1 基于 ZYNQ7020 的 LCD 电子相册

工程源码	|-ACZ702 v2.0 开发板 |-- SD_LCD
相关视频课程	无

章节导读

在本节中将实现从 SD 卡中读取 BMP 图片并显示到 LCD 上，同时可通过按键浏览控制。在 SD 卡文本读写实验中，我们简单介绍了 FATFS 的技术原理，并步骤化地讲解文本的创建、写入、读取；而在这一节中，我们将更深层次地阐述 BMP 图片的读取与显示的方法。本实验的目标是使读者能对 SD 卡读取图片的应用有一个基本的了解和初步的实践经验。

1.1 图片格式介绍

(1) 常见图片格式

图片格式种类很多，比较常见的种类有以下几种：

1. JPEG：这是一种非常常用的有损压缩图片格式，适用于颜色丰富、细节丰富的照片和图片。JPEG 常用于网络上，因为其小巧的文件大小可以节省带宽和加载时间。

2. PNG：PNG 格式是一种无损压缩的格式，支持透明度，常用于网页设计元素，例如按钮、图标和背景。它提供了比 JPEG 更丰富的颜色和更好的压缩。

3. GIF：GIF 是一种无损压缩的 8 位（256 色）格式，常用于创建简单的动画和小图标。GIF 对于颜色较少的图像效果最好。

4. BMP：位图图片格式（BMP）是一种未压缩的格式，所有的图像细节都被保留。由于其文件大小非常大，BMP 格式通常不用于网络或移动设备。

5. TIFF：TIFF 格式通常用于专业领域，如摄影和出版，因为这种格式保留了所有的图像细节并且是无损的。TIFF 文件的大小通常比 JPEG 大得多。

6. RAW：RAW 格式是许多高端数码相机的默认格式。与其他格式不同，RAW 文件包含了传感器捕获的所有数据，因此提供了最高质量的图像和最大程度的后期处理灵活性。

7. SVG：可缩放矢量图形（SVG）是一种 XML-based 的矢量图像格式，设计用于描述二维矢量和基于矢量的混合图像。SVG 图像可以在任何大小下保持清晰，并且通常用于网页和打印设计中。

(2) BMP 格式

在嵌入式开发中，相对而言 BMP 格式使用更加广泛。一般 BMP 图像文件是由四部分组成：

1. 位图文件头：14 字节，描述了整个 BMP 文件的基本信息，包括文件大小、保留字等。

2. 位图信息头：40 字节，记录了有关图像的详细数据和元数据，如图像宽度、高度、像素格式等。

3. 调色板：如果图像是索引色（也就是每个像素不是直接定义颜色，而是定义了一个调色板的索引），调色板就包含在文件中，在位图信息之后。调色板主要是颜色索引与实际颜色的映射关系。

4. 位图数据：包含实际的像素信息。如果图像是索引色，那么它将是调色板的索引。对于非索引颜色，像素数据将直接提供颜色信息。

24 位图是最常见的格式，每一个像素的颜色信息在此种格式中由 24 位来表示，也就是说，它会用 8 位来分别表示色彩三元素 RGB。除此之外，还有其他表示图像的方式，如 8 位，16 位等等。

接下来，使用 Notepad++软件（已安装 HEX-Editor 插件）分析图片“0.bmp”的文件内容。在图片上右键选择 Notepad++打开，如图 1-1 所示。

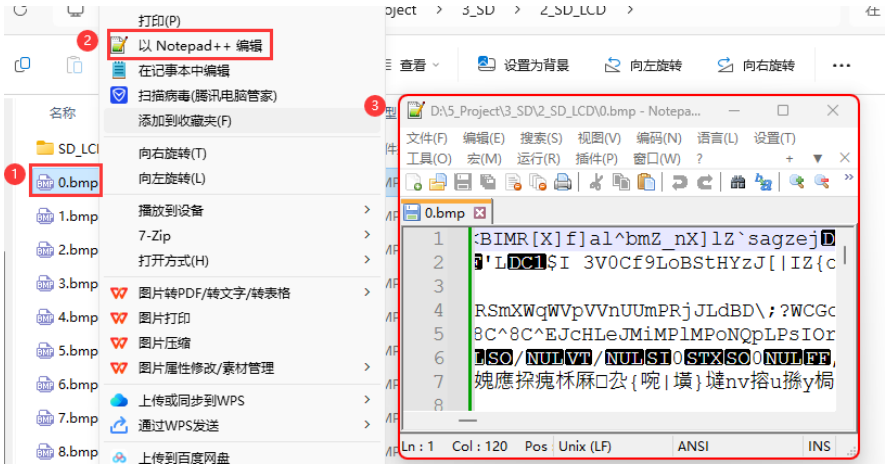

图 1-1 图片分析

如下图所示，继续点击菜单栏上的“插件”，选择“HEX-Editor”，点击“View in HEX”

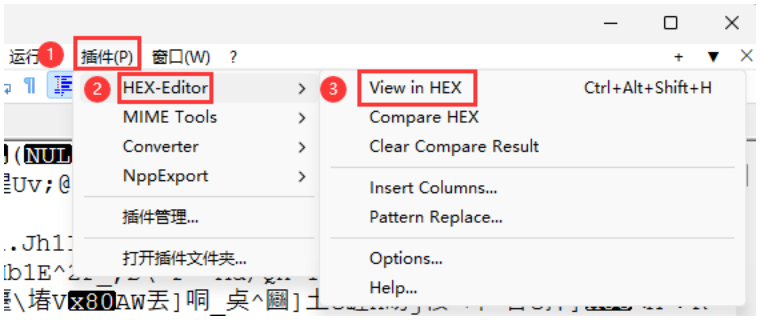

图 1-2 View in HEX

位图文件头，如图 1-3 蓝色区域所示，00~0dh 占 14 字节。其中参数值和作用可以查看表 1。

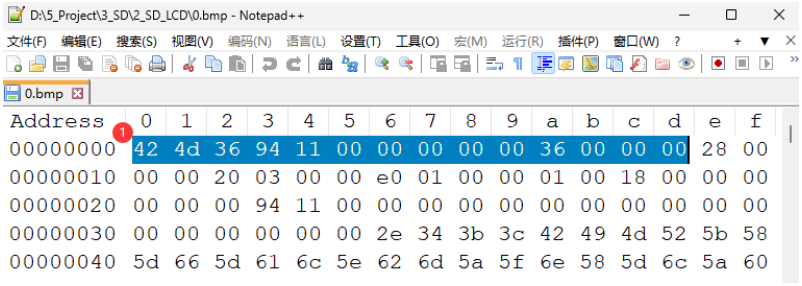

图 1-3 位图文件头

表 1 位图文件头地址

变量名	地址	参数值	作用
bfType	00~01h	42 4d	文件类型标识符，"BM"，表示这是一个.bmp 的位图文件。
bfSize	02~05h	36 94 11 00	文件大小，0x119436 转为十进制 1152054
bfReserved1	06~07h	00 00	保留字段，一般为 0。
bfReserved2	08~09h	00 00	保留字段，一般为 0。
bfOffBits	0a~0dh	36 00 00 00	这是实际位图数据的偏移量，此处是 54 字节，表示从文件开始到像素数据开始的字节偏移量。

位图信息头，如图 1-4 红色区域，共 40 字节。其中参数值和作用可以查看表 2。

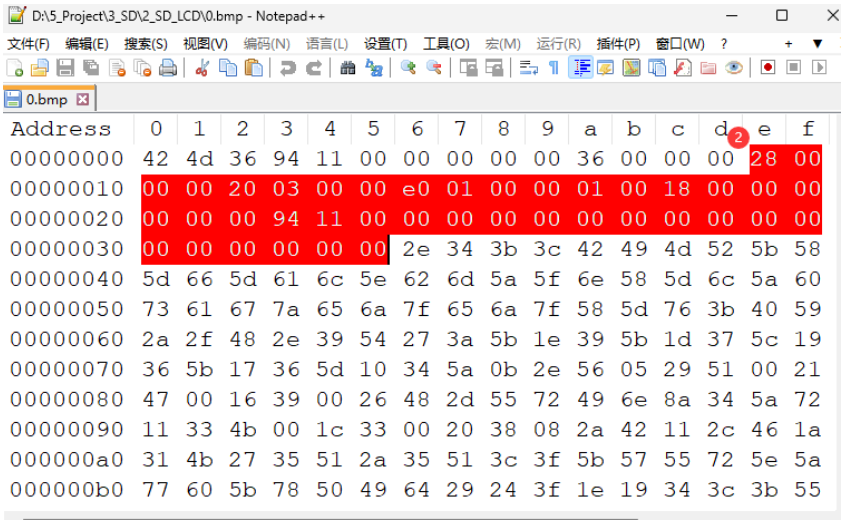

图 1-4 位图信息头

表 2 位图信息头

变量名	地址	参数值	作用
biSize	0e~11h	28 00 00 00	表示位图信息头的大小，此处是 40 字节
biWidth	12~15h	20 03 00 00	表示图像的宽度，此处是 800 像素
biHeight	16~19h	e0 01 00 00	表示图像的高度，此处是 480 像素
biPlanes	1A~1Bh	01 00	表示位平面数，一般为 1
biBitCount	1C~1Dh	18 00	表示每个像素的位数，此处是 24 位，也就是真彩色
biCompression	1E~21h	00 00 00 00	表示压缩类型，此处为 0，也就是不压缩。
biSizeImage	22~25h	00 94 11 00	表示图像大小，此处是 1152000 字节

