

**ArduCAM Camera Shield Series**

SPI Camera Software Application Note

Rev 2.0, Oct 2016



Table of Contents

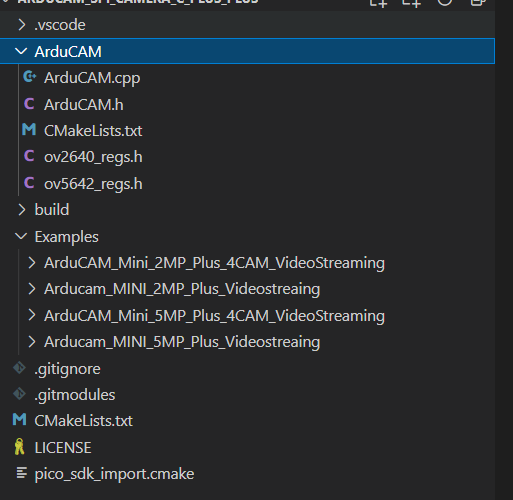
1. [Introduction 3](#_TOC_250068)
2. [Software Library Structure 3](#_TOC_250067)
3. [Quick Start Guide 3](#_TOC_250066)
4. [Example Sketches 5](#_TOC_250065)
   1. [ArduCAM Mini Examples 5](#_TOC_250064)
      1. [ArduCAM\_Mini\_XXX\_Plus\_VideoStreaming 5](#_TOC_250063)
      2. [ArduCAM\_MINI\_XXX\_Plus\_4CAM\_VideoSteaming 5](#_TOC_250062)
5. [ArduChip Functions 7](#_TOC_250035)
   1. [Single Capture Mode 8](#_TOC_250034)
   2. [Multiple Capture Mode 8](#_TOC_250033)
   3. [Short Video Capture Mode 8](#_TOC_250032)
   4. [Single Read Operation 8](#_TOC_250031)
   5. [Burst Read Operation 8](#_TOC_250030)
   6. [Rewind Read Operation 9](#_TOC_250029)
   7. [Low Power Mode 9](#_TOC_250028)
      1. [Power down the sensor circuit 9](#_TOC_250027)
      2. [Sensor standby 9](#_TOC_250026)
6. [ArduCAM APIs 9](#_TOC_250025)
   1. [void InitCAM (void) 9](#_TOC_250024)
   2. [void flush\_fifo (void) 10](#_TOC_250023)
   3. [void start\_capture (void) 10](#_TOC_250022)
   4. [void clear\_fifo\_flag (void) 10](#_TOC_250021)
   5. [void write\_reg(uint8\_t addr, uint8\_t data) 10](#_TOC_250020)
   6. [uint8\_t read\_reg(uint8\_t addr) 10](#_TOC_250019)
   7. [uint32\_t read\_fifo\_length(void) 10](#_TOC_250018)
   8. [void set\_fifo\_burst(void) 10](#_TOC_250017)
   9. [int wrSensorRegs8\_8(const struct sensor\_reg\*) 10](#_TOC_250016)
   10. [int wrSensorRegs8\_16(const struct sensor\_reg\*) 10](#_TOC_250015)
   11. [int wrSensorRegs16\_8(const struct sensor\_reg\*) 10](#_TOC_250014)
   12. [int wrSensorRegs16\_16(const struct sensor\_reg\*) 10](#_TOC_250013)
   13. [byte wrSensorReg8\_8(int regID, int regDat) 11](#_TOC_250012)
   14. [byte wrSensorReg8\_16(int regID, int regDat) 11](#_TOC_250011)
   15. [byte wrSensorReg16\_8(int regID, int regDat) 11](#_TOC_250010)
   16. [byte wrSensorReg16\_16(int regID, int regDat) 11](#_TOC_250009)
   17. [byte rdSensorReg8\_8(uint8\_t regID, uint8\_t\* regDat) 11](#_TOC_250008)
   18. [byte rdSensorReg16\_8(uint16\_t regID, uint8\_t\* regDat) 11](#_TOC_250007)
   19. [byte rdSensorReg8\_16(uint8\_t regID, uint16\_t\* regDat) 11](#_TOC_250006)
   20. [byte rdSensorReg16\_16(uint16\_t regID, uint16\_t\* regDat) 11](#_TOC_250005)
   21. [void OV2640\_set\_JPEG\_size(uint8\_t size) 12](#_TOC_250004)
   22. [void OV5642\_set\_JPEG\_size(uint8\_t size) 12](#_TOC_250003)
   23. [void set\_format(byte fmt) 12](#_TOC_250001)
7. [Registers Table 13](#_TOC_250000)

# 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.](https://github.com/arducam)

# Software Library Structure

The ARDCAM library is designed for the PICO platform.



