The mini folder contains examples for ArduCAM Mini shields. All of the examples are designed for ArduCAM-Mini-2MP and ArduCAM-Mini-5MP, and will take effect automatically according to the Macro definition in the memorysaver.h file.

4.1.1 ArduCAM_Mini_Video_Streaming

This example illustrates how to send continues capture commands to ArduCAM and transfer the JPEG image data back to host application via Arduino onboard USB-Serial interface. Note that the higher resolution wills cause higher image size and reduce the streaming frame rate accordingly. These examples should work with host application to view the captured images.

4.1.2 ArduCAM_Mini_LowPowerMode

This example is similar to example 4.1.1, and illustrates how to disable unwanted power consumption from the sensor and memory chip after each capture. It is useful for battery powered application. This example is recommended for ArduCAM mini 5MP module, because it will become extremely hot when running in full power. These examples should work with host application to view the captured images.

4.1.3 ArduCAM_Mini_Capture2SD

This example demonstrates how to capture time elapse image and save to TF/SD card.

4.1.4 ArduCAM_Mini_Video2SD

This example demonstrates how to capture low resolution and low frame rate MJPEG video to AVI file.

4.1.5 ArduCAM_Mini_4CAM

This example demonstrates how to connect 4 ArduCAM-Mini (2MP or 5MP) to ArduCAM multi-camera adapter board and capture images over the USB-Serial similar to example 4.1.1. This example should work with this host application.

4.2 ArduCAM Mini 5MP Plus Examples

Similar to ArduCAM Mini examples, the examples in this folder is designed for ArduCAM-Mini-5MP-Plus (OV5640/OV5642) modules with additional performance and features.

4.2.1 ArduCAM_Mini_5MP_Plus_Video_Streaming

Similar to ArduCAM-Mini example, see section 4.1.1.

4.2.2 ArduCAM_Mini_5MP_Plus_LowPowerMode

Similar to ArduCAM-Mini example, see section 4.1.2.

4.2.3 ArduCAM_Mini_5MP_Plus_Multi_Capture2SD

This example illustrates how to capture continuous pictures then save to TF/SD card, the maximum allowed pictures numbers is limited to 7. This example is useful if you want to create HDR image with different exposure value by manually set different exposure value at the beginning of each frame.

4.2.4 ArduCAM_Mini_5MP_Plus_short_movie_clip

This example illustrates how to record short movie clip by capturing continuous JPEG images until the entire 8MByte frame buffer is full, then save the MJPEG images as an AVI file into TF/SD card. You can playback the avi file on your PC media player software.

4.2.5 ArduCAM_Mini_5MP_Plus_4CAM

Similar to ArduCAM-Mini example, see section 4.1.5.

4.3 ArduCAM REVC Examples

The REVC folder contains examples for ArduCAM Rev.C and Rev.C+ shield. It requires additional UTFT4ArduCAM_SPI library as mentioned earlier. All of the examples are designed for different camera modules, and will take effect automatically according to the Macro definition in the memorysaver.h file.

4.3.1 ArduCAM_Camera_Playback

This example captures a 320x240 resolution BMP file and stores into SD card memory, then playback captured image on LCD screen if press the shutter button more than 3 seconds.

4.3.2 ArduCAM_Digital_Camera

This example acts like a true point to shoot digital camera. It starts live preview on LCD screen, and captures high resolution JPEG image after press the shutter button. Note that the image size has to fit into the onboard frame buffer size in order to prevent buffer overflow.

4.3.3 ArduCAM_Video_Streaming

Similar to ArduCAM-Mini example, see section 4.1.1.

4.3.4 ArduCAM_Capture2SD

Similar to ArduCAM-Mini example, see section 4.1.3.

4.3.5 ArduCAM_Video2SD

Similar to ArduCAM-Mini example, see section 4.1.4.

4.4 ArduCAM Shield V2 Examples

Similar to ArduCAM Shield Rev.C/Rev.C+ and ArduCAM-Mini-5MP-Plus examples, the examples in this folder is designed for ArduCAM-Shield-V2 with additional performance and features. All of the examples are designed for different camera modules, and will take effect automatically according to the Macro definition in the memorysaver.h file. And examples also work with ArduCAM-Nano-ESP8266 module.

4.4.1 ArduCAM_Camera_Playback

Similar to ArduCAM-Mini example, see section 4.3.1.

4.4.2 ArduCAM_Digital_Camera

Similar to ArduCAM-Mini example, see section 4.3.2.