1.2 硬件逻辑系统设计

1.2.1 添加 IP 核

本次设计我们使用到 Zynq 处理系统 IP 核、AXI-VDMA IP 核、AXI-Stream to Video Out IP 核、Video Timing Controller（VTC）IP 核、Dynamic Clock Generator IP 核、rgb to lcd IP 核。

rgb to lcd 核是自定义 IP 核，其导入与添加方法这里不再赘述，用户可以参考自定义 IP 核章节。

1.2.2 IP 核配置

(1) Zynq IP 核

首先，如下图所示，修改 Bank 1 为 LVCMOS 1.8V。

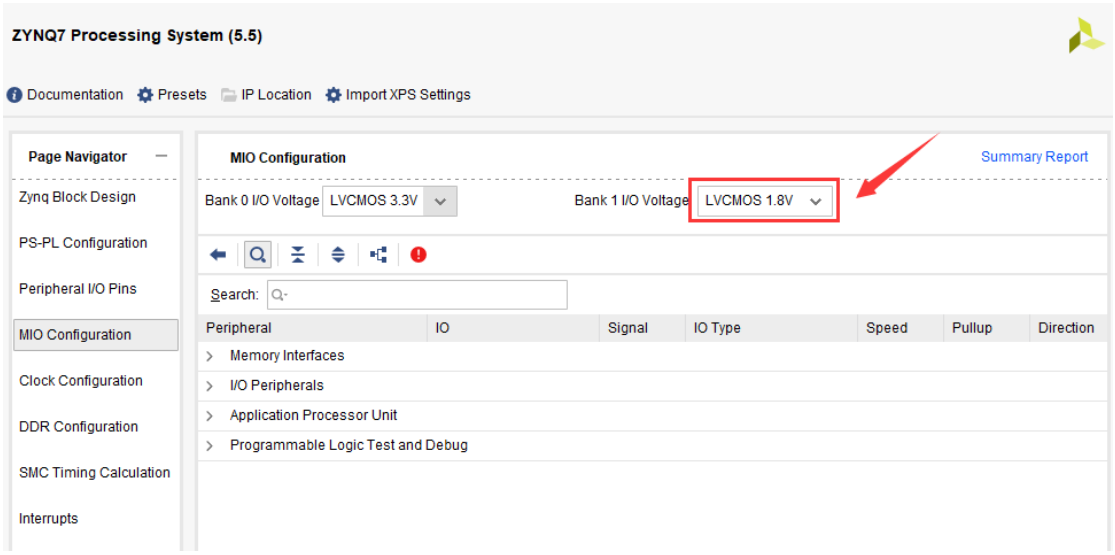

图 1-5 Bank 1

打开 DDR 配置界面，设置 DDR 型号为“MT41K128M16 JT-125”，如图 1-6 所示。

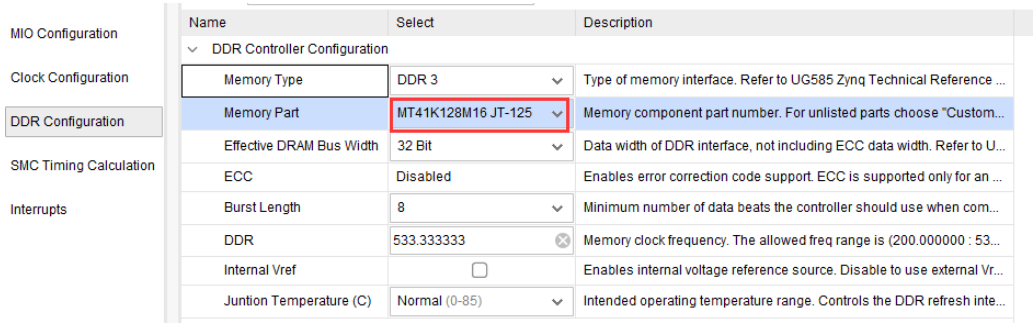

图 1-6 DDR 型号

AXI GPIO 核的工作时钟应该低于 120MHz，因此这里我们打开 Clock Configuration 界面，按照图 1-7 配置 PL 的时钟为 100MHz。

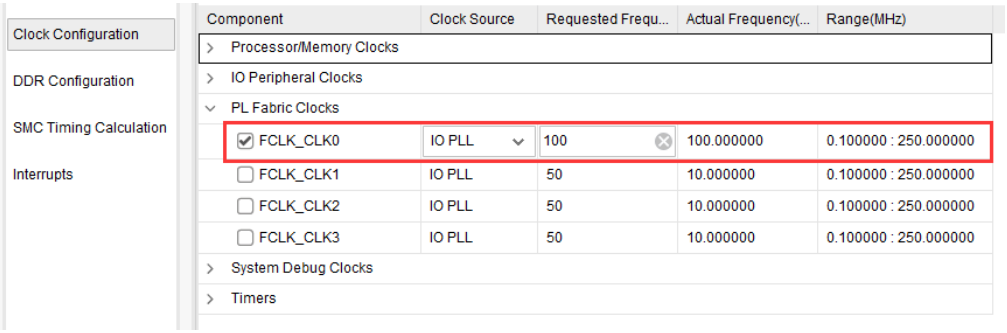

图 1-7 PL 时钟配置

PL 侧的中断需要使能对应中断端口才能连接到 PS，本次设计所产生的中断类型属于共享外设中断的 PL 中断。因此，如图 1-8 所示，使能 IRQ_F2P[15:0]

端口。

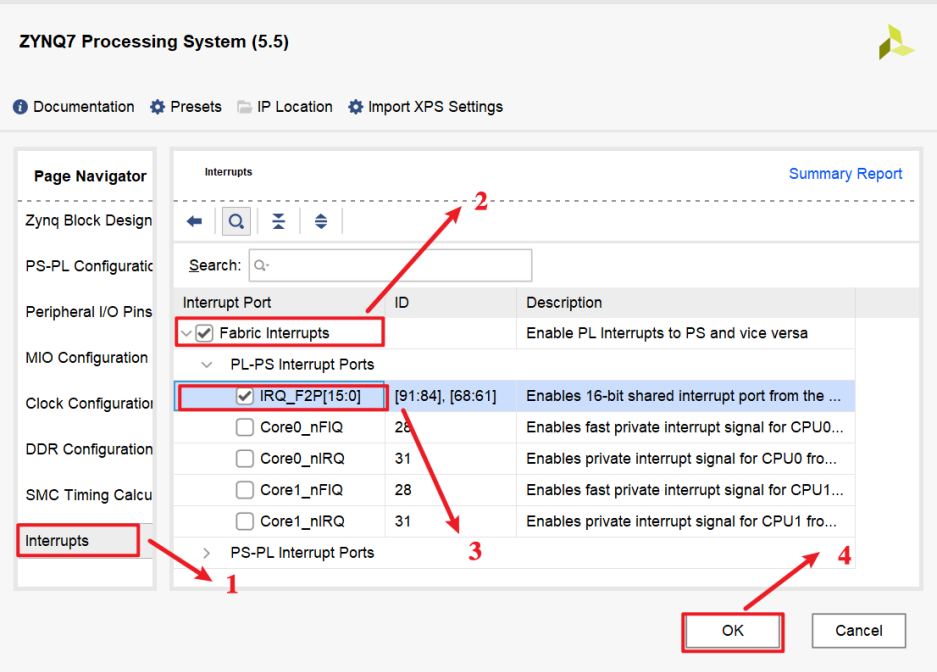

图 1-8 使能中断端口

控制图片翻页时需要使用到按键，所以勾选上 GPIO；

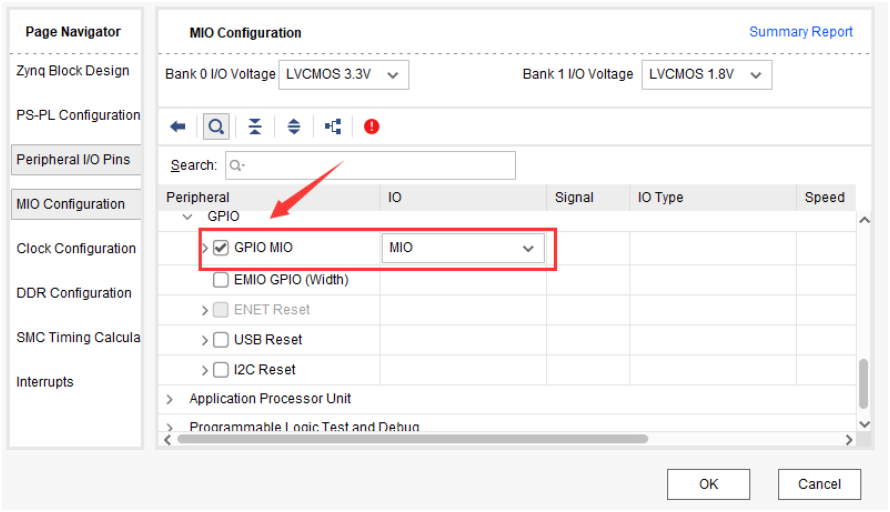

图 1-9 GPIO 配置

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)；ACZ702 开发板上 PS 侧串口引脚对应 MIO48...49，外设使能如图 1-10 所示：

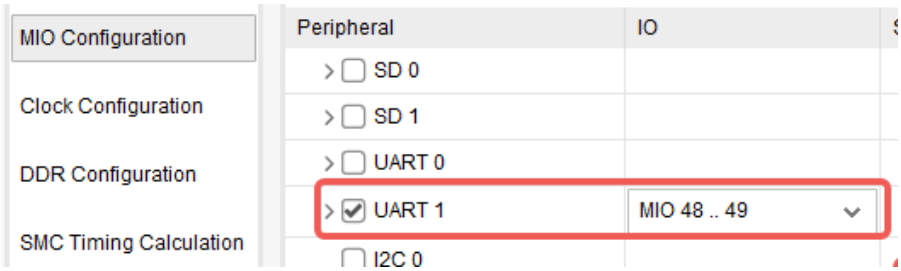

图 1-10 PS 端串口配置

实验图片数据会从 SD 卡传输到 DDR3，在这中间会在 DDR（PS）与 VDMA (PL)间进行数据传输。

在 Zynq 上有着专门用于 PS 与 PL 间高速数据传输的 HP 接口，本次设计我们将使用到该接口。HP 接口的使能如图 1-11 所示，进入 PS-PL 配置界面，展开 HP Slave AXI 接口，勾选 HP0 即可。