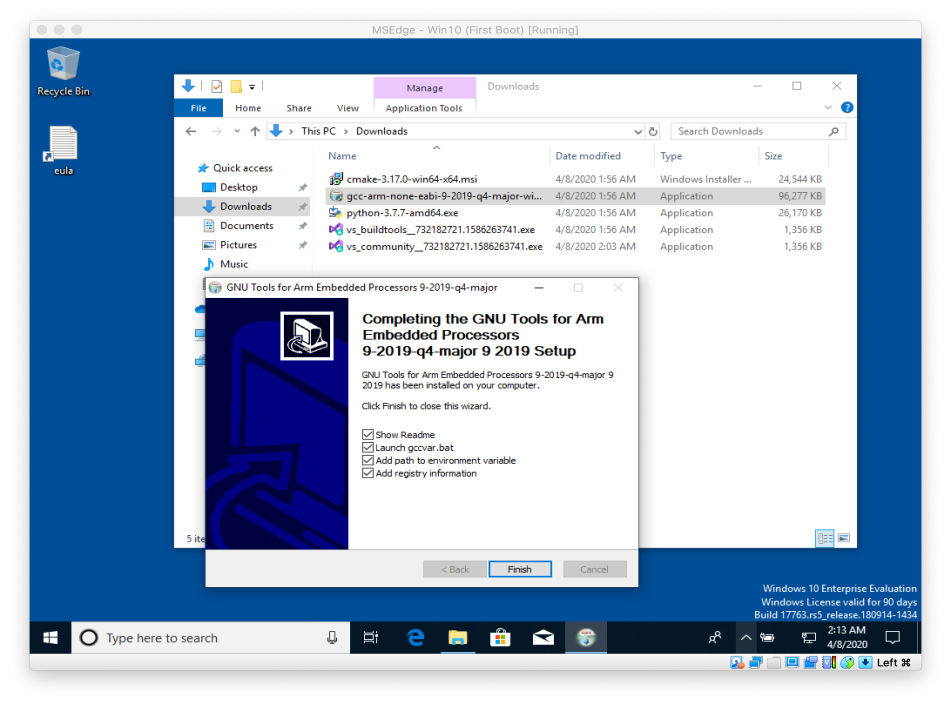
# Quick Start Guide

**Installing the Toolchain**

To build you will need to install some extra tools.

* + - * [ARM GCC compiler](https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads)
      * [CMake](https://cmake.org/download/)
      * [Build Tools for Visual Studio 2019](https://visualstudio.microsoft.com/downloads/#build-tools-for-visual-studio-2019)
      * [Python 3.9](https://www.python.org/downloads/windows/)
      * [Git](https://git-scm.com/download/win)

## 3.1 Installing ARM GCC Compiler



During installation you should tick the box to register the path to the ARM compiler as an environment variable in the Windows shell when prompted to do so.

