# 1 基于 OV5640 的照相机实验

| | |
|---|---|
| 工程源码 | \|-ACZ702 v2.0 开发板<br>\|-- SD_Camera |
| 相关视频课程 | 无 |

**章节导读**

在本节中将运用 OV5640 摄像头、SD 卡和 LCD 等外设，初步实现图像采集功能。我们将简要介绍如何将 OV5640 采集到的图像存储到 SD 卡中，并步骤化地说明如何将帧缓存处理为 BMP 图片，方便浏览播放。本实验的目标是使读者对图片的采集和处理有一个基本的了解和初步的实践经验。

## 1.1 实验前言

本实验是基于 OV5640 图像采集显示系统进行配置。OV5640 章节已经涵盖了大部分所需知识，这些知识是理解并实践本次实验的重要基础。在硬件逻辑系统设计方面，本实验与以往的架构相比，并没有太大的改变，仅对部分 IP 核进行了精简和移除。因此，对于可能出现的不熟悉的知识点，推荐读者参考回顾 OV5640 章节。

在 CPU 软件程序设计部分，本次实验增加了一些新的功能，特别是在数据处理和图片保存方面，例如实现从 DDR 中抽取图片缓存并将其保存至 SD 卡中，这是一个重点内容。

## 1.2 硬件逻辑系统设计

本章节中，因为考虑到图片效果，去掉了 RGB565 保存，所以不再需要 rgb888to565 IP 核，图像数据直接使用 RGB888 格式存储。

本次设计将直接在 OV5640_LCD 工程的基础上构建，打开 OV5640_LCD 工程，选择菜单栏的 File->Project->Save As...，输入新的工程名后打开模块设计。

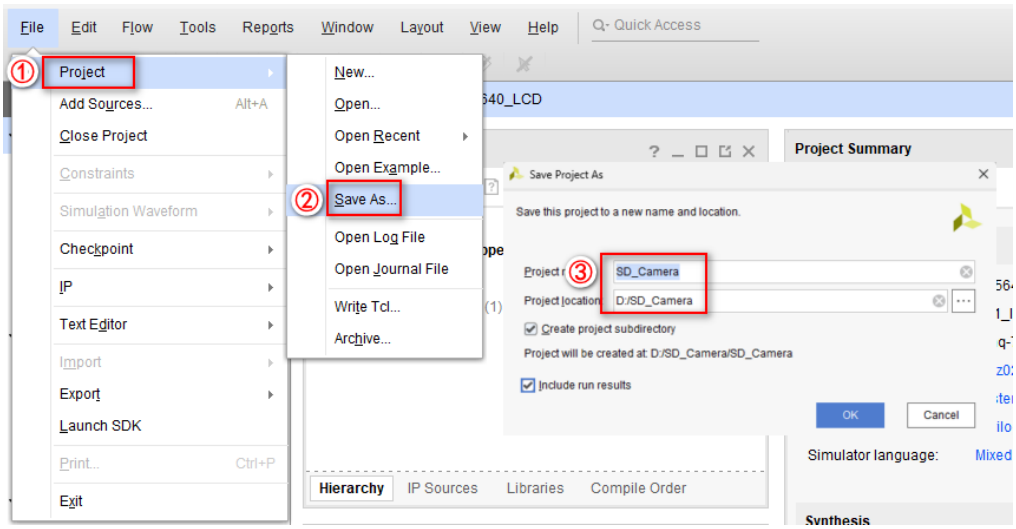


图 1-1 新工程

## 1.2.1 导入自定义 IP 核

打开文件管理器，将 OV5640_LCD 例程下的 ip_repo 文件夹复制到新建的工程目录下。例程位于 ACZ702 资料包中，相对路径如下：*小梅哥 ACZ702 型 Zynq 开发板资料\盘 A_ACZ702 开发板标准配套资料\02_设计实例\03_【裸机例程】基于 C 编程的 Zynq 裸机例程\ACZ70x0\OV5640_LCD*

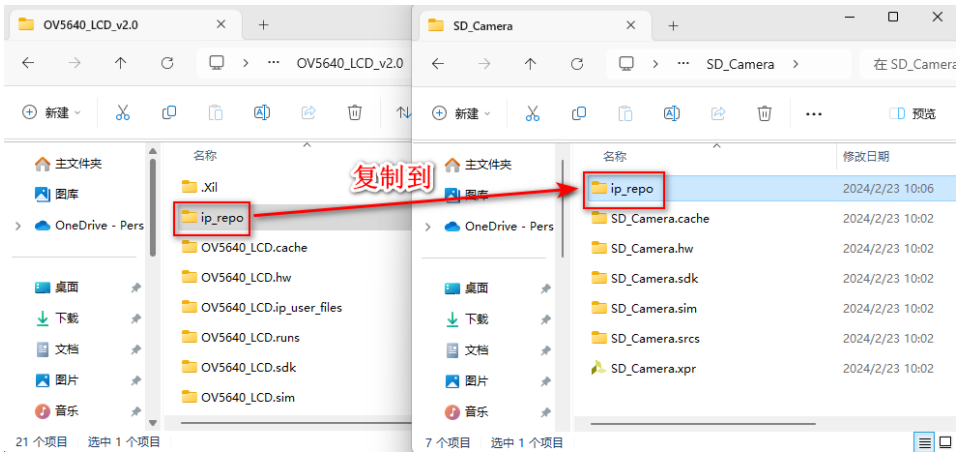


图 1-2 添加 ip_repo

接下来，如下图所示，将新工程中，残留的原工程 IP 存储库删除掉，添加上新拷贝的 ip_repo 文件夹路径。

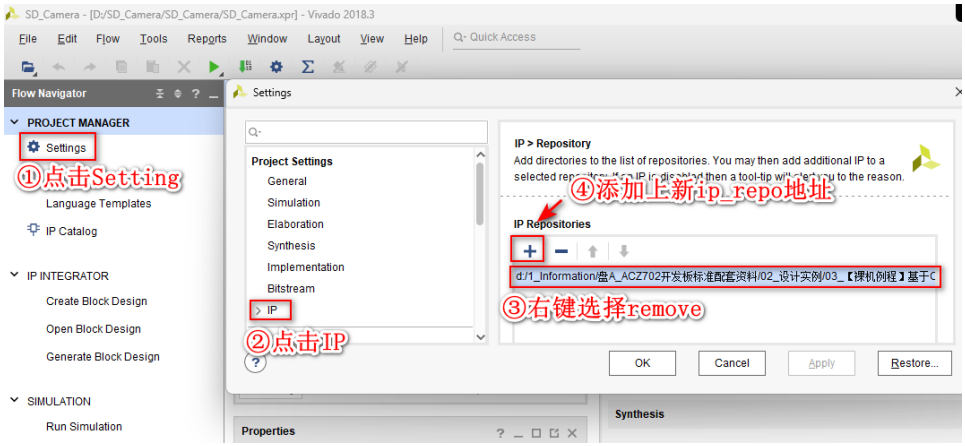


图 1-3 添加 ip_repo 路径

## 1.2.2 IP 核配置与端口连接

因为是从原工程中生成的新工程，此时，只需要简单修改几个地方即可

(1) Zynq IP 核配置

如下图所示，使能 SD 0、UART 1、GPIO。在读取 SD 卡过程中，可能出现 bug，此时可以通过 PS 串口打印出相关数据，方便分析（Uart 也可以不配置，如果出现错误再配置分析）。