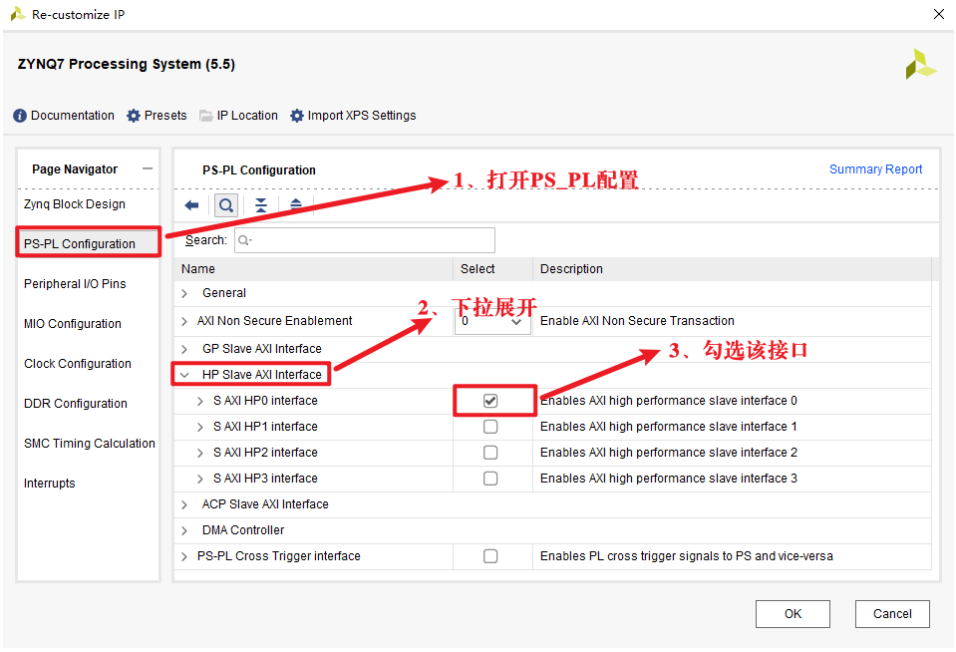

图 1-11 HP Slave AXI 接口

本章重点是从 SD 卡中读取数据，所以“SD 0”肯定要勾选的。

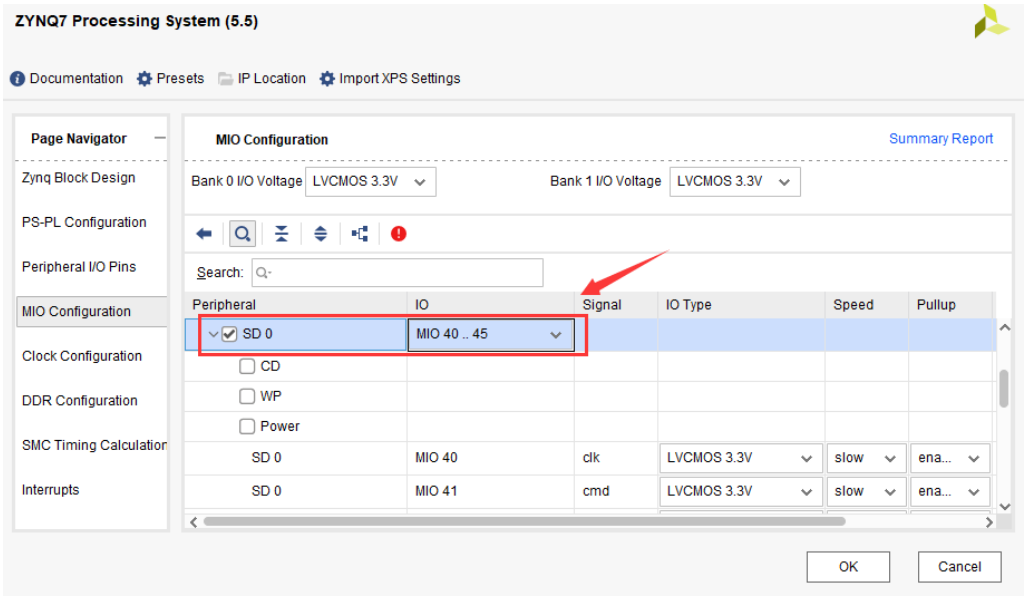

图 1-12 SD

(2) AXI GPIO IP 核

在 Zynq IP 核配置完成后，双击 AXI GPIO IP 核，将位宽设置为 1，勾上中断使能。

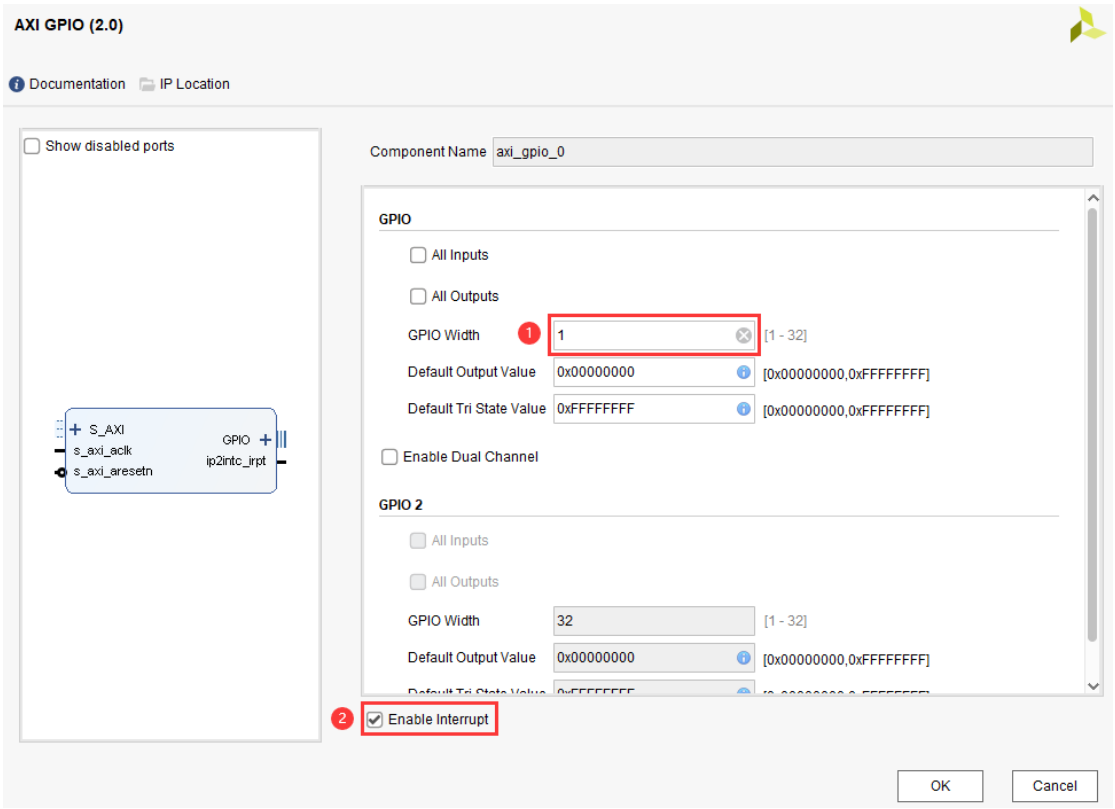

图 1-13 AXI Uartlite IP 核配置

(3) AXI Video Direct Memory Access 的配置

将位宽设置为 32，帧缓存设置为 1，取消写通道使能；配置 Memory Map 数据位宽为 64、突发读长度为 64、Stream 数据位宽为 24，行缓冲区深度为 1024，完成后如图 1-14 所示。