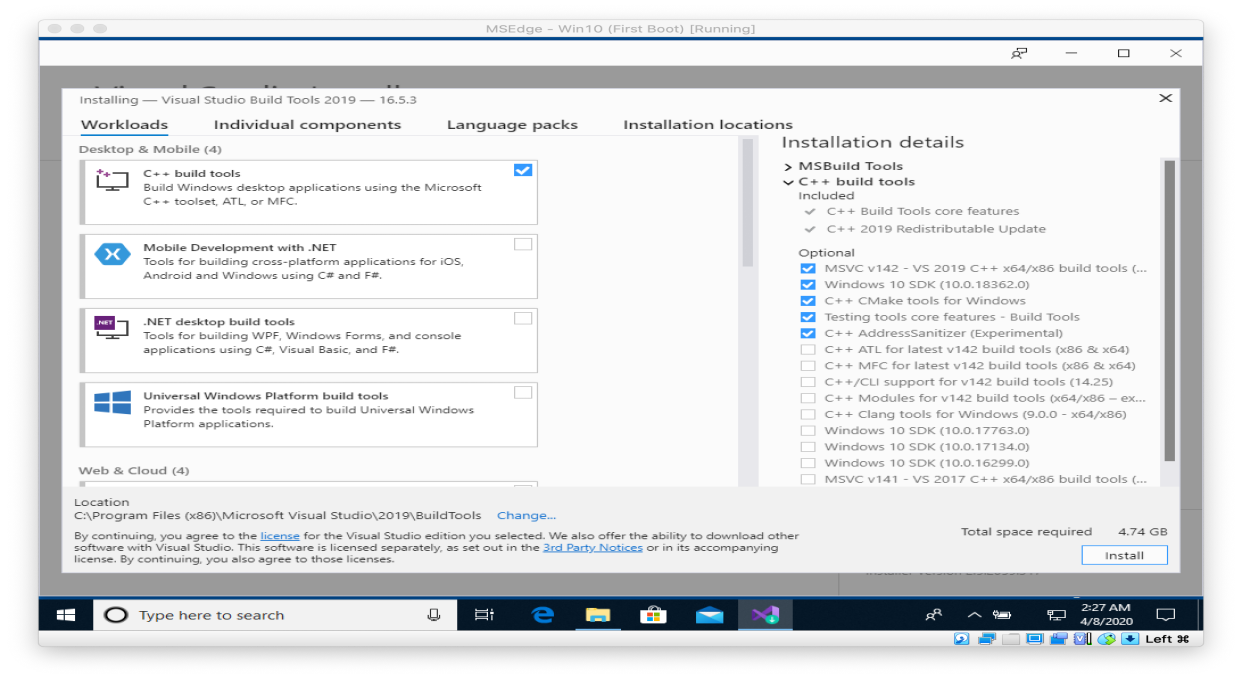
## 3.2 Installing CMake

**WARNING**

|  |
| --- |
| There’s a bug in the Release Candidate (rc) versions of CMake 3.20 which means you need to delete your entire  build directory before you can do a subsequent build. CMake 3.19 doesn’t contain this bug and works fine. |

During the installation add CMake to the system PATH for all users when prompted by the installer.

## 3.3 Installing Build Tools for Visual Studio 2019



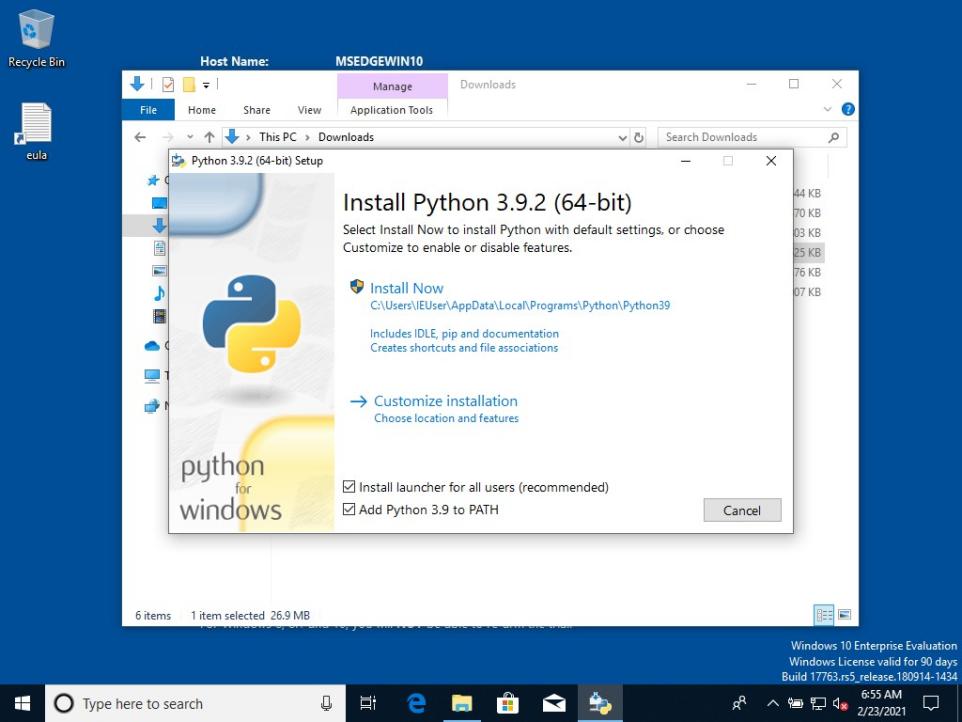
When prompted by the Build Tools for Visual Studio installer you need to install the C++ build tools only.