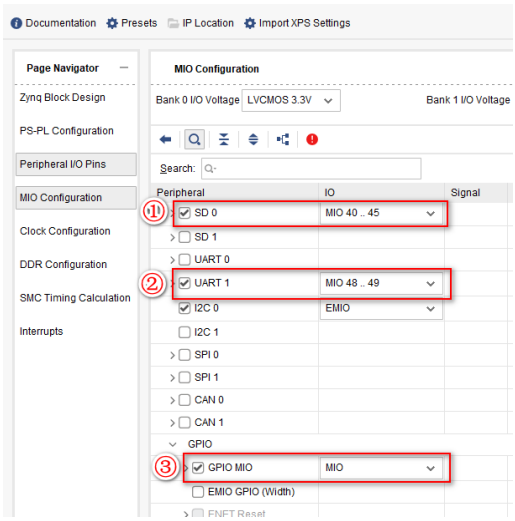


图 1-4 PL 时钟配置

(2) 删除 rgb888torgb565_0 与 rgb565torgb888_0 IP 核

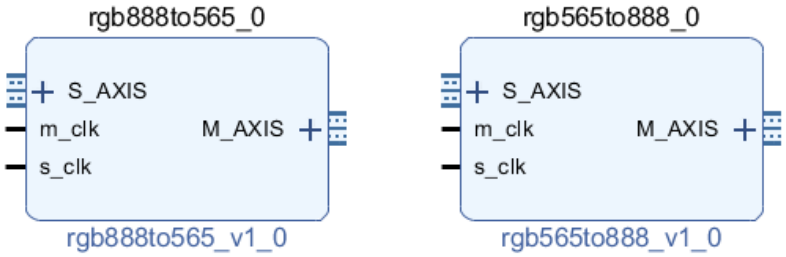


图 1-5 删除 IP 核

(3) 连接端口

将 video_out 端口（Vodeo In AXI4-Stream IP 核）连接到 S_AXIS_S2MM 端口（VDMA IP 核），可参考图连接。

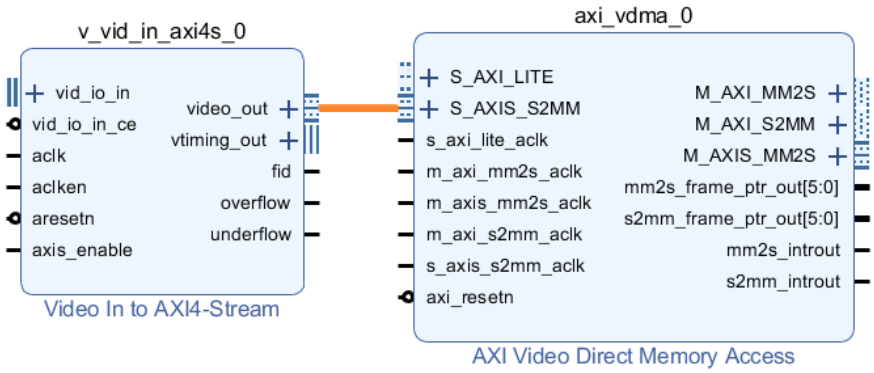


图 1-6 video_out 端口与 S_AXIS_S2MM 端口

将 M_AXIS_MM2S 端口（VDMA IP 核）连接到 video in（AXI4-Stream to Video Out IP 核）。

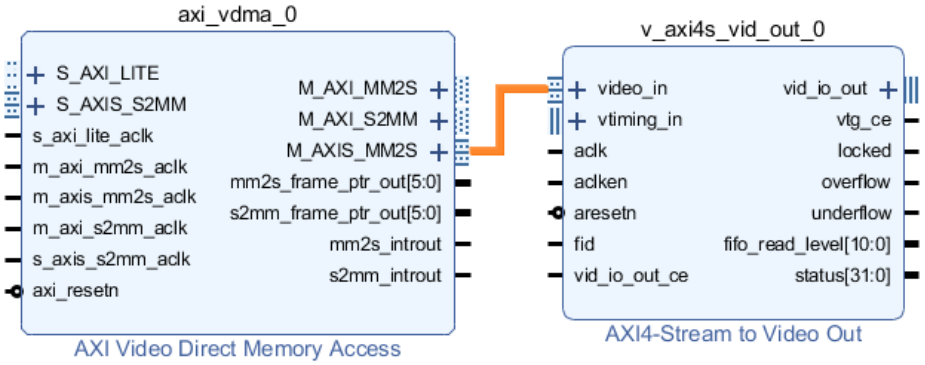


图 1-7 M_AXIS_MM2S 端口与 video in 端口

(4) VDMA IP 核配置

将 Stream Data Width 值修改为 24，Line Buffer Depth 值修改为 1024，如下图所示。

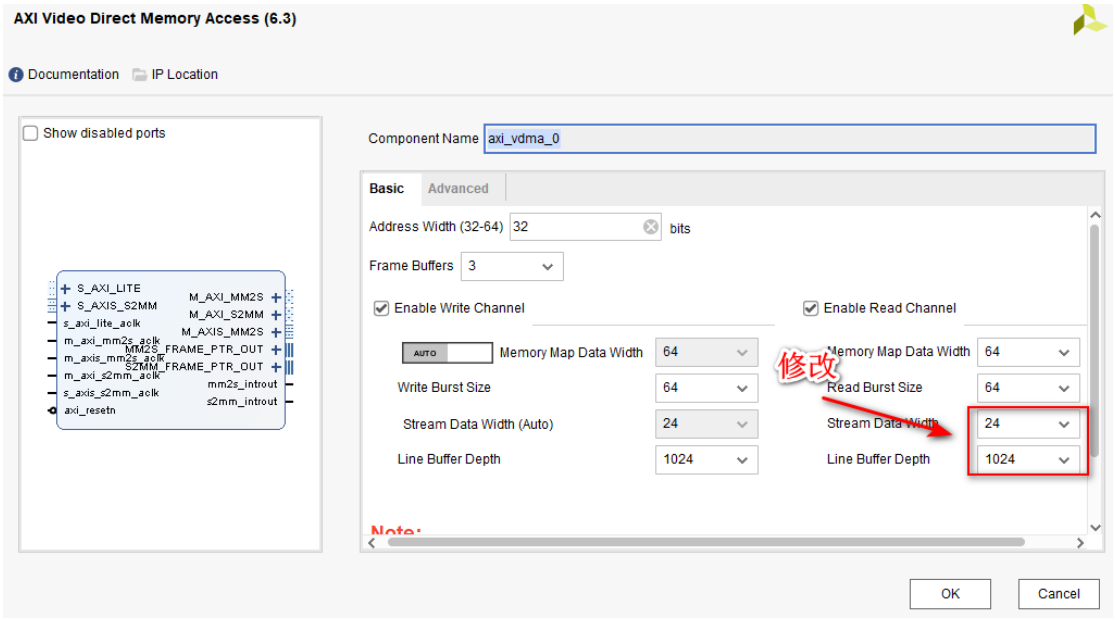


图 1-8 VDMA IP 核配置

配置完成后，点击 Regenerate Layout，重新生成布局，如图 1-9 所示。


图 1-9 重新生成布局

最后，验证设计，出现图 1-10 所示的“Validation successfu...”，说明无错误。