4.4.3 ArduCAM_Video_Streaming

Similar to ArduCAM-Mini example, see section 4.1.1.

4.4.4 ArduCAM_Multi_Capture2SD

Similar to ArduCAM-Mini example, see section 4.2.3.

4.4.5 ArduCAM_short_movie_clip

Similar to ArduCAM-Mini example, see section 4.2.4.

4.4.6 ArduCAM_MT9M001_Camera_RAW

This example captures a 1280x1240 resolution bayer RAW image and stores into TF/SD card memory. The LCD screen will look weird because the LCD only support RGB565 format, when the video is RAW format, the color represents incorrectly.

4.5 ESP8266 UNO Examples

The examples in the ESP8266 folder is for ArduCAM ESP8266 UNO board V1 or V2, and should work with ArduCAM-Mini and ArduCAM-Mini-Plus modules.

4.5.1 ArduCAM_ESP8266_OVxxxx_Capture

This example demonstrates how to capture image over HTTP standard protocol and display the captured image/video on the webpage in the html folder. The example is configured as AP mode with default IP address 192.168.4.1 without password. You can also manually modify the parameters in the example to configure the camera as station mode and connect to your home router.

4.5.2 ArduCAM_ESP8266_Vx_OVxxxx_Capture2SD

Similar to ArduCAM-Mini example, see section 4.1.3.

4.5.3 ArduCAM_ESP8266_Vx_OVxxxx_Video2SD

Similar to ArduCAM-Mini example, see section 4.1.4.

5 ArduChip Functions

ArduChip is ArduCAM property technology which handles all the timing control over camera interface, LCD interface, frame buffer and SPI interface timings with a set of registers. The ArduChip register address is also called Command Code, user can use low level APIs with these command codes to achieve customized combination of actions that off the shelf APIs don’t have.

Different ArcuCAM platform uses different ArduChip and has different functionalities. Here is a list of possible hardware platforms:

Hardware Platform	Functions					
	Single Capture/ Read	Burst Read	Multiple Capture	Rewind	Low Power Mode	Short Video Capture
ArduCAM Shield Rev.C	√					
ArduCAM Shield Rev.C+	√	√	√	√		
ArduCAM-Mini-2MP	√	√		√	√	
ArduCAM-Mini-5MP	√	√	√	√	√	
ArduCAM-Mini-5MP (Bit-Roation-Fixed)	√	√	√	√	√	
ArduCAM Shield V2	√	√	√	√		√
ArduCAM-Mini-5MP-Plus (OV5642)	√	√	√	√	√	√
ArduCAM-Mini-5MP-Plus (OV5640)	√	√	√	√	√	√

5.1 Single Capture Mode

It is a basic capture function of the ArduChip. The capture command code is 0x84, and write ‘1’ to bit[1] to start a capture sequence. And then polling bit[3] which is the capture done flag by sending command code 0x41. After capture is done, user have to clear the capture done flag by sending command code 0x41 and write ‘1’ into bit[0] before next capture command.

5.2 Multiple Capture Mode

By sending the command code 0x81 and with writing the number of images to be capture into bit[2:0], before starting the capture command as the single capture sequence does. Please note that user should trade off between the resolution and number of images to be captured and do not make the frame buffer overflow.

5.3 Short Video Capture Mode

Use the same command as the Multiple Capture Mode. When the value bit[2:0] equals to 7, the ArduCAM will continuously capture the images until the entire frame buffer is full. User can save the captured MJPEG to AVI files to create short movie clips.

5.4 Single Read Operation

It is basic memory read function which start a single read operation and read a single byte each time. By sending command code 0x3D to start a single read operation, a single byte is read out from the frame buffer.

5.5 Burst Read Operation

It is advance capture function which can read multiple bytes out of the frame buffer by just sending a single command code 0x3C.

Please note that for these hardware platforms (ArduCAM Shield Rev.C+, ArduCAM-Mini-2MP, ArduCAM-Mini-5MP) the first read byte should be ignored in the first read transaction, because it is a dummy byte. In the following read transaction, the first byte read is the last read byte in the last read transaction, it is very important. And do not use other SPI command between burst read transaction. Detail timing can be found from Figure 5.