**NOTE**

You must install the full "Windows 10 SDK" package as the SDK will need to build the pioasm and elf2uf2 tools locally. Removing it from the list of installed items will mean that you will be unable to build Raspberry Pi Pico binaries.

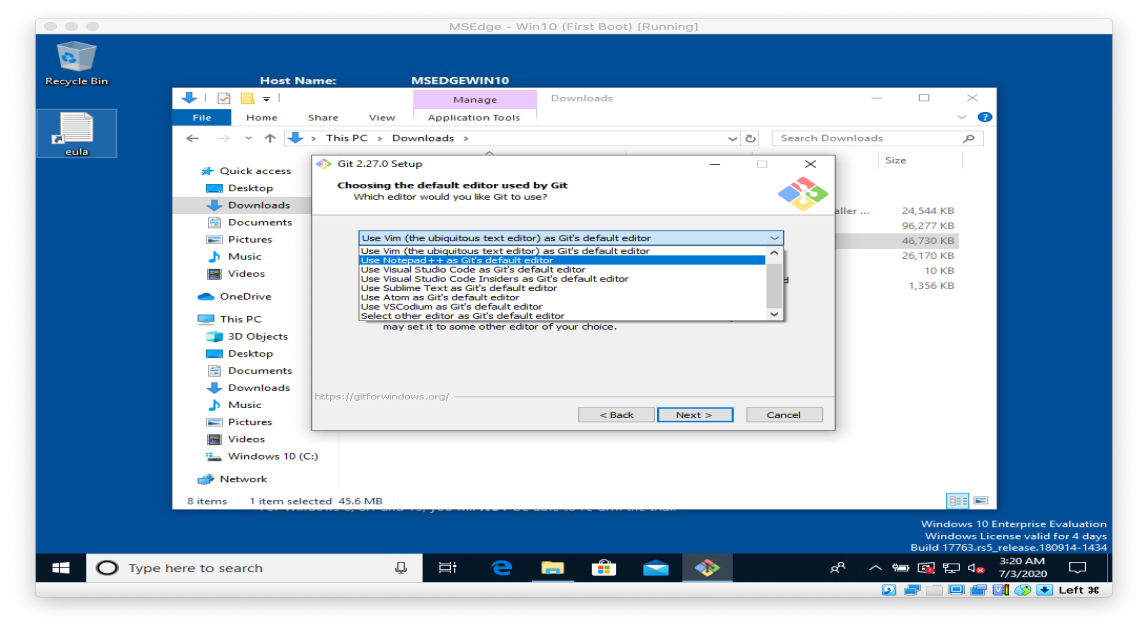
## 3.4 Installing Python 3.9

During the installation, ensure that it’s installed 'for all users' and add Python 3.9 to the system PATH when prompted by the installer. You should additionally disable the MAX\_PATH length limit when prompted at the end of the Python installation.



## 3.6 Installing Git

When installing Git you should ensure that you change the default editor away from vim, see [Figure 18](#_bookmark58).



Ensure you tick the checkbox to allow Git to be used from third-party tools and, unless you have a strong reason otherwise, when installing Git you should also check the box "Checkout as is, commit as-is", select "Use Windows' default console window", and "Enable experimental support for pseudo consoles" during the installation process.

**3.5.1 Getting the SDK and examples**

C:\Users\pico\Downloads> git clone -b master https://github.com/raspberrypi/pico-sdk.git C:\Users\pico\Downloads> cd pico-sdk

C:\Users\pico\Downloads\pico-sdk> git submodule update --init C:\Users\pico\Downloads\pico-sdk> cd ..

C:\Users\pico\Downloads> git clone -b master https://github.com/raspberrypi/pico-examples.git

**3.5.2 Building "Hello World" from the Command Line**

Go ahead and open a Developer Command Prompt Window from the Windows Menu, by selecting Windows > Visual Studio 2019 > Developer Command Prompt from the menu.

Then set the path to the SDK as follows,

C:\Users\pico\Downloads> setx PICO\_SDK\_PATH "..\..\pico-sdk"

You now need **close your current Command Prompt Window** and open a second Command Prompt Window where this environment variable will now be set correctly before proceeding.