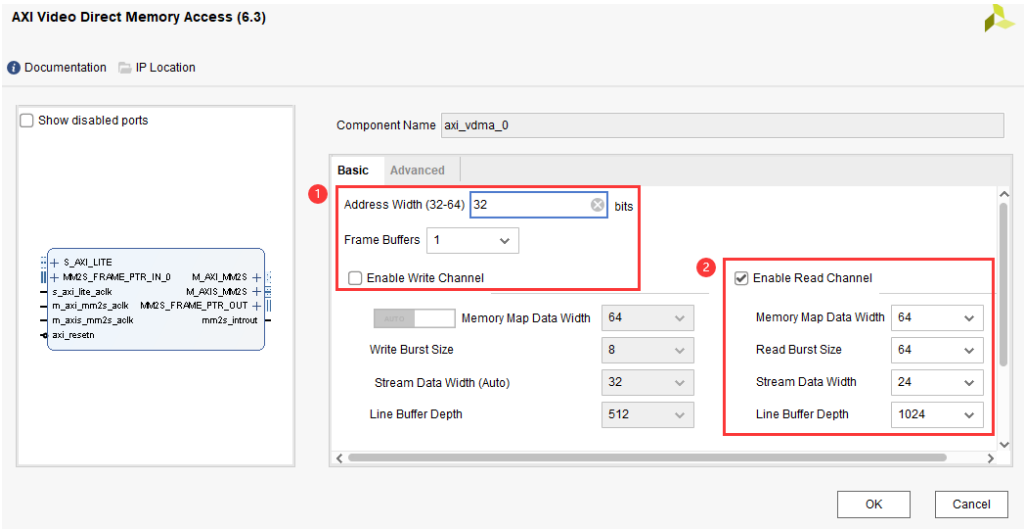

图 1-14 VDMA 配置

(4) AXI-Stream to Video Out IP 核

该 IP 核配置简单，只需将时钟模式一栏修改为 Independent。

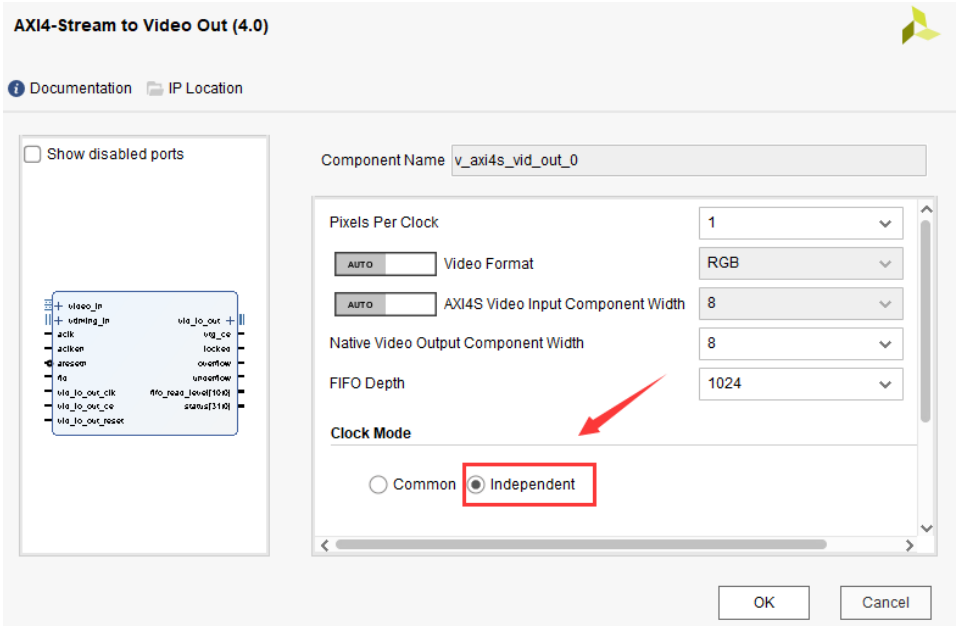

图 1-15 时钟模式修改

(5) Video Timing Controller

使用 VTC IP 核可以非常灵活地控制视频同步信号的产生和处理。在本次实验中主要用于生成时序信号，所以只需使能生成选项，取消检测选项的使能。

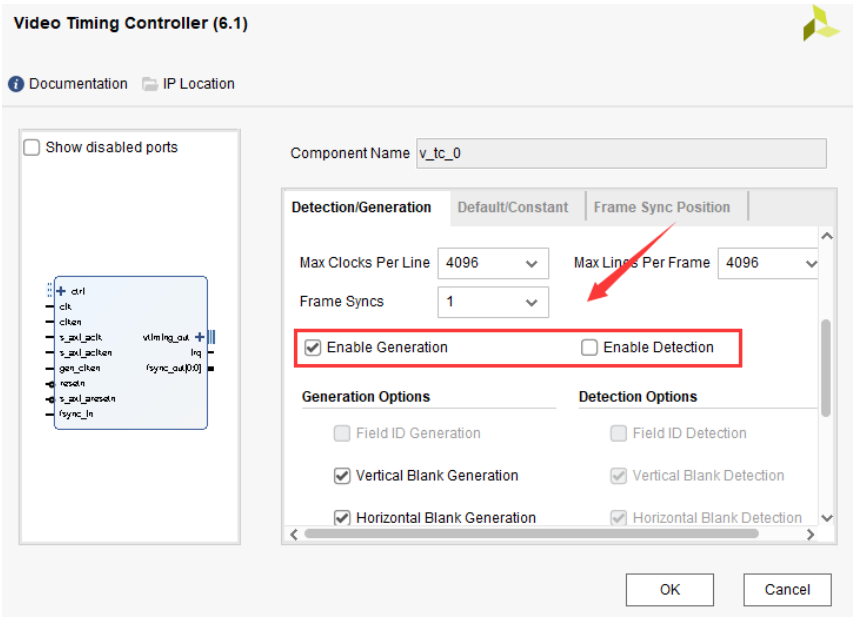

图 1-16 VTC 配置

其余 IP 核无需配置，默认使用即可。

1.2.3 导出引脚

接下来我们需要将 IP 核的引脚导出，点击上方的蓝色小字“Run Block Automation”，让系统自动帮我们导出引脚。此时软件会弹出弹窗，勾选“ALL Automation”，其余部分保持默认，点击 OK，软件便会帮我们导出。

1.2.4 端口连接

(1) 手动连接

首先，连接时钟电路，如下图所示，将 clk、PXL_CLK_O、vid_io_out_clk、pixel_clk 相连接。

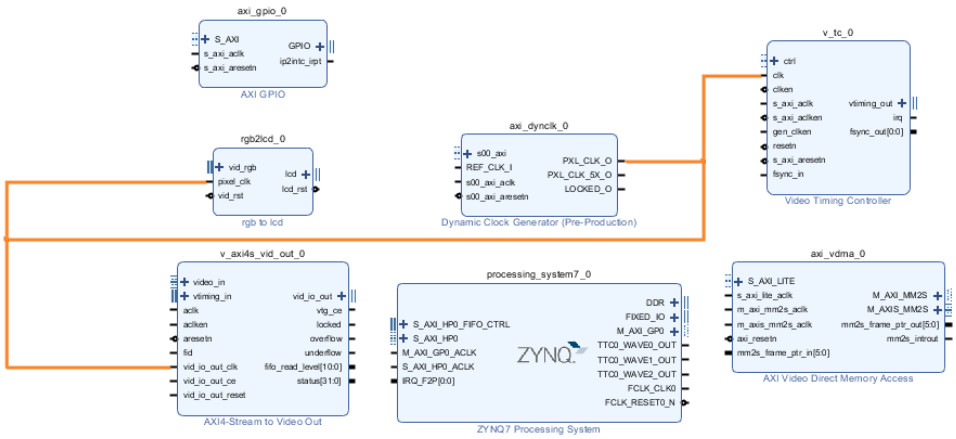

图 1-17 FCLK_CLK0

接下来，将 AXI GPIO 核的 ip2intc_irpt 连接到 ZYNQ 核的 IRQ_F2P

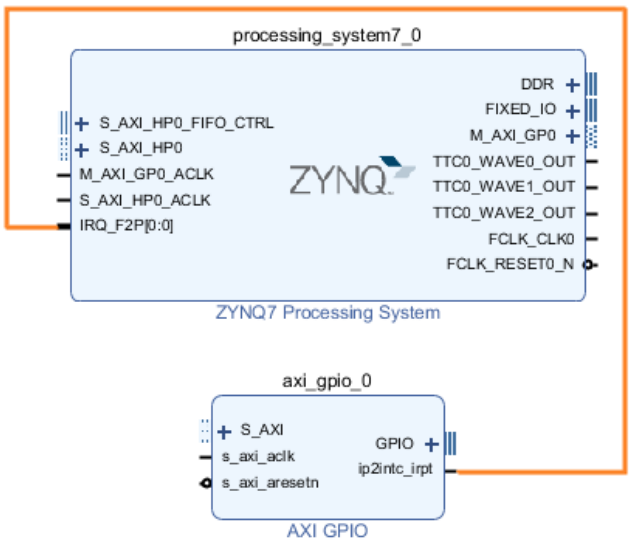

图 1-18 ip2intc_irpt 连接

(2) 点击“Run Block Automation”，勾选“ processing_system7_0”，继续点击“OK”。

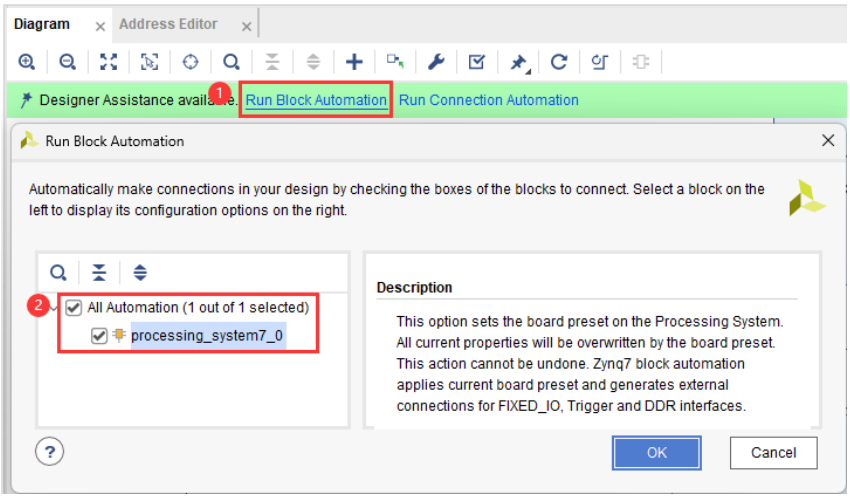

图 1-19 Run Block Automation

(3) 点击“Run Connection Automation”，并勾选弹窗中全部对象，点击 OK，等待自动连接。

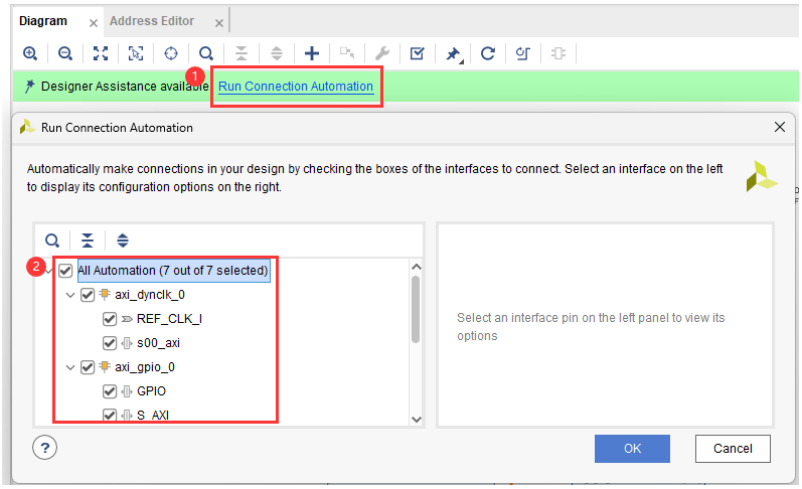

图 1-20 自动连接

点击 Regenerate Layout，重新生成布局，如图 1-21 所示。

图 1-21 重新生成布局

(4) 此时可以看到仍然有很多端口没有连接，接下来按照表 3、表 4、表 5 所列出来的端口进行连接。

表 3 Video Out 端口连接

Video Out 端口	其他端口
vid_io_out	vid_rgb (rgb to lcd)
vtg_ce	gen_clken (VTC)
aclk	FCLK_CLK0 (ZYNQ)
vtiming_in	vtiming_out (VTC)
video_in	M_AXIS_MM2S(VDMA)

表 4 VDMA 端口连接

VDMA 端口	其他端口
m_axis_mm2s_aclk	FCLK_CLK0 (ZYNQ)