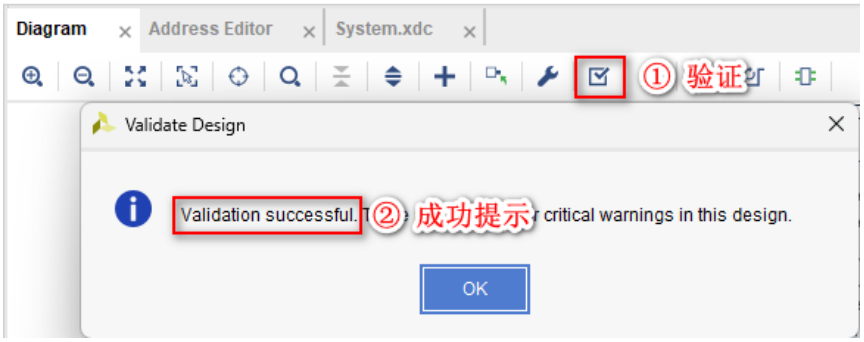


图 1-10 验证设计

## 1.2.3 生成封装

点击 sources 资源栏下我们创建的 system 模块设计，单击右键，在展开的功能中选择“Generate Output Products...”生成输出。

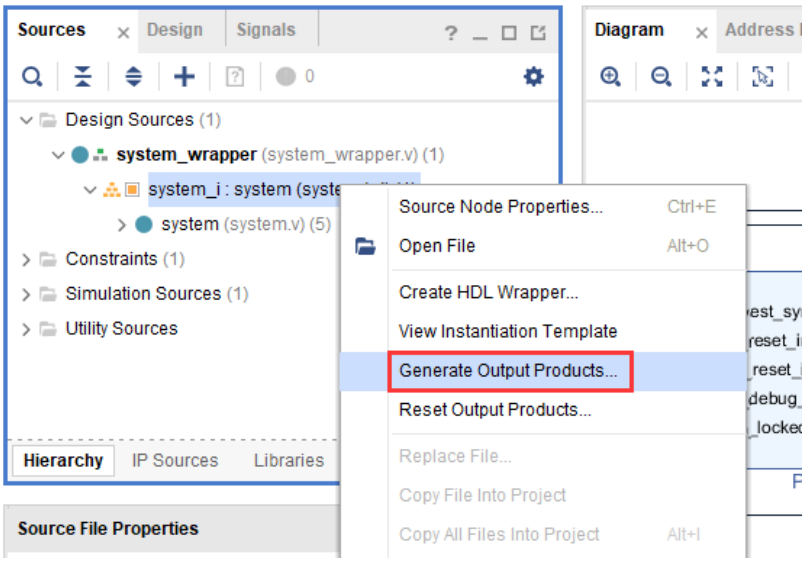


图 1-11 Generate Output Products

如图 1-12 所示，接下来软件会弹出生成输出前的设置界面。在合成选项栏直接选择“Out Of context per IP”即可，下方的“Number of jobs”选项选择最大值 16。设置完成后点击“Generate”开始生成。

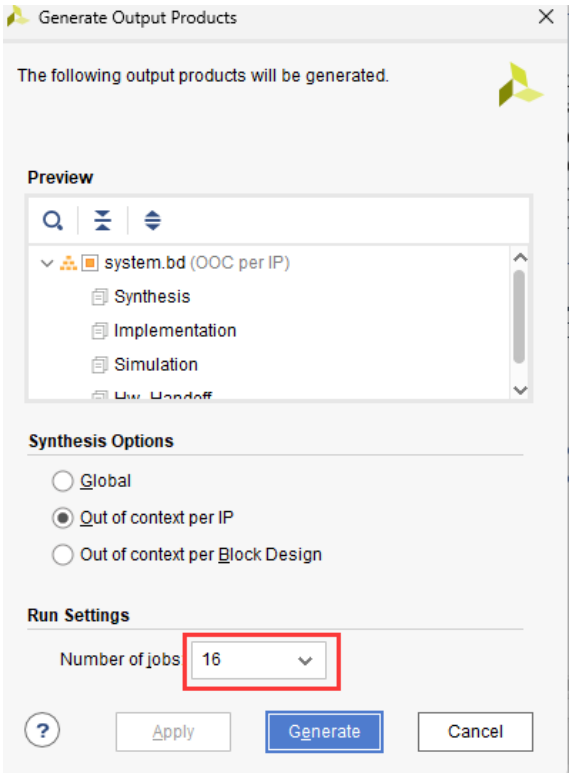


图 1-12 设置生成速度

继续右键单击 system 模块，在展开的功能中选择“Create HDL Wrapper...”创建 HDL 封装。

## 1.2.4 生成比特流

因为是基于 OV5640_LCD 工程创建的，所以不再需要管脚约束；但是在生成比特流之前，要删除 SDK 文件夹，防止原工程残留的文件影响。

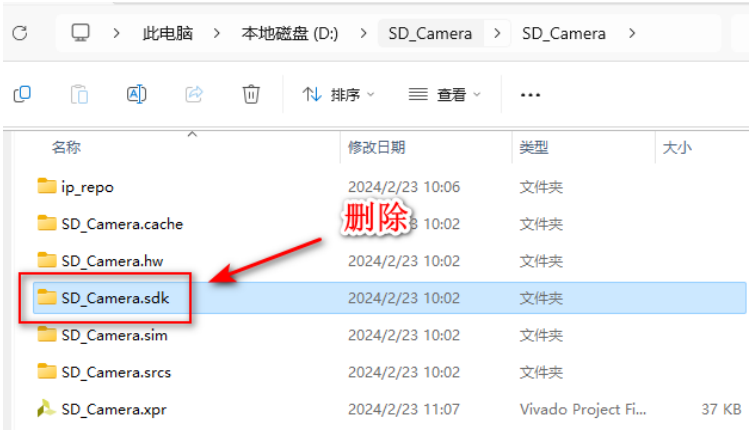


图 1-13 删除 SDK 文件夹

返回软件界面，点击“Generate Bitstream”开始生成比特流；将生成速度设置到最大“16”；出现下图 1-14 所示弹窗，说明比特流生成成功，点击 cancel 即可。