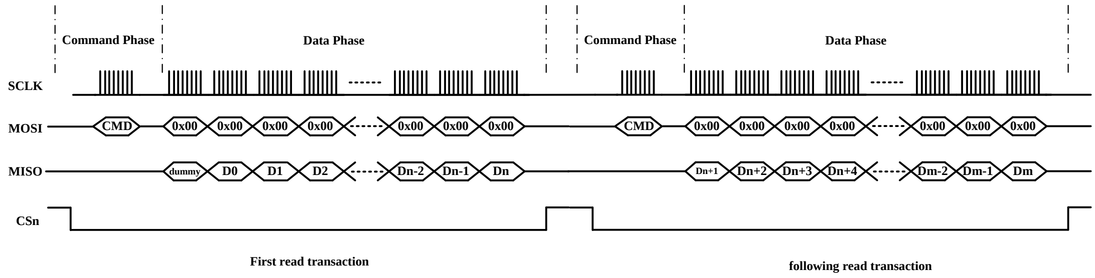

Figure 5 Burst read timing diagram 1

For hardware platforms (ArduCAM-Shield-V2, ArduCAM-Mini-5MP-Plus), you don't need to worry about the first byte. Detail timing can be found from Figure 6.

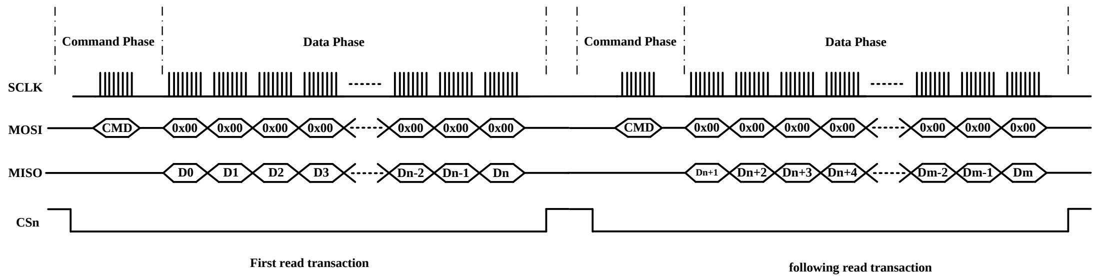

Figure 6 Burst read timing diagram 2

5.6 Rewind Read Operation

Rewind read is useful for some application that need access the same pixel data multiple times. By sending the command code 0x84 and write ‘1’ to bit[5] in the data phase, it will reset the memory read pointer to ZERO. Then user can read the image data from the start of the memory.

5.7 Low Power Mode

For some battery powered device power consumption is very important. There are two levels to achieve low power mode, user have to combine these modes according to their own power strategy.

5.7.1 Power down the sensor circuit

It is achieved by controlling the power enable pin of the onboard LDOs. The power enable pin is controlled by the GPIO[2] of ArduChip. By sending the command code 0x86 and write ‘1’ to bit[2] to enable the LDOs, or write ‘0’ to bit[2] to disable the LDOs to save power. Note that power down the sensor circuit, the camera settings are lost. User should reinitialize the sensor when power up the sensor circuit again.

5.7.2 Sensor standby

It is achieved by controlling the power enable pin of the onboard LDOs. The power enable pin is controlled by the GPIO[1] of ArduChip. By sending the command code 0x86 and write ‘1’ to bit[1] to set the sensor into standby mode, or write ‘0’ to bit[1] to set the sensor out of standby mode. Note that the sensor settings are not lost when in standby mode, and reinitialize is not needed.

6 ArduCAM APIs

There are a set of API functions that issue different commands to ArduCAM shield.

6.1 void InitCAM (void)

InitCAM function initializes the hardware information of the user system, such as the SPI chip select port initialization and image sensor slave address initialization.

6.2 void flush_fifo (void)

flash fifo function is used to reset the fifo read pointer to ZERO.

6.3 void start_capture (void)

start_capture function is used to issue a capture command. After this command the ArduCAM hardware will wait for a start of a new frame then store the entire frame data to onboard frame buffer.

6.4 void clear_fifo_flag (void)

Once a frame image is buffed to onboard memory, the capture completion flag is asserted automatically. The clear_fifo_flag function is used to clear this flag before issuing next capture command.

6.5 void write_reg(uint8_t addr, uint8_t data)

Param1: ArduChip register address (or command code)

Param2: data to be written into the register

write_reg is a basic function to write the ArduChip internal registers.

6.6 uint8_t read_reg(uint8_t addr)

Param1: ArduChip register address (or command code)

Return value: register value

read_reg is a basic function to read ArduChip internal register value.