表 5 rgb 2 lcd 端口连接

rgb 2 lcd 端口	其他端口
vid_rst	LOCKED_O (dynclk)

连接完成后，最后效果如图 1-22 所示。

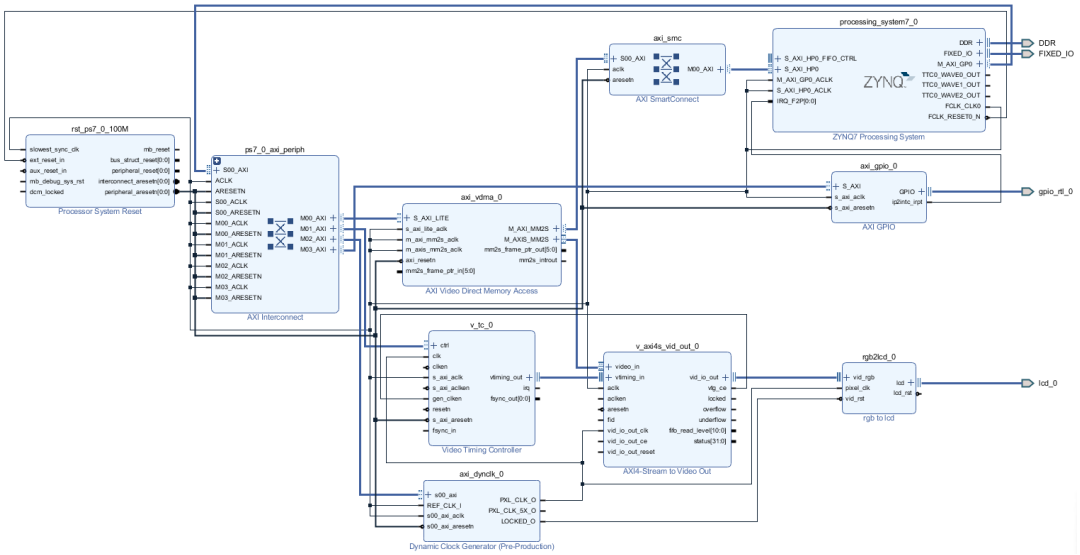

图 1-22 端口连接图

(5) 导出 lcd 引脚

选中 rgb 2 lcd 的 lcd 引脚，右键选择 Make External 导出

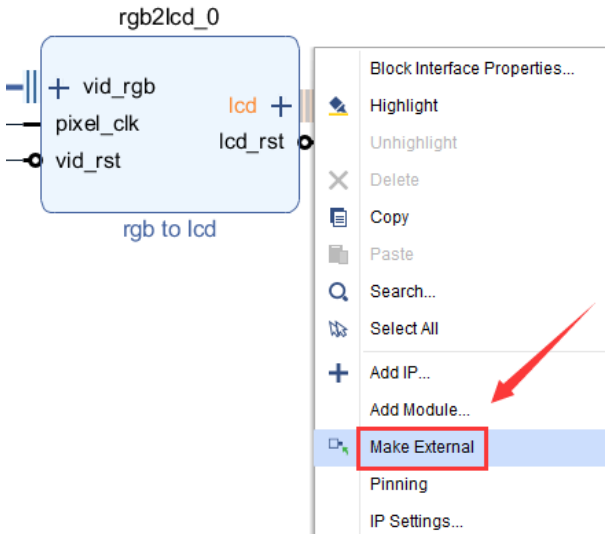

图 1-23 导出 lcd 引脚

最后，验证设计，出现图 1-24 所示的“Validation successfu...”，说明无错误，点击 OK 并按下 Ctrl+S 保存。

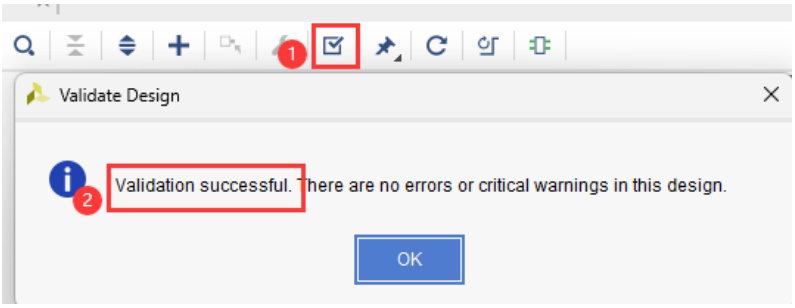

图 1-24 验证设计

1.2.5 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。

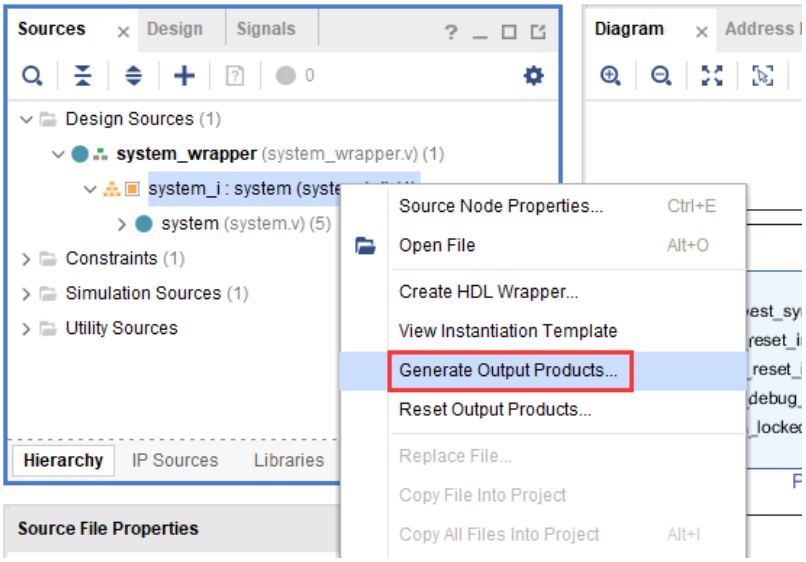

图 1-25 Generate Output Products

如图 1-26 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。

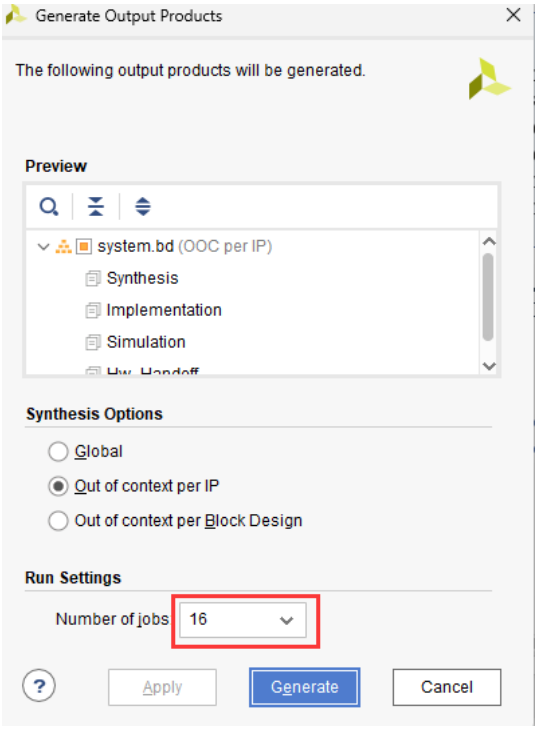

图 1-26 设置生成速度

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。

1.2.6 管脚约束

点击左侧导航栏的“Open Elaborated Design”进行约束和分配。

表 6 LCD 管脚约束

Pin Name	Signial Name	Pin NO.		Pin Name	Signial Name	Pin NO.
Display_R4	lcd_0_data[15]	W20		Display_B4	lcd_0_data [4]	Y19
Display_R3	lcd_0_data [14]	W19		Display_B3	lcd_0_data [3]	W18
Display_R2	lcd_0_data [13]	V17		Display_B2	lcd_0_data [2]	Y18
Display_R1	lcd_0_data [12]	V16		Display_B1	lcd_0_data [1]	W16
Display_R0	lcd_0_data [11]	T15		Display_B0	lcd_0_data [0]	Y17
Display_G5	lcd_0_data [10]	V20		Display _PCLK	lcd_0_pclk	U15
Display_G4	lcd_0_data [9]	U17		Display _HSYNC	lcd_0_hs	U14
Display_G3	lcd_0_data [8]	V18		Display _VSYNC	lcd_0_vs	W14
Display_G2	lcd_0_data [7]	T16		Display _DE	lcd_0_de	W15
Display_G1	lcd_0_data [6]	R16		Display _BL	lcd_0_bl	R17
Display_G0	lcd_0_data [5]	U19				

(1) LCD

对照表配置完引脚后，首先将管脚电平设置为 “LVCMOS33*”，再根据表设置管脚号，完成后，如下图所示。

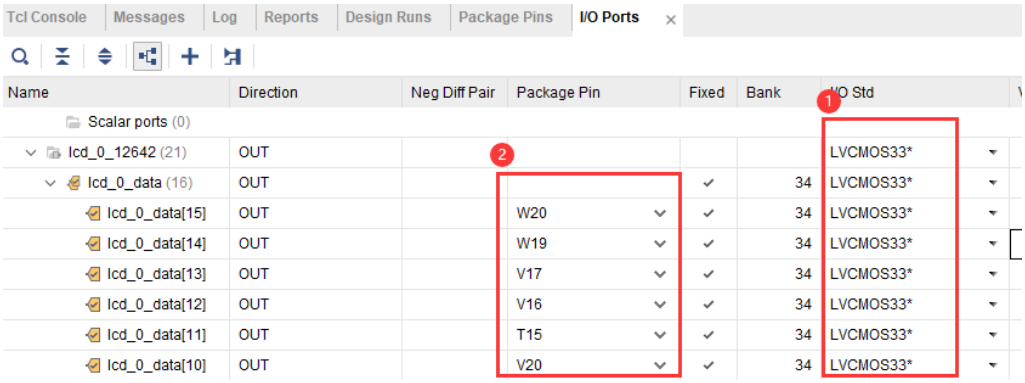

图 1-27 LCD 管脚配置

(2) GPIO

GPIO_0_0_tri_io 引脚编号设置为 F20，管脚电平设置为 LVCMOS33。

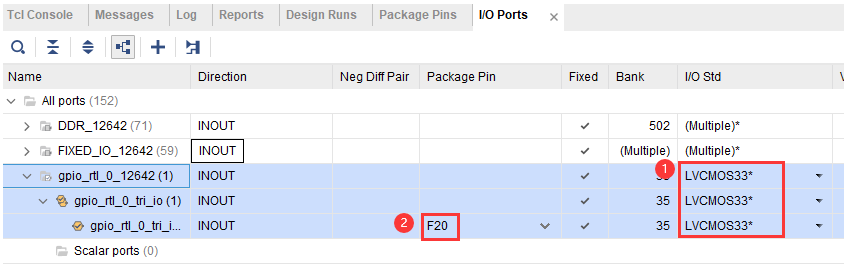

图 1-28 GPIO 管脚电平设置

完成分配后使用快捷键 Ctrl+S 对约束文件进行保存，可以自定义命名，点击 OK 即可。

1.2.7 生成比特流

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 1-29 所示弹窗，说明比特流生成成功，点击 cancel 即可。

