ArduCAM Camera Shield

SPI Camera Software Application Note

Rev 1.0, June 2015

Table of Contents

1 Introduction

This application note describes the detail software operation of ArduCAM camera shield. The latest source code library 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.

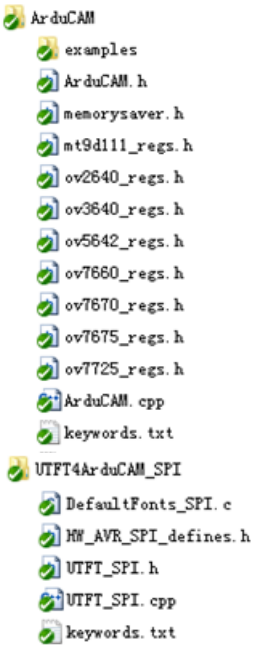

Figure 1 ArduCAM Libraries Structure

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

Once the ArduCAM libraries are copied to Arduino libraries directory, the ArduCAM examples can be found from the menu File->Examples->ArduCAM as the Figure 2 shown.

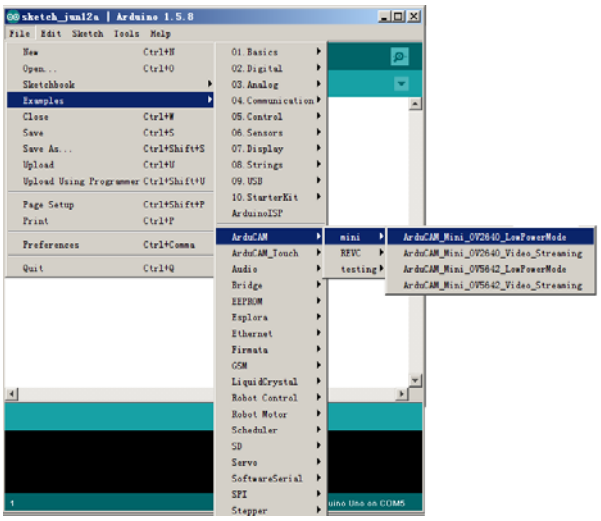

Figure 2 Arduino IDE examples

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.

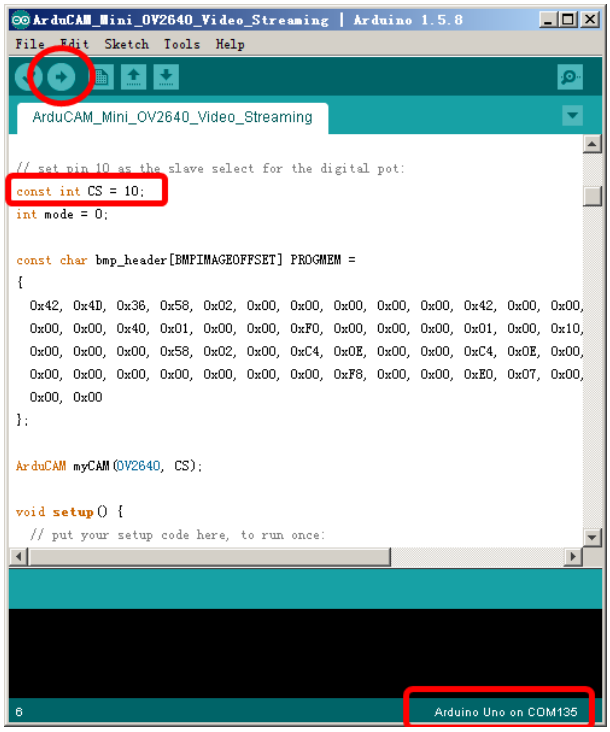

Figure 3 Example Sketch

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 three sub directories for different ArduCAM models as the Figure 4 shown.