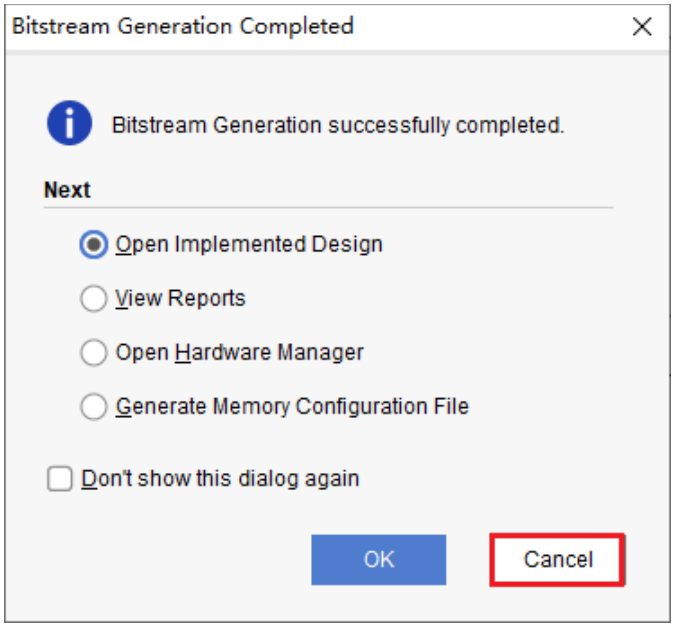


图 1-14 比特流生成成功

## 1.2.5 导出硬件

首先点击 File，然后在展开的功能栏中选择 Export，最后在 Export 的多个选择项中选择“Export Hardware...”将硬件描述文件导出。

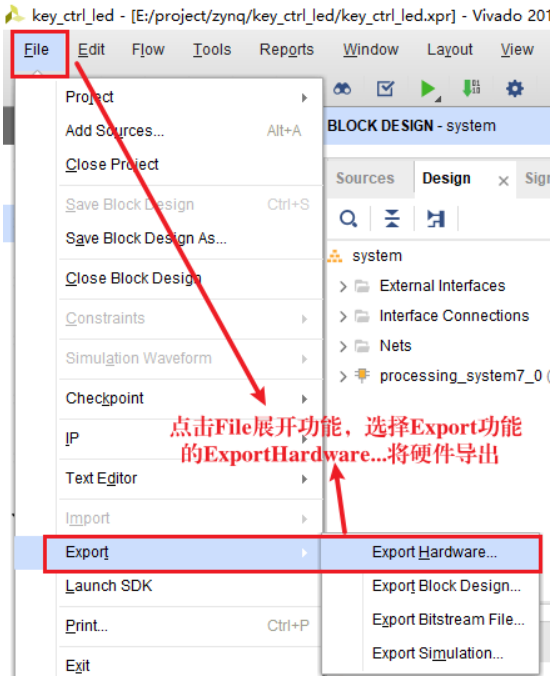


图 1-15 Export Hardware

此时软件会弹出弹窗询问我们是否包含比特流，对于本次设计，我们涉及到了 PL 端的资源使用，因此需要勾选比特流。如图 1-16 所示，勾选比特流后点击 OK 开始导出。

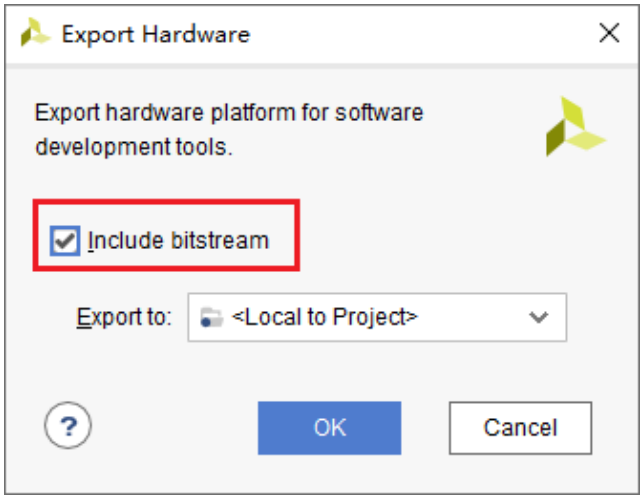


图 1-16 勾选比特流

# 1.3 CPU 软件程序设计

接下来点击 Launch SDK，打开 SDK 软件，开始 CPU 软件程序设计。

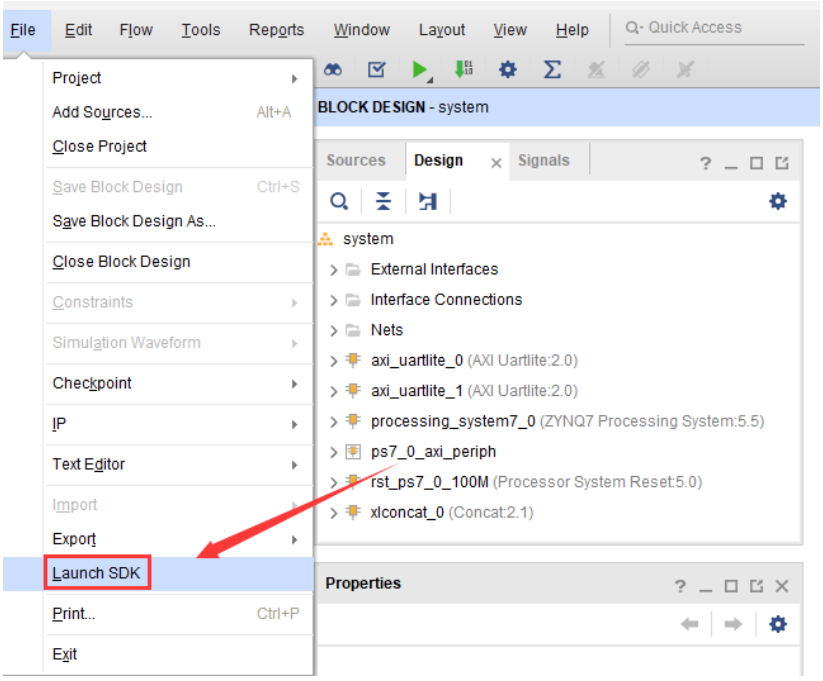


图 1-17 Launch SDK

## 1.3.1 创建 SDK 工程

打开 SDK 后我们需要新建一个 SDK 工程，点击软件左上方的 File，在展开的功能栏中依次选择 New/Application Project，如图 1-18 所示。