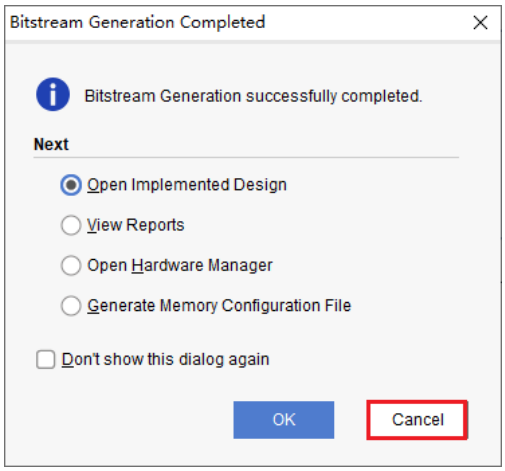

图 1-29 比特流生成成功

1.2.8 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。

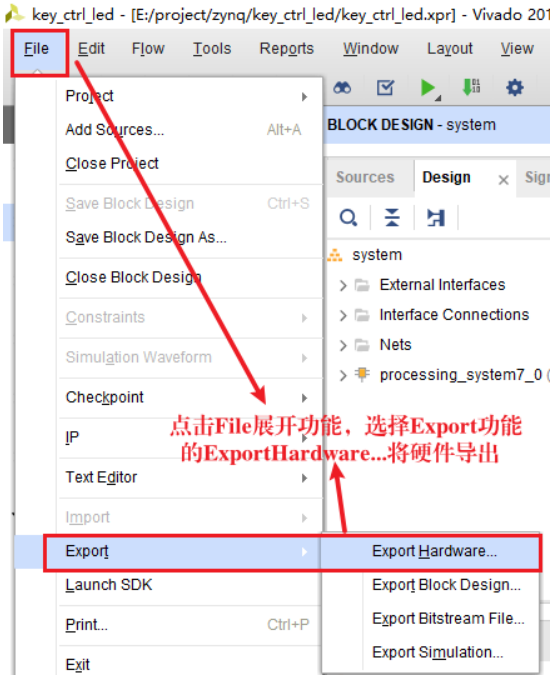

图 1-30 Export Hardware

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，我们涉及了 PL 端的资源使用，因此需要勾选比特流。如图 1-31 所示，勾选比特流后点击 OK 开始导出。

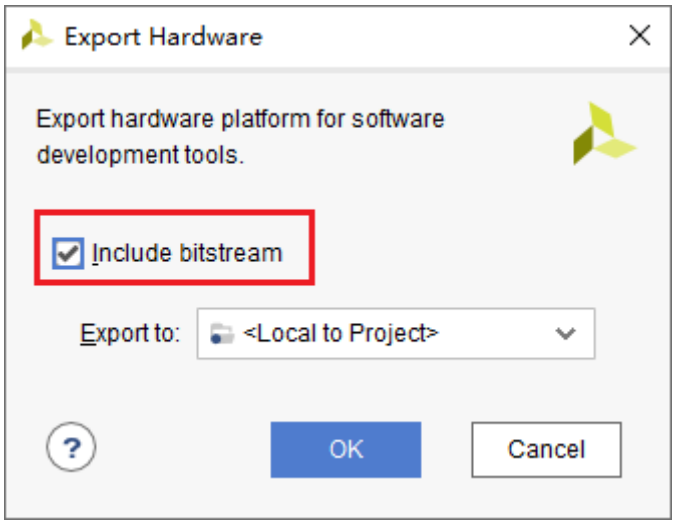

图 1-31 勾选比特流

1.3 CPU 软件程序设计

接下来打开 SDK，开始 CPU 软件程序设计。

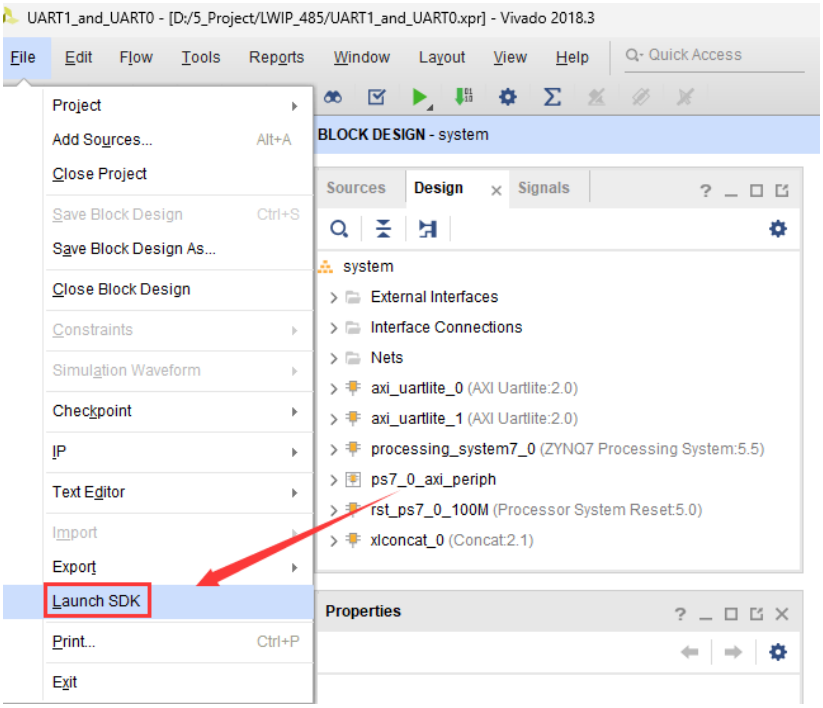

图 1-32 Launch SDK

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。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_LCD”。这里我们保持默认，点击 Next 进行下一步。

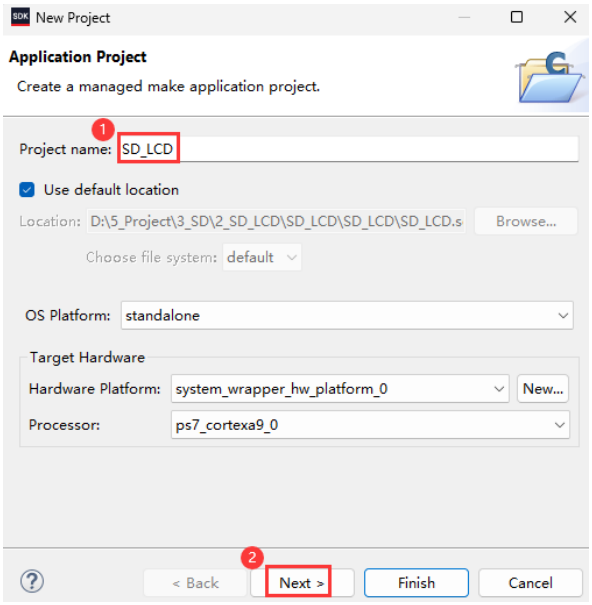

图 1-33 新建项目

然后为工程选择模板，选择“Empty Application”，如图 1-34 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。

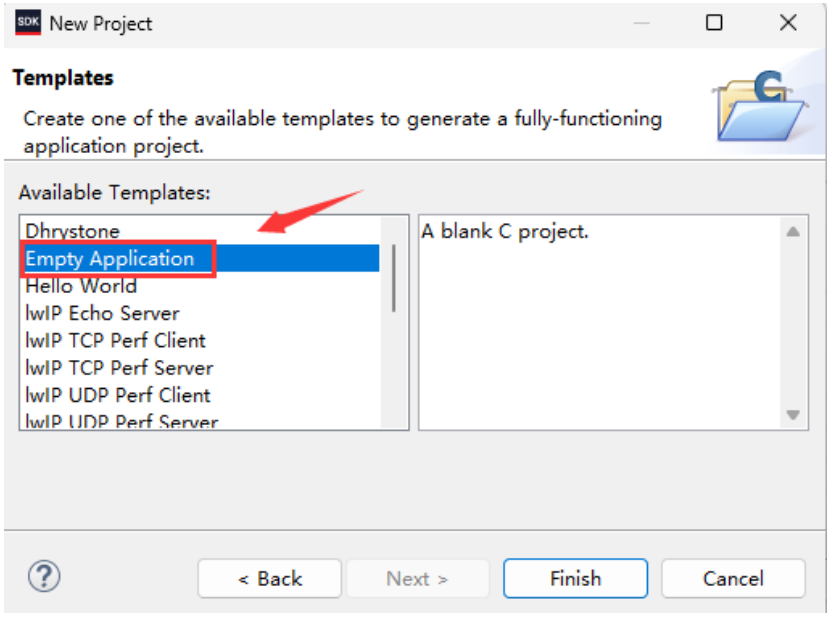

图 1-34 选择模板

1.3.2 添加应用库

(1) ACZ702_Lib 文件夹

打开我们提供的应用库资料，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\ ACZ702_Lib*。

在创建的工程下面，新建一个 ACZ702_Lib 文件夹将“AXI_GPIO”、“PS_GPIO”、“LCD”、“SCU”、“SD_Card”、“TTC” 这 4 个文件夹添加到其中。