Navigate into the pico-examples folder, and build the 'Hello World' example as follows,

C:\Users\pico\Downloads> cd pico-examples C:\Users\pico\Downloads\pico-examples> mkdir build C:\Users\pico\Downloads\pico-examples> cd build

C:\Users\pico\Downloads\pico-examples\build> cmake -G "NMake Makefiles" .. C:\Users\pico\Downloads\pico-examples\build> nmake

to build the target. This will produce ELF, bin, and uf2 targets, you can find these in the hello\_world/serial and hello\_world/usb directories inside your build directory. The UF2 binaries can be dragged-and-dropped directly onto a RP2040 board attached to your computer using USB.

**3.5.3 Building "Hello World" from Visual Studio Code**

Now you’ve installed the toolchain you can install [Visual Studio Code](https://code.visualstudio.com/download) and build your projects inside the that environment rather than from the command line.

Go ahead and [download](https://code.visualstudio.com/download) and install Visual Studio Code for Windows. After installation open a Developer Command Prompt Window from the Windows Menu, by selecting Windows > Visual Studio 2019 > Developer Command Prompt from the menu. Then type,

at the prompt. This will open Visual Studio Code with all the correct environment variables set so that the toolchain is correctly configured.

C:> code

**WARNING**

If you start Visual Studio code by clicking on its desktop icon, or directly from the Start Menu then the build environment will **not** be correctly configured. Although this can be done manually later in the CMake Tools Settings, the easiest way to configure the Visual Studio Code environment is just to open it from a Developer Command Prompt Window where these environmental variables are already set.

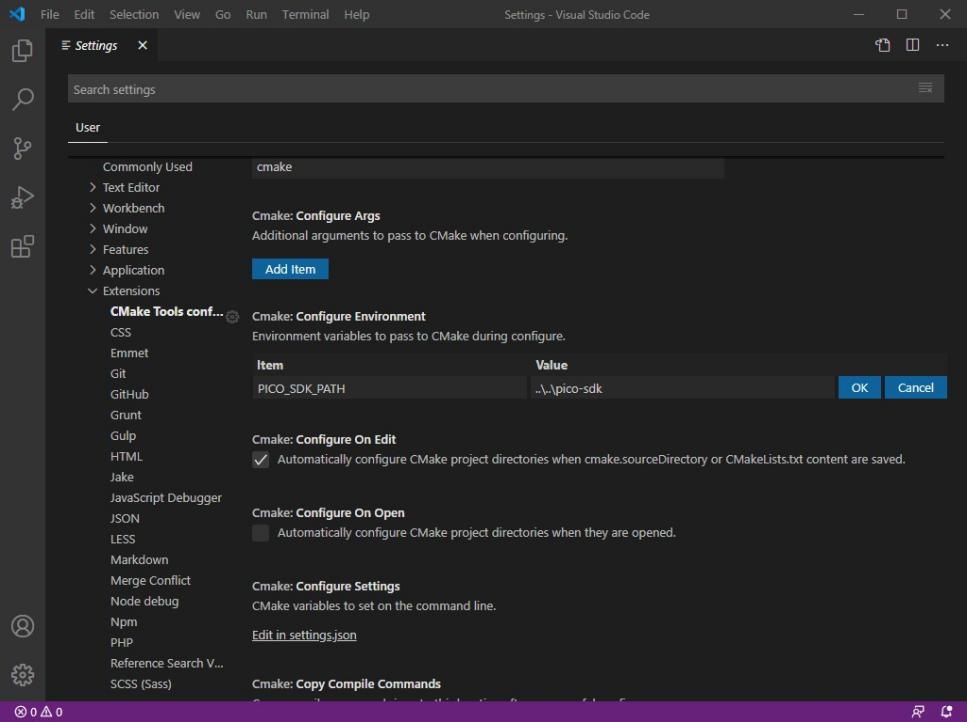
We’ll now need to install the [CMake Tools](https://marketplace.visualstudio.com/items?itemName=ms-vscode.cmake-tools) extension. Click on the Extensions icon in the left-hand toolbar (or type Ctrl + Shift + X), and search for "CMake Tools" and click on the entry in the list, and then click on the install button.

Then click on the Cog Wheel at the bottom of the navigation bar on the left-hand side of the interface and select "Settings". Then in the Settings pane click on "Extensions" and the "CMake Tools configuration". Then scroll down to "Cmake: Configure Environment". Click on "Add Item" and add set the PICO\_SDK\_PATH to be ..\..\pico-sdk as in [Figure 19](#_bookmark62).