图 1-18 新建工程

接下来会进入工程创建界面。此时我们需要为工程命名，设置为“SD_Camera”。这里我们保持默认，点击 Next 进行下一步。

然后为工程选择模板，选择“Empty Application”，如图 1-19 所示，此时右边窗口便会出现当前工程模板的描述，点击 Finish 完成工程创建。

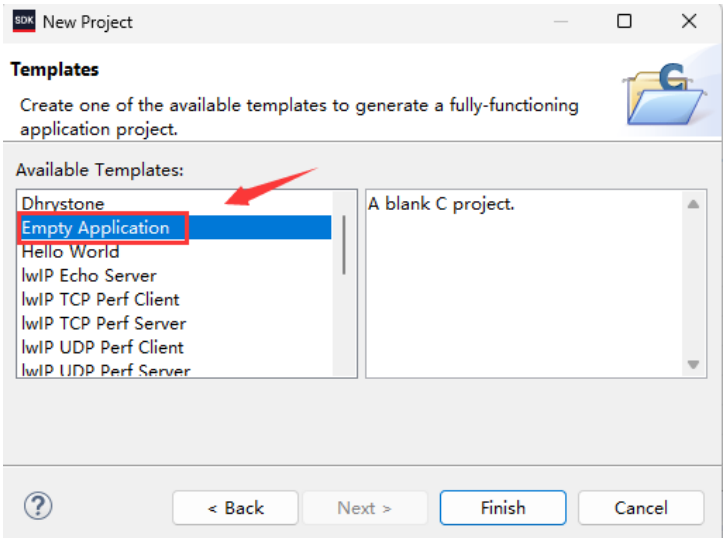


图 1-19 选择模板

## 1.3.2 添加 FATFS 库

对于 FATFS 库的添加，具体方法请参考“SD 卡的文本读写实验”这一节，里面详细列出添加步骤。

## 1.3.3 添加应用库

(1) ACZ702_Lib 文件夹

在创建的工程下面，将例程中的 ACZ702_Lib 文件夹复制到下图所在位置，里面包含“BMP_WR”、“LCD”、“OV5640”、“PS_GPIO”、“PS_IIC”、“SCU”7 个文件夹。