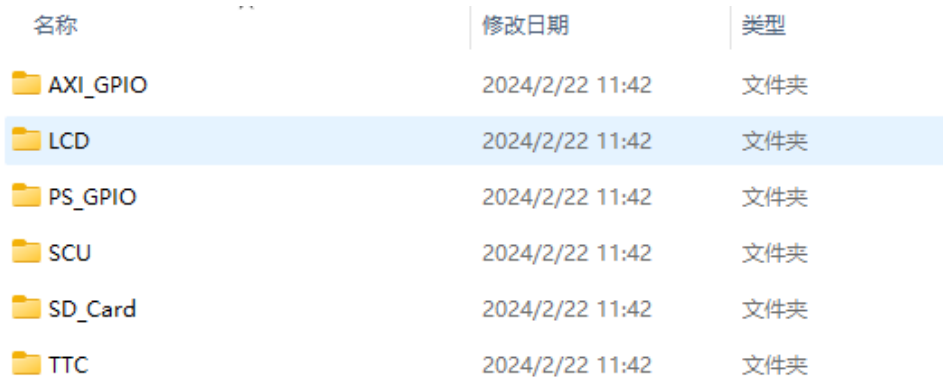

图 1-35 ACZ702_Lib 文件夹

(2) USER 文件夹

打开应用库资料，将 ACZ702_Lib 目录下的 USER 文件夹中的文件拷贝，粘贴到我们创建的新工程路径“*.../SD_LCD/src*”下方，结果如下图所示。

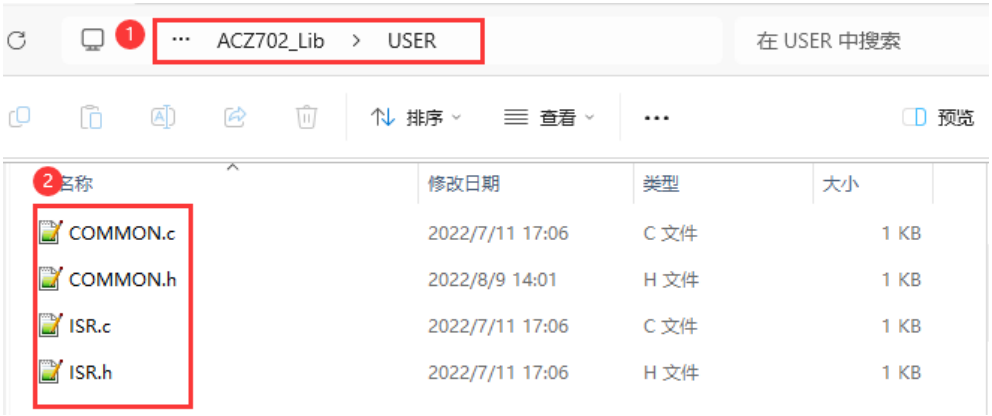

图 1-36 USER 文件夹

此时打开创建的工程，左键点击“SD_LCD”，然后按下 F5，出现图 1-37 所示内容，说明添加应用库成功。

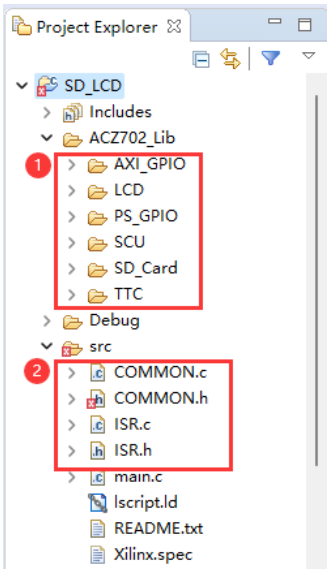

图 1-37 工程创建完成界面

1.3.3 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。

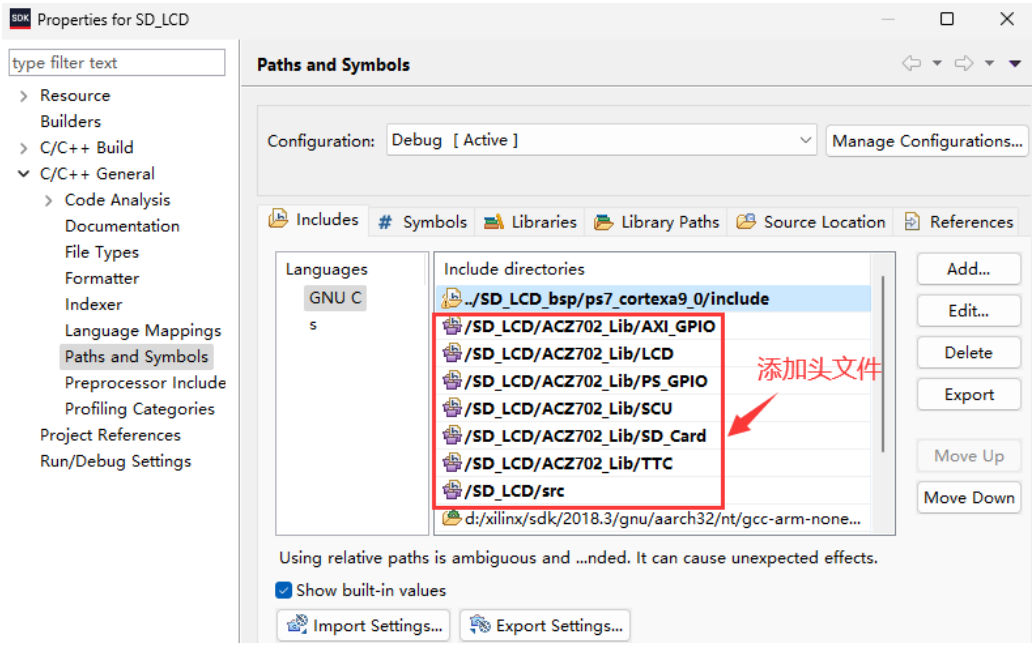

图 1-38 头文件路径

1.3.4 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。

1.3.5 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，因为需要使用按键控制图片的上下滑动，所以首先初始化 AXI_GPIO 和 PS_GPIO；考虑按键抖动问题， 添加上 TTC 定时器消除抖动，每次按键按下后，先进入 GPIO 中断处理函数，再延时 10ms；

然后初始化 LCD，首先显示第一张图片，在 while 循环里通过检测按键按下的累计次数，判断显示哪一张图片。

```
#include "COMMON.h"

int Last_Key_Cnt0 = -1;
int main(void)
{
    //初始化通用中断控制器
    ScuGic_Init();

    //初始化 AXI GPIO 驱动程序
    AXI_GPIO_Init(&AXI_GPIO0, XPAR_AXI_GPIO_0_DEVICE_ID);

    //初始化 PS 端的 GPIO 外设
    PS_GPIO_Init();

    //设置为输入，通道选择按键
    PS_GPIO_SetMode(PS_KEY, INPUT, 0);
    PS_GPIO_SetMode(KEY_S1, INPUT, 0);


    //初始化 AXI GPIO 中断、PS GPIO 中断
    AXI_GPIO_Intc_Init(&AXI_GPIO0,XPAR_FABRIC_AXI_GPIO_0_IP2INTC_IRPT
_INTR,XGPIO_IR_CH1_MASK, AXI_GPIO_IRQ_Handler);

    PS_GPIO_Int_Init();

    //设置引脚中断模式
    PS_GPIO_SetInt(PS_KEY, XGPIOPS_IRQ_TYPE_EDGE_FALLING);

    //设置 TTC0_1 为滴答定时器，用来做按键延时消抖，中断触发间隔为 10ms，等
待开启
    TTC_Tick_Init(&TTC0_1_Timer, XPAR_XTTCPS_1_DEVICE_ID,
XPS_TTC0_1_INT_ID,10000, TTC0_1_IRQ_Handler);
```

```
    //初始化 LCD
    LCD_Init();

    //显示 0.bmp 图片
    SD_to_LCD(FrameBuffer_Addr,"0.bmp");

    while(1)
    {
        //防止无限执行 SD_to_LCD 函数
        if (Key_Cnt0 != Last_Key_Cnt0) {
            Last_Key_Cnt0 = Key_Cnt0;
            switch(Key_Cnt0)
            {
                case 0:
                    SD_to_LCD(FrameBuffer_Addr,"0.bmp");
                    break;
                case 1:
                    SD_to_LCD(FrameBuffer_Addr,"1.bmp");
                    break;
                case 2:
                    SD_to_LCD(FrameBuffer_Addr,"2.bmp");
                    break;
                case 3:
                    SD_to_LCD(FrameBuffer_Addr,"3.bmp");
                    break;
                case 4:
                    SD_to_LCD(FrameBuffer_Addr,"4.bmp");
                    break;
                case 5:
                    SD_to_LCD(FrameBuffer_Addr,"5.bmp");
                    break;
                case 6:
                    SD_to_LCD(FrameBuffer_Addr,"6.bmp");
                    break;
                case 7:
                    SD_to_LCD(FrameBuffer_Addr,"7.bmp");
                    break;
                case 8:
                    SD_to_LCD(FrameBuffer_Addr,"8.bmp");
                    break;
                case 9:
                    SD_to_LCD(FrameBuffer_Addr,"9.bmp");
                    break;
            }
        }
    }
    return 0;
```

```
}
```

(2) COMMON.h

定义变量，用于根据按下次数显示不同图片；

```
uint32_t Key_Cnt0=0;//按键计数
```

(3) SD_CARD.c

该文件里面主要包含 24 位 BMP 图片显示的代码，首先通过 f_mount、f_open、f_read 获取图片的内容，图片的宽度是将 0x12、0x13、0x14、0x15 处的值组合起来，形成一个 32 位整数，该值就是宽度；同理，图片的长度是将 0x16、0x17、0x18、0x19 处的值组合起来。