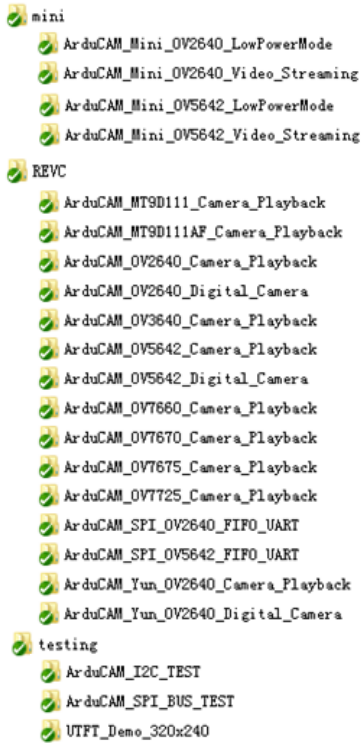

Figure 4 Example Folder Structure

4.1 ArduCAM Mini Examples

The mini folder contains examples for ArduCAM Mini shields. All of the examples use burst fifo read operation in order to increase the performance.

4.1.1 ArduCAM_Mini_OVxxxx_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.

4.1.2 ArduCAM_Mini_OVxxxx_LowPowerMode

This example 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.

4.2 ArduCAM REVC Examples

The REVC folder contains examples for ArduCAM Rev.C shield. It requires additional UTFT4ArduCAM_SPI library as mentioned earlier. And it doesn't support new features like the ArduCAM Mini does.

4.2.1 ArduCAM_xxxx_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.

4.2.2 ArduCAM_xxxx_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.

4.2.3 ArduCAM_SPI_xxxx_FIFO_UART

This example is used to transfer the captured image over UART serial port. User can use this example upload the captured image to host apps for viewing or post processing.

4.3 Testing Examples

The testing folder contains the testing code for basic hardware. It is useful if coming across hardware problems.

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.

ArduChip Mini shield now offers series of new feature, which is not available for Rev.C shield.

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 (New feature)

. 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 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.

5.4 Burst Read Operation (New feature)

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 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.

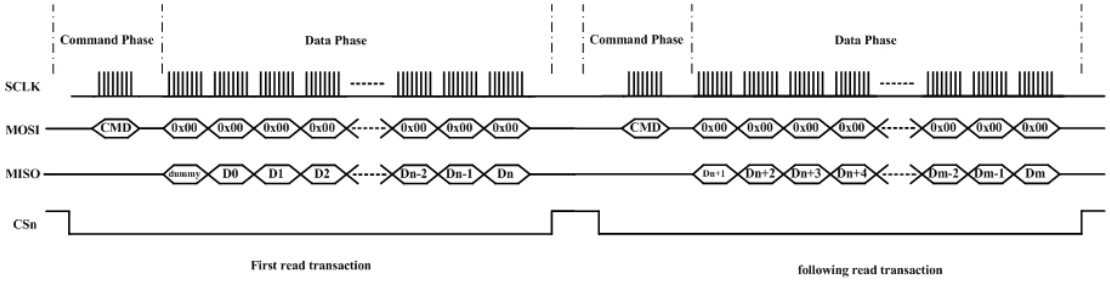

Figure 5 Burst read timing diagram

5.5 Rewind Read Operation (New feature)

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.

5.6 Low Power Mode (New feature)

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

5.6.1 Power down the memory controller circuit

By sending command code 0x83 and write '1' to bit[6], the memory controller circuit will be powered down, and the data will be lost in the frame buffer.

5.6.2 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.

5.6.3 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 ArduCAM 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 ArduCAM 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. It is only support ArduCAM Mini shield, the Rev.C shield is not supported.

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. It is only support ArduCAM Mini shield, the Rev.C shield is not supported.

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	1	//320x240
#define OV5642_640x480	2	//640x480
#define OV5642_1280x720	3	//1280x720
#define OV5642_1920x1080	4	//1920x1080
#define OV5642_2048x1563	5	//2048x1563
#define OV5642_2592x1944	6	//2592x1944

6.23 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.

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
0x02	RW	Reserved
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[3]: Sensor data delay 0 = no delay, 1= delay 1 PCLK Bit[4]: FIFO mode control 0 = FIFO mode disable, 1 = enable FIFO mode Bit[6]: low power mode control 0 = normal mode, 1 = low power mode
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	RO	Reserved
0x3F	RO	Reserved
0x40	RO	ArduChip version, constant value 0x40 for 2MP model 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[18:16]
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