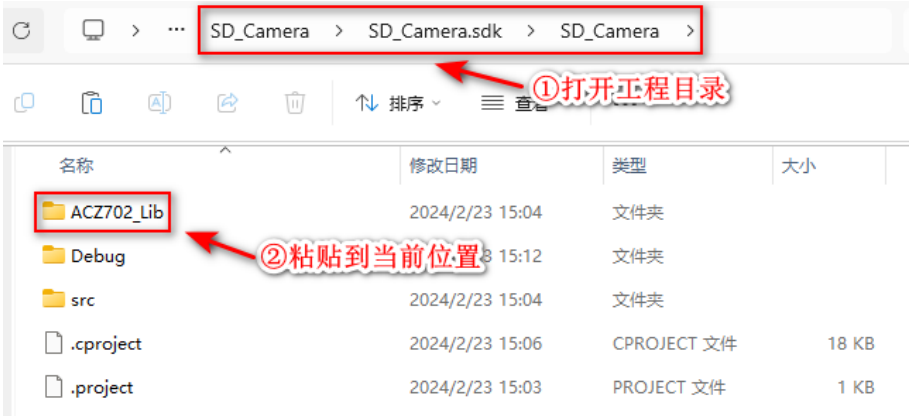


图 1-20 ACZ702_Lib 文件夹

(2) USER 文件夹

打开例程，将 src 文件夹下的 USER 文件夹中的文件拷贝，粘贴到我们创建

的新工程路径“*.../SD_Camera/src*”下方，结果如下图所示。


图 1-21 USER 文件夹

此时打开创建的工程，左键点击“SD_Camera”，然后按下 F5，出现图 1-22，说明添加应用库成功。

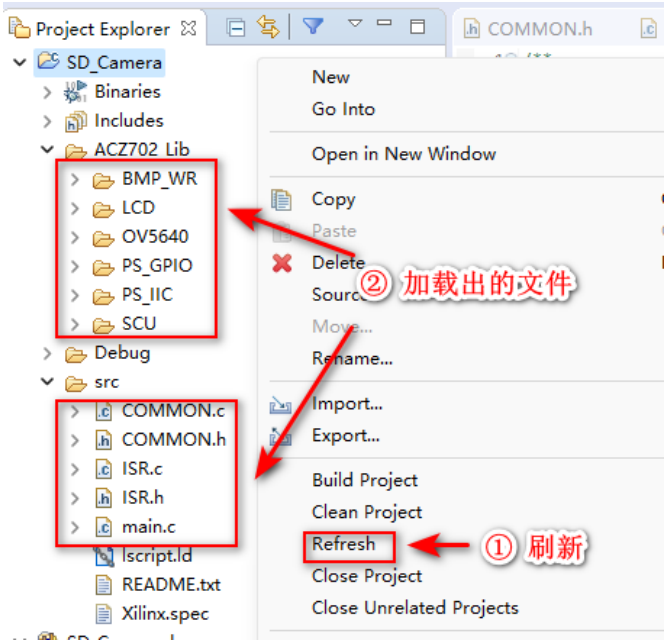


图 1-22 工程创建完成界面

## 1.3.4 添加头文件路径

前面我们导入了库到工程中，每个库都包含着对应的头文件，对于 SDK 而言，此时这些头文件都是无效的，我们需要将这些头文件路径添加进工程中，完成后如下图所示。

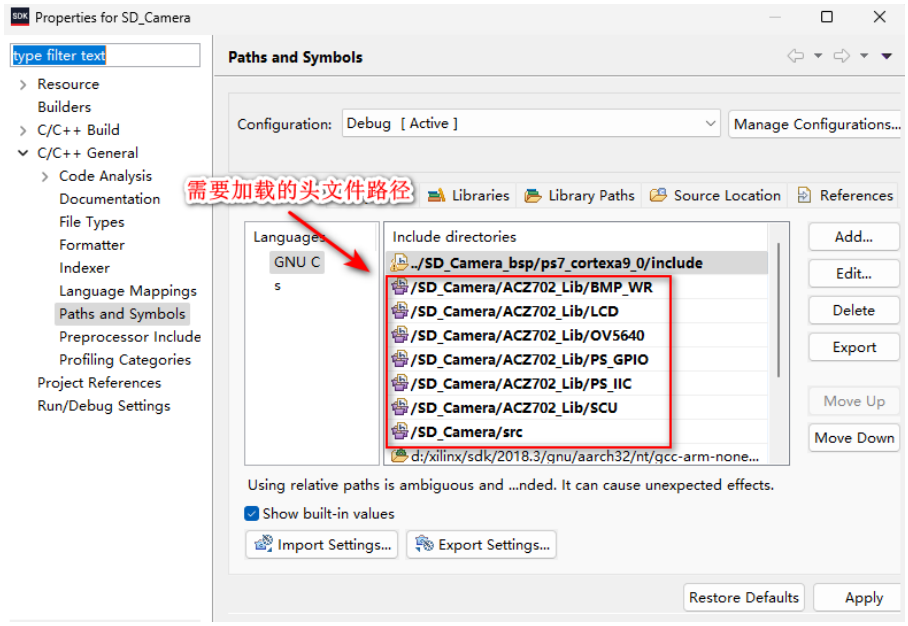


图 1-23 头文件路径

## 1.3.5 添加用户代码

具体代码，可以参考例程，下面挑选重点代码，进行讲解说明：

(1) main.c

在 main 函数中，首先对按键、OV5640 进行初始化；然后配置 VDMA，传输视频数据；继续初始化 LCD 并配置；当程序执行到这里，基本上可以实时获取图像数据；接下来配置 SD 卡（对 SD 卡相关函数不理解的可以回顾文本读取章节）。

接下来进入 while 循环中，通过判断按键的高低电平，获取此时是否按下“快门”，如果按下就进行延时消除抖动，执行 XAxiVdma_StartParking 函数后，缓存区不会再更新数据，LCD 上显示的图片会达到暂停的效果，此刻也会将捕捉到的图像传输到 SD 卡保存，一旦程序执行完保存操作，会运行 XAxiVdma_StopParking 函数继续接收来自 OV5640 的数据。

但是如果去查看图片会发现，它是倒置的；所以使用两个 for 循环将图片旋转 180 度。

```
#include "COMMON.h"
//文件系统
FATFS fs;
FRESULT Res;

//BMP 图片编号与文件名
int Cnt = 0;
```

```
char Photo_Name[20];

//存放按键（MIO47）的电平状态，0 为低电平，1 为高电平
uint8_t State;

int main(void)
{
    PS_GPIO_Init(); //初始化 PS 端 MIO 和 EMIO
    PS_GPIO_SetMode(PS_KEY, INPUT, 0); //设置 PS_KEY(MIO47)方向为输入

    OV5640_Init();  //初始化 ov5640

    //配置 VDMA
    run_vdma_frame_buffer(&Vdma, VDMA_ID, RGB_LCD.width,
RGB_LCD.height, FrameBuffer_Addr,0, 0,BOTH);

    //初始化 Display controller
    DisplayInitialize(&DispCtrl, DISP_VTC_ID, DYNCLK_BASEADDR);

    //设置 VideoMode
    DisplaySetMode(&DispCtrl, &RGB_LCD);
    DisplayStart(&DispCtrl);

    //挂载 SD 卡
    Res = f_mount(&fs, "0:/", 1);
    if(Res != FR_OK) {
        //挂载失败
        printf("Mount failed to return a value of \"%d\" !\n", Res);
        return -1;
    }

    while(1)
    {
        //读取 PS_KEY 的电平值并存储到 State 变量里
        State = PS_GPIO_GetPort(PS_KEY);
        if(0 == State) {
            usleep(10000);//延时一段时间
            if(0 == State)//重新判断按键状态
            {
                //开启停车模式，暂停当前帧
                XAxiVdma_StartParking(&Vdma, 1, XAXIVDMA_READ);

                //将图像旋转 180 度
                for(int j = 0; j < RGB_LCD.height; j++) {
                    for(int i = 0; i < RGB_LCD.width; i++) {
                        memcpy(
```

```
                            (void *) (Photo_Addr + ((RGB_LCD.height -
                                    1 - j)*RGB_LCD.width +
                                    RGB_LCD.width - 1 - i) * 3),
                            (void *) (FrameBuffer_Addr + (j *
                                    RGB_LCD.width + i) * 3),
                            3
                        );
                    }
                }

                //给图片命名
                sprintf(Photo_Name, "Photo(%02d).bmp", Cnt);

                //将图片写入到 SD 卡
                bmp_write(Photo_Name, (char *)&BMODE_800x480, (char
                        *)Photo_Addr);

                //BMP 图片编号累加
                Cnt++;
            }
        }

                //关闭停车模式，继续显示
                XAxiVdma_StopParking(&Vdma, XAXIVDMA_READ);
    }
    return 0;
}
```

(2) bmp_write 函数

继续分析 bmp_write 函数，它的主要作用是将帧缓存写入到 SD 卡中。首先，根据传入的 name 来新建一个文件。然后，将传入的 head_buf 写入文件，这个 head_buf 包含了 BMP 图片文件头的信息，一般长度为 54 字节。如果文件头写入失败，函数直接返回。

接着，从头部信息中读出图像的宽度和高度，分别赋值给 Ximage 和 Yimage。

之后，通过双重循环对每一像素进行处理，从 data_buf 读取每一像素的颜色信息，写入到 Write_line_buf 缓冲区。

之后，将 Write_line_buf 中的内容写入文件。如果写入失败，打印出错误信息并返回。

最后，调整像素的地址，因为这里的循环是从底到顶写入图像的。

所有像素处理完毕后，关闭文件，并打印出成功的信息。

总的来说，这个函数的作用就是根据给定的头部信息和像素信息，新建并写入一个 BMP 图片文件。

```
void bmp_write(char * name, char *head_buf, char *data_buf)
{
    short y,x;
    short Ximage;
    short Yimage;
    u32 iPixelAddr = 0;
    FRESULT res;
    unsigned int br;          // File R/W count

    memset(&Write_line_buf, 0, 1920*3) ;

    res = f_open(&fil, name, FA_CREATE_ALWAYS | FA_WRITE);
    if(res != FR_OK)
    {
        return ;
    }
    res = f_write(&fil, head_buf, 54, &br) ;
    if(res != FR_OK)
    {
        return ;
    }
    Ximage=(unsigned short)((head_buf[19] << 8) | head_buf[18]);
    Yimage=(unsigned short)((head_buf[23] << 8) | head_buf[22]);
    iPixelAddr = (Yimage-1)*Ximage*3 ;
    for(y = 0; y < Yimage ; y++)
    {
        for(x = 0; x < Ximage; x++)
        {
            Write_line_buf[x*3 +0] = data_buf[x*3 + iPixelAddr + 0] ;
            Write_line_buf[x*3 +1] = data_buf[x*3 + iPixelAddr + 1] ;
            Write_line_buf[x*3 +2] = data_buf[x*3 + iPixelAddr + 2] ;
        }
        res = f_write(&fil, Write_line_buf, Ximage*3, &br) ;
        if(res != FR_OK)
        {
            printf("Write BMP Failed!\n");
            return;
        }
        iPixelAddr -= Ximage*3;
    }

    f_close(&fil);
    printf("Write BMP Successfully!\n");
}
```