最后，根据宽度和长度，将像素值（f_read 读取 3 个字节）逐个的放入 LCD 中。

```
#include "SD_CARD.h"

FATFS fs;
FIL file;
FRESULT Res;

//24 位 BMP 图显示
void SD_to_LCD(u8 *frame_lcd,char *filename)
{
    unsigned int width,height;
    int i, j;
    BYTE pixel[3];

    //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;
    }

    //打开图片文件
    Res = f_open(&file,filename,FA_READ);
    if(Res != FR_OK) {
        // 文件打开失败，需要根据返回的结果处理错误。
        printf("Open failed to return a value of \"%d\" !\n",Res);
        return -1;
    }

    //读取 BMP 文件头
    f_read(&file,file_head,54,NULL);
```

```
    //获取图片长度和高度
    width = (file_head[0x15]<<24) | (file_head[0x14]<<16) |
(file_head[0x13]<<8) | file_head[0x12];
    height = (file_head[0x19]<<24) | (file_head[0x18]<<16) |
(file_head[0x17]<<8) | file_head[0x16];

    //读取图片数据
    for(i = height - 1; i >= 0; i--){
        for(j = 0; j < width; j++){
            f_read(&file, &pixel, 3, NULL);

            // 计算此像素在 frame_lcd 中的位置
            int pixel_position = i * (width) * 3 + j * 3;

            frame_lcd[pixel_position] = pixel[0];
            frame_lcd[pixel_position + 1] = pixel[1];
            frame_lcd[pixel_position + 2] = pixel[2];
        }
    }

    //关闭文件
    f_close(&file);

    //刷新 Cache，更新数据
    Xil_DCacheFlush();
}
```

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：将 10 张图片提前放入 SD 卡中后，并装载到开发板；下载程序进行按键测试，观察 PS 按键与 PL 按键分别按下后，LCD 上的图片是否发生改变，如果变化说明电子相册功能实现。

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. Micro SD 卡 x1
4. Type-c 线缆 x1
5. 软排线一根

1.4.1 硬件连接

将 SD 卡通过读卡器连接到电脑，将图片文件复制到 SD 卡中；继续将 SD 卡插入到开发板卡槽当中。所以硬件连接如下图所示：

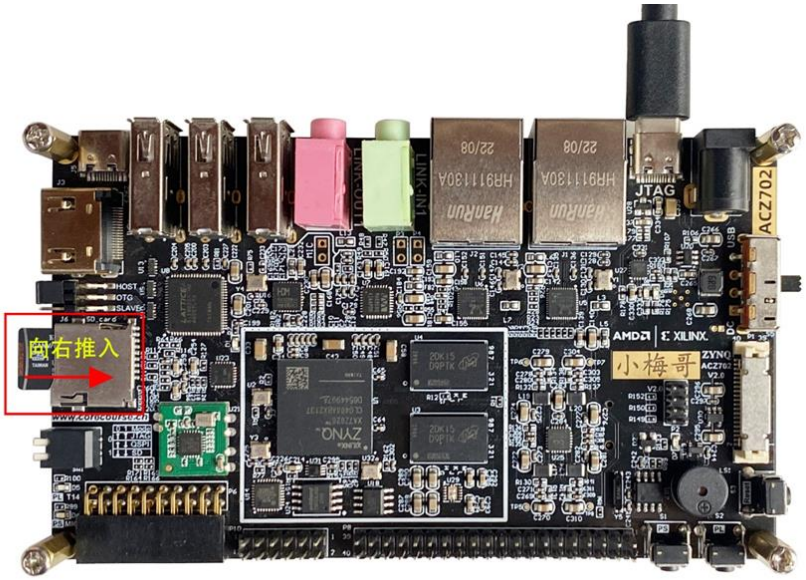

图 1-39 SD 卡连接图

使用软排线将 LCD 屏与开发板相连，注意要先将黑色压槽往上拨开。连接时将带有蓝色胶条的部分朝上，银白线条接口部分朝下，软排线与卡槽接口对应好后，按下压槽即完成了。

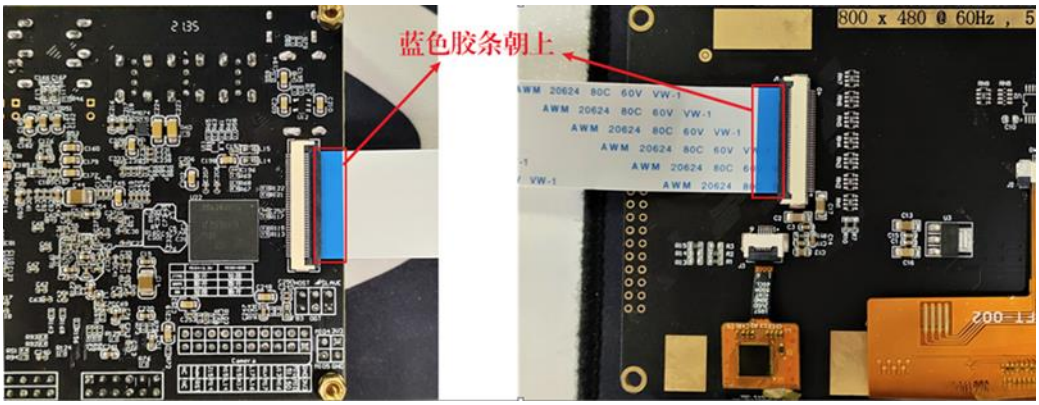

图 1-40 LCD 连接图

设计对供电的要求不高，因此可以直接使用 type-c 线连接。

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 1-41 和图

1-42 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

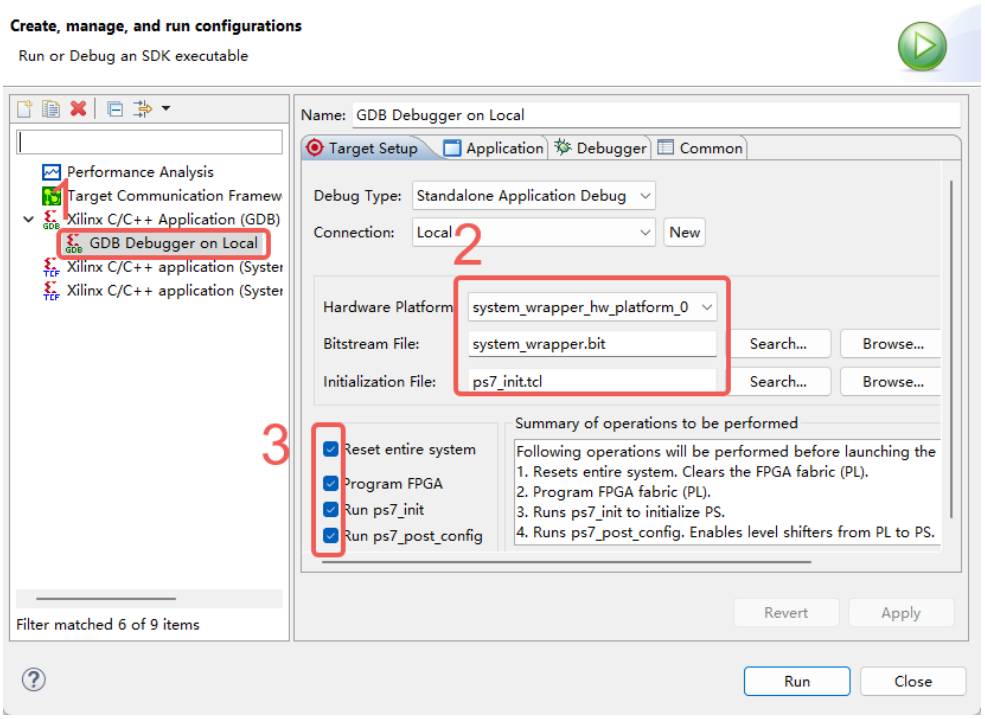

图 1-41 配置烧录任务

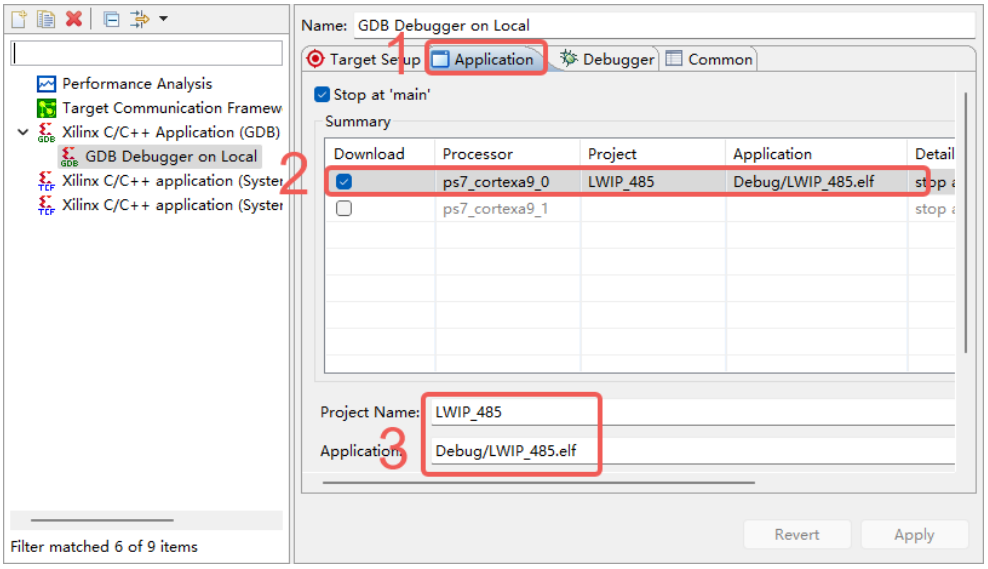

图 1-42 配置烧录任务

1.4.3 图片显示测试

打开工程，在 SDK 中运行程序。程序运行后在 LCD 屏幕上会出现第一张图片。

图 1-43 显示效果

此时，可以通过 S1 和 S2 两个按键控制图片上下翻页；

1.5 总结

在这次实验中，我们使用了 SD 卡和 LCD 模块来实现电子相册的功能，该实验为读者提供了一个初步的实践场景，展示了如何从 SD 卡读取图片。