*Figure 19. Setting*

PICO\_SDK\_PATH

*Environment Variable in the CMake Extension*



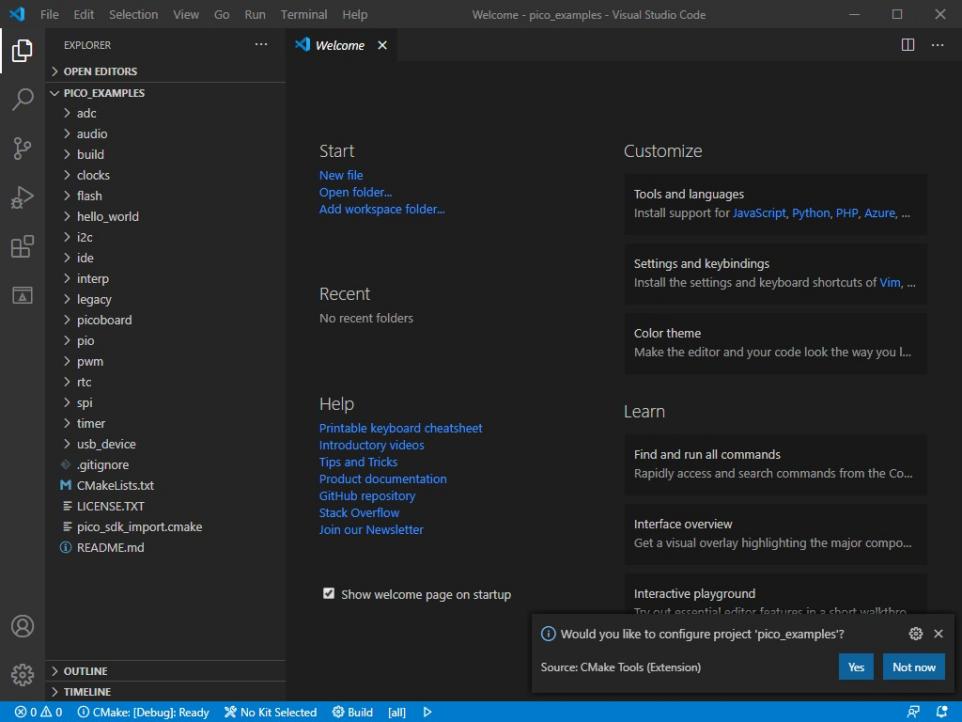
Additionally you will need to scroll down to "Cmake: Generator" and enter "NMake Makefiles" into the box.

IMPORTANT

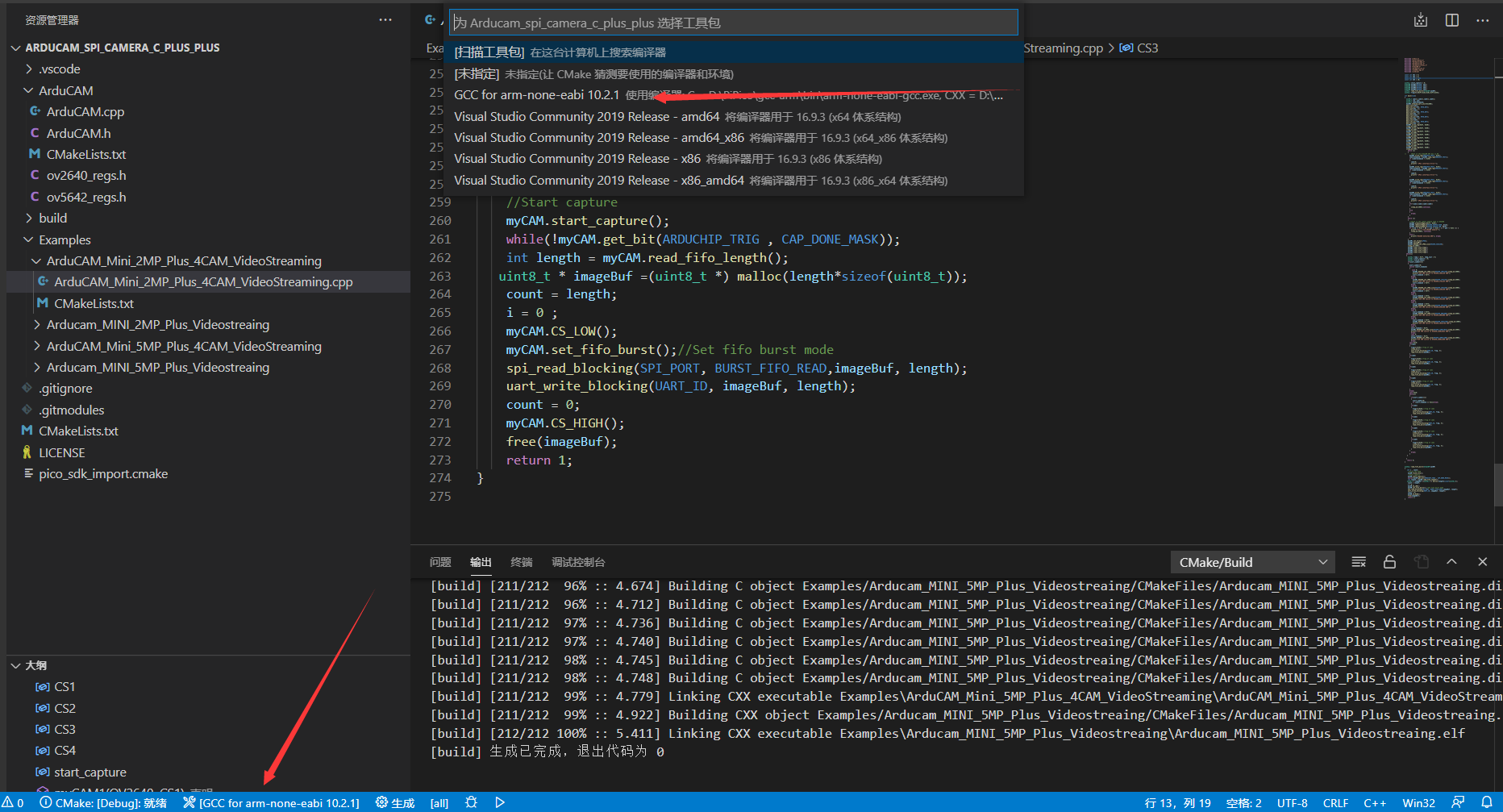
If you do not change the "Cmake: Generator" Visual Studio will default to ninja and the build might fail.