# 1.4 板级调试与验证

本次实验的板级验证阶段主要围绕以下任务进行：在加载程序后，观察 LCD 是否出现图像；如出现图像，测试按键按下，图像是否暂停；接下来等待图像再次流动后，取下 SD 卡，然后将卡放入读卡器，插入到电脑后，打开文件管理器，查看 SD 卡目录是否存在 bmp 图像文件，如果出现说明拍照功能已经成功实现。

系统所需硬件如下：

1. ACZ702 v2.0 开发板 x1
2. 4.3 寸（800*480）LCD 屏一个
3. OV5640 摄像头一个
4. Micro SD 卡 x1
5. 软排线一根
6. Type-c 线缆 x1

## 1.4.1 硬件连接

使用软排线将 LCD 屏与开发板相连，然后将 OV5640 插入开发板上摄像头接口， 整个设计对供电的要求不高，因此可以直接使用 type-c 供电。

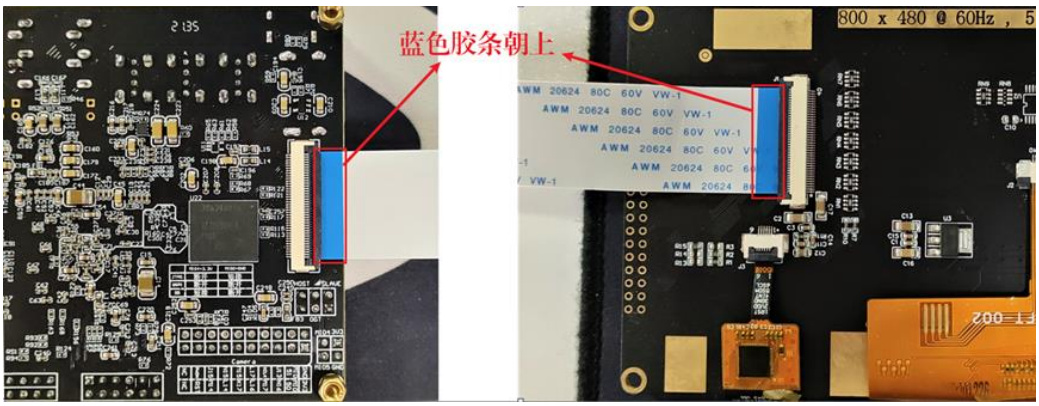


图 1-24 软排线连接

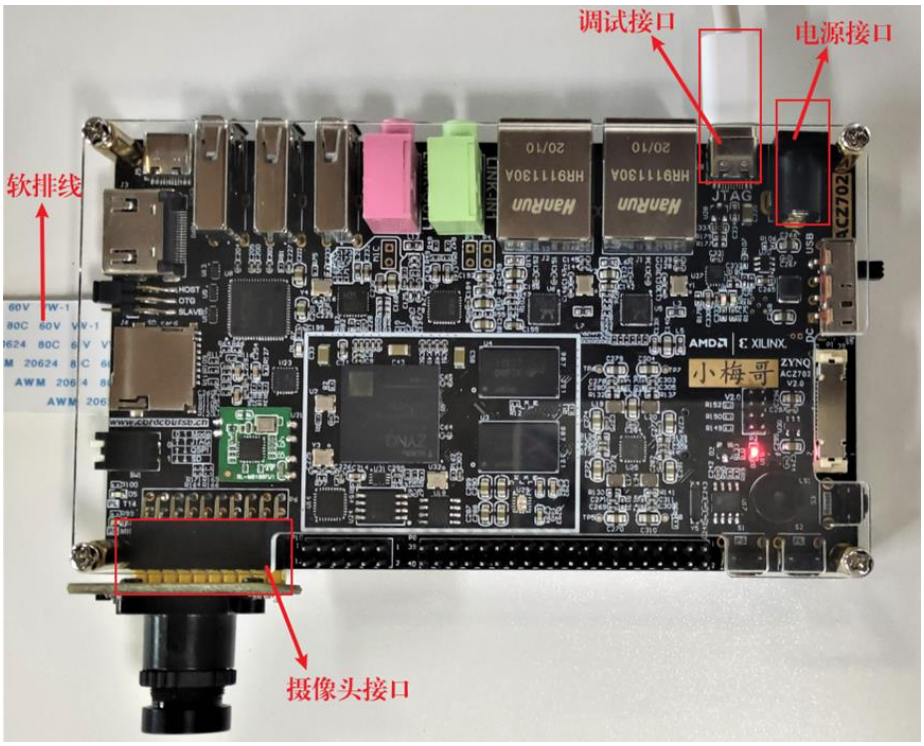


图 1-25 摄像头连接图

最后，将 SD 卡插入卡槽即可。

## 1.4.2 下载验证

点击模板工程资源文件，将生成的烧录文件下载至开发板中如图 1-26 和图 1-27 所示，烧录流程如下：

1. 双击 GDB 或 System Debugger 新建配置任务，推荐使用 GDB。

2. 在右侧检查是否添加比特流文件和 PS 初始化脚本，通常软件会自动添加，如果没有，用户可以关闭并重新打开该界面或自己手动添加。

3. 确认下方三个选项都被勾选，这四个选项分别是系统复位、配置 FPGA、PS 初始化和 PL-PS 电平转换。后两个选项通常是默认勾选的，用户需要手动勾选前两项以确保 PL 部分能够被正确配置。

4. 点击上方的 Application 切换到应用界面。

5. 检查.elf 文件是否添加且烧录任务是否被选中。这里.elf 文件由软件编译产生，参与用户程序的运行。 SDK 默认情况下设置了自动编译，用户保存设计后软件便会编译并生成.elf 文件，所以当搜索不到.elf 文件时检查设计是否保存。