6.7 uint32_t read_fifo_length(void)

Return value: 32 bit length of captured image

read_fifo_length function is used to determine the length of current captured image. Note the Rev.C shield doesn't support this feature.

6.8 void set_fifo_burst(void)

set_fifo_burst function is used to set the read memory into burst read mode. It should be called before burst memory read operation. Note the Rev.C shield doesn't support this feature.

6.9 int wrSensorRegs8_8(const struct sensor_reg*)

Param1: sensor setting data array

Return value: error status

wrSensorRegs8_8 function is used to write array of settings into sensor’s internal register over I2C interface and sensor’s register is accessed with 8bit address and 8bit data.

6.10 int wrSensorRegs8_16(const struct sensor_reg*)

Param1: sensor setting data array

Return value: error status

wrSensorRegs8_16 function is used to write array of settings into sensor’s internal register over I2C interface and sensor’s register is accessed with 8bit address and 16bit data.

6.11 int wrSensorRegs16_8(const struct sensor_reg*)

Param1: sensor setting data array

Return value: error status

wrSensorRegs16_8 function is used to write array of settings into sensor’s internal register over I2C interface and sensor’s register is accessed with 16bit address and 8bit data.

6.12 int wrSensorRegs16_16(const struct sensor_reg*)

Param1: sensor setting data array

Return value: error status

wrSensorRegs16_16 function is used to write array of settings into sensor’s internal register over I2C interface and sensor’s register is accessed with 16bit address and 16bit data.

6.13 byte wrSensorReg8_8(int regID, int regDat)

Param1: sensor internal register address

Param2: value to be written into the register

Return value: error status

wrSensorReg8_8 function is used to write a single sensor’s internal register over I2C interface and sensor’s register is accessed with 8bit address and 8bit data.

6.14 byte wrSensorReg8_16(int regID, int regDat)

Param1: sensor internal register address

Param2: value to be written into the register

Return value: error status

wrSensorReg8_16 function is used to write a single sensor’s internal register over I2C interface and sensor’s register is accessed with 8bit address and 16bit data.

6.15 byte wrSensorReg16_8(int regID, int regDat)

Param1: sensor internal register address

Param2: value to be written into the register

Return value: error status

wrSensorReg16_8 function is used to write a single sensor’s internal register over I2C interface and sensor’s register is accessed with 16bit address and 8bit data.

6.16 byte wrSensorReg16_16(int regID, int regDat)

Param1: sensor internal register address

Param2: value to be written into the register

Return value: error status

wrSensorReg16_16 function is used to write a single sensor’s internal register over I2C interface and sensor’s register is accessed with 16bit address and 16bit data.

6.17 byte rdSensorReg8_8(uint8_t regID, uint8_t* regDat)

Param1: sensor internal register address

Param2: value read from the register

Return value: error status

rdSensorReg8_8 function is used to read a single sensor’s internal register value over I2C interface and sensor’s register is accessed with 8bit address and 8bit data.

6.18 byte rdSensorReg16_8(uint16_t regID, uint8_t* regDat)

Param1: sensor internal register address

Param2: value read from the register

Return value: error status

rdSensorReg16_8 function is used to read a single sensor’s internal register value over I2C interface and sensor’s register is accessed with 16bit address and 8bit data.

6.19 byte rdSensorReg8_16(uint8_t regID, uint16_t* regDat)

Param1: sensor internal register address

Param2: value read from the register

Return value: error status

rdSensorReg8_16 function is used to read a single sensor’s internal register value over I2C interface and sensor’s register is accessed with 8bit address and 8bit data.

6.20 byte rdSensorReg16_16(uint16_t regID, uint16_t* regDat)

Param1: sensor internal register address

Param2: value read from the register

Return value: error status

rdSensorReg16_16 function is used to read a single sensor's internal register value over I2C interface and sensor's register is accessed with 16bit address and 16bit data.