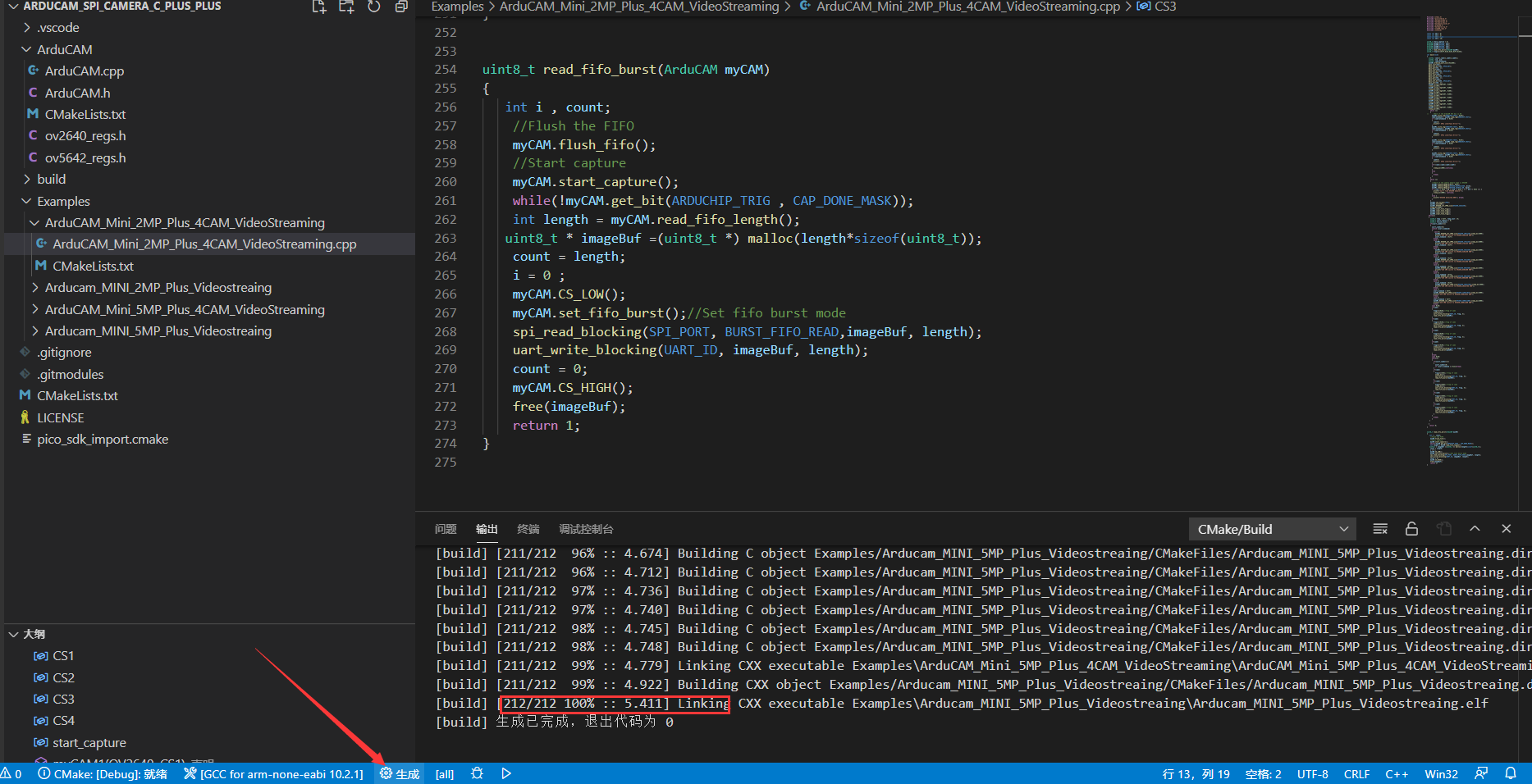
Now close the Settings page and go to the File menu and click on "Open Folder" and navigate to pico-examples repo and hit "Okay". You’ll be prompted to configure the project, see [Figure 20](#_bookmark63). Select "GCC for arm-none-eabi" for your compiler.