6. 点击“ Run”开始烧录

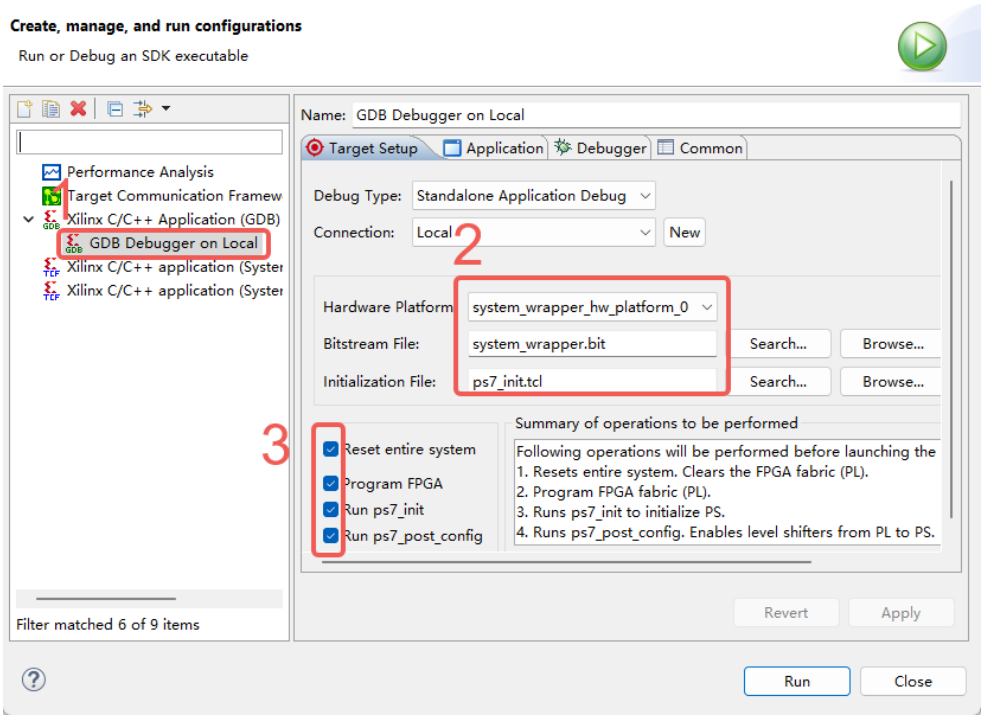


图 1-26 配置烧录任务 1

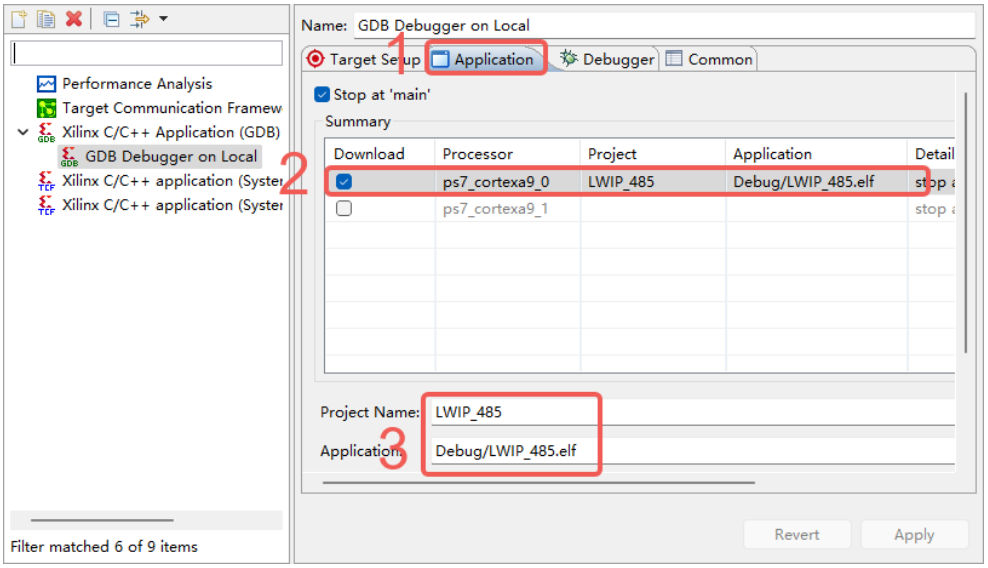


图 1-27 配置烧录任务 2

## 1.4.3 图像采集测试

打开 SDK 工程，下载程序到开发板，等待 LCD 屏幕出现图像。

使用 S1（PS）按键模拟“快门键”，按下快门键后，LCD 上的图像会出现暂停，此时可以观察拍摄的图片是否准确；当屏幕继续“流动”，说明图片已经保存到 SD 卡中；对于不满意的图片，此时也可以重新拍摄。

注意：在图像暂停期间不能再次按下“快门键”，此时缓存中仍然是上一次拍摄的图片；需要等待 LCD 上图像再次变化，才会刷新缓存。

图片拍摄如下图所示。

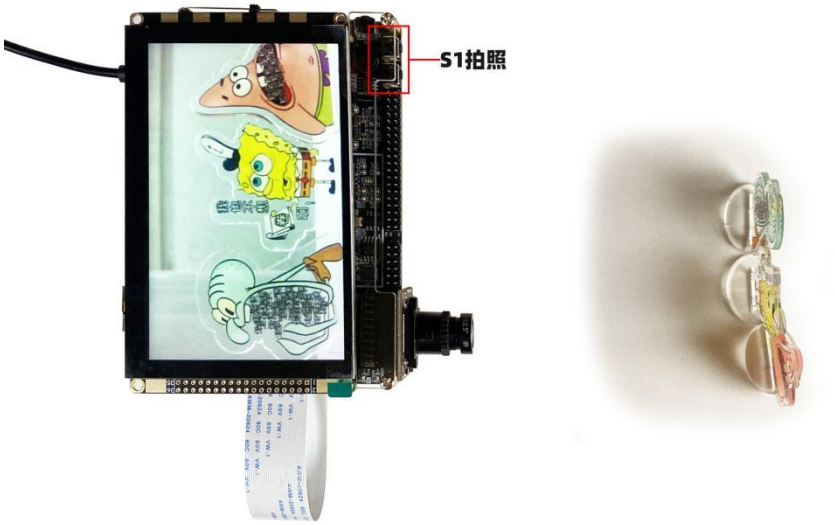


图 1-28 拍摄图片

取下 SD 卡后，通过读卡器插入到电脑端，然后打开 Photo(00).bmp 图片，效果如下图所示：


图 1-29 电脑屏幕显示效果

## 1.5 总结

在这次实验中，我们使用了 LCD、OV5640 等模块来实现照相机功能，并同时利用 SD 卡保存图片，便于后期传输到电脑浏览。本次实验为读者提供了一个初步的实践场景，展示了如何从 SD 卡中写入数据，以及如何将帧缓存转为 BMP 格式图片。