6.21 void OV2640_set_JPEG_size(uint8_t size)

Param1: resolution code

OV2640_set_JPEG_size function is used to set the desired resolution with JPEG format for OV2640. Current support resolution is shown as follows:

```
#define OV2640_160x120      0   //160x120
#define OV2640_176x144      1   //176x144
#define OV2640_320x240      2   //320x240
#define OV2640_352x288      3   //352x288
#define OV2640_640x480      4   //640x480
#define OV2640_800x600      5   //800x600
#define OV2640_1024x768     6   //1024x768
#define OV2640_1280x1024    7   //1280x1024
#define OV2640_1600x1200    8   //1600x1200
```

6.22 void OV5642_set_JPEG_size(uint8_t size)

Param1: resolution code

OV5642_set_JPEG_size function is used to set the desired resolution with JPEG format for OV5642. Current support resolution is shown as follows:

```
#define OV5642_320x240      0   //320x240
#define OV5642_640x480      1   //640x480
#define OV5642_1024x768     2   //1024x768
#define OV5642_1280x960     3   //1280x960
#define OV5642_1600x1200    4   //1600x1200
#define OV5642_2048x1536    5   //2048x1536
#define OV5642_2592x1944    6   //2592x1944
```

6.23 void OV5640_set_JPEG_size(uint8_t size)

Param1: resolution code

OV5640_set_JPEG_size function is used to set the desired resolution with JPEG format for OV564. Current support resolution is shown as follows:

```
#define OV5640_320x240      0   //320x240
#define OV5640_352x288      1   //352x288
#define OV5640_640x480      2   //640x480
#define OV5640_800x480      3   //800x480
#define OV5640_1024x768     4   //1024x768
#define OV5640_1280x960     5   //1280x960
#define OV5640_1600x1200    6   //1600x1200
#define OV5640_2048x1536    7   //2048x1536
#define OV5640_2592x1944    8   //2592x1944
```

6.24 void set_format(byte fmt)

set_format function is used to set the sensor between RGB mode and JPEG mode. The

InitCAM function should be called after set_format function.

7 Registers Table

Sensor and FIFO timing is controlled with a set of registers which is implemented in the ArduChip. User can send capture commands and read image data with a simple SPI slave interface. The detail description of registers' bits can be found in the software section in this document. Not all the registers are implemented in a given hardware platform, please check the hardware develop guide for detail register description for certain hardware you've got.

As mentioned earlier the first bit[7] of the command phase is read/write byte, '0' is for read and '1' is for write, and the bit[6:0] is the address to be read or write in the data phase. So user has to combine the 8 bits address according to the read or write commands they want to issue.

Table 1 ArduChip Register Table

Register Address bit[6:0]	Register Type	Description
0x00	RW	Test Register
0x01	RW	Capture Control Register Bit[2:0]: Number of frames to be captured The value in this register + 1 equal to the number of frames to be captured. The value=7 means capture continuous frames until the frame buffer is full, it is used for short video clip recording.
0x02	RW	Bus Mode Determine who is owner of the data bus, only one owner is allowed. Bit[7:2]: Reserved Bit[1]: Camera write LCD bus Bit[0]: MCU write LCD bus
0x03	RW	Sensor Interface Timing Register Bit[0]: Sensor Hsync Polarity, 0 = active high, 1 = active low Bit[1]: Sensor Vsync Polarity 0 = active high, 1 = active low Bit[2]: LCD backlight enable 0 = enable, 1 = disable Bit[3]: Sensor PCLK reverse 0 = normal, 1= reversed PCLK
0x04	RW	FIFO control Register Bit[0]: write '1' to clear FIFO write done flag Bit[1]: write '1' to start capture Bit[4]: write '1' to reset FIFO write pointer Bit[5]: write '1' to reset FIFO read pointer
0x05	RW	GPIO Direction Register Bit[0]: Sensor reset IO direction

		Bit[1]: Sensor power down IO direction Bit[2]: Sensor power enable IO direction 0 = input, 1 = output
0x06	RW	GPIO Write Register Bit[0]: Sensor reset IO value Bit[1]: Sensor power down IO value Bit[2]: Sensor power enable IO value
0x3B	RO	Reserved
0x3C	RO	Burst FIFO read operation
0x3D	RO	Single FIFO read operation
0x3E	WO	LCD control register with RS=0
0x3F	WO	LCD control register with RS=1
0x40	RO	ArduChip version Bit[7:4]: integer part of the revision number Bit[3:0]: decimal part of the revision number
0x41	RO	Bit[0]: camera vsync pin status Bit[3]: camera write FIFO done flag
0x42	RO	Camera write FIFO size[7:0]
0x43	RO	Camera write FIFO size[15:8]
0x44	RO	Camera write FIFO size[22:16]
0x45	RO	GPIO Read Register Bit[0]: Sensor reset IO value Bit[1]: Sensor power down IO value Bit[2]: Sensor power enable IO value