*Figure 20. Prompt to configure your project in Visual Studio Code*



**Open the project and compile the project as follows.**





# 4 Example Sketches

In the sample folder, there are four host applications, as shown in the figure.

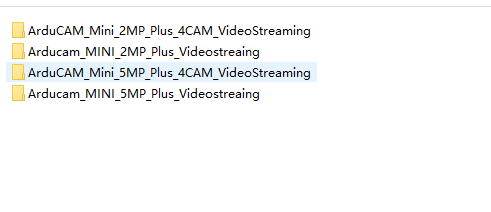


Figure 4 Example Folder Structure

**4.1 ArduCAM\_Mini\_XXX\_Plus\_VideoStreaming**

This example illustrates how to send continues capture commands to Pico and transfer the JPEG image data back to host application via Pico 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](https://github.com/ArduCAM/Arduino/tree/master/ArduCAM/examples/host_app) to view the captured image

**4.2 ArduCAM\_MINI\_XXX\_Plus\_4CAM\_VideoStreaming**

This example demonstrates how to connect four Arducam-minis (2 or 5 megapixels) to the Arducam Multi-Camera Adapter Board and capture images via USB-Serial.This example should work with the host application.

# 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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Hardware Platform** | **Functions** | | | | | |
| Single Capture/  Read | Burst Read | Multiple Capture | Rewind | Low Power  Mode | Short Video  Capture |
| ArduCAM-Mini-2MP | √ | √ |  | √ | √ |  |
| ArduCAM-Mini-5MP-Plus  (OV5642) | √ | √ | √ | √ | √ | √ |

is a list of possible hardware platforms:

**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-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.

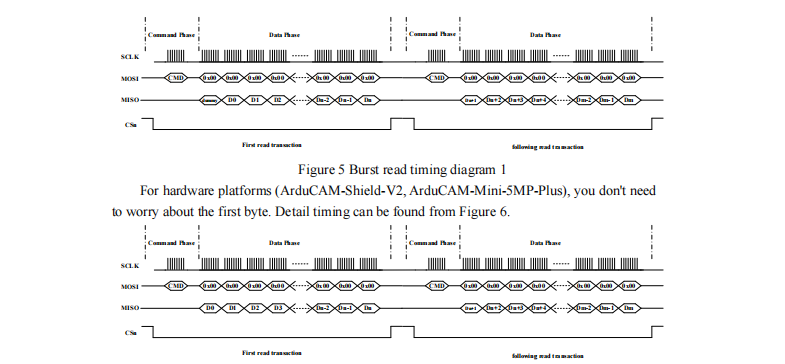


Figure 6 Burst read timing diagram 2

**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

ite\_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 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 |