

官方网站: www.corecourse.cn
技术群组:

6. DVP_Capture 模块：实现每两个数据拼接为 1 个 16 位的数据并按照写 RAM 或 FIFO 的接口形式输出。
7. usb_send_ctrl 模块：USB 数据输出控制模块，控制 USB 启动传输和 USB FIFO 的清除工作。
8. usb_stream_in 模块：USB 数据流发送模块，将最终采集到的数据通过 USB 发送出去。

本章需要设计的模块包括usb_cmd模块、usb_cmd_rx模块和usb_send_ctrl模块。

1.2 模块设计

下面将对本次实验需要重新设计的模块进行介绍。

1.2.1 接收转命令模块

接收转命令模块（usb_cmd）将 USB 传输过来的指令数据帧进行拆解，得到需要的指令数据传送给别的模块进行处理，该模块的结构框图如下图 1-2 所示。

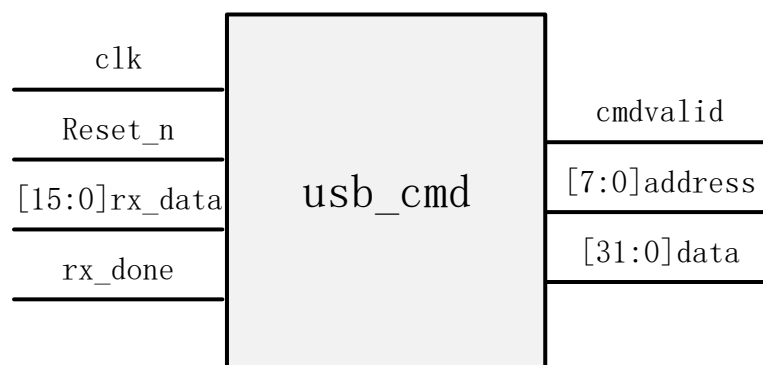


图 1-2 接收转命令模块框图

模块信号说明如下表 1-1 所示。

表 1-1 接收转命令模块信号说明表

信号名称	I/O	信号意义
clk	I	模块工作时钟
Reset_n	I	模块复位信号，低电平复位
rx_data [15:0]	I	USB 接收数据流模块接收到的 16 位数据
rx_done	I	USB 一次数据接收完成标志信号
cmdvalid	O	输出命令有效标志信号
address[7:0]	O	配置 AD7606 的寄存器地址信号
data[31:0]	O	写入到寄存器中的数据

USB 一次发送的命令帧内容为 32 个字节，为了实现通过 USB 修改这些寄存器的值，需要对发送一次的数据进行拆解才能实现，对于设计的数据帧，一帧数据一共 8 个字节，包含帧头、帧尾、地址段、数据段。帧格式如下表 1-2 所示：

表 1-2 帧格式说明表

数据	D0	D1	D2	D3	D4	D5	D6	D7
功能	帧头 0	帧头 1	地址 address	data[31:24]	data[23: 16]	data[15:8]	data[7:0]	帧尾
值	0x55	0xA5	XX	XX	XX	XX	XX	0xF0

从上表中可以看出，每帧数据一共 8 个字节，分别用 D0~D7 表示，其中，D0 和 D1 两个数据作为帧头，其值固定为 0x55、0xA5，D7 作为帧尾，其值固定为 0xF0。帧头和帧尾的作用是为了准确识别数据帧，确保接收的数据是我们需要分析的。D2 代表的是要操作的寄存器地址，D3 为要写入寄存器的数据的 24~31 位，D4 为要写入寄存器的数据的 16~24 位，D5 为要写入寄存器的数据的 8~15 位，D6 为要写入寄存器的数据的 0~7 位。

该模块的作用就是将 USB 接收到的数据拆解成上述帧格式，将 D2 作为地址 address 输出，指定修改哪个寄存器，D3~D6 共 32 位作为数据 data 输出。下面将对模块中的部分代码进行说明：

首先，当检测到了 rx_done 信号，data_str 连续 4 次存储 USB 接收到的数据，组成 32 字节的命令帧。代码如下所示：

```
always@(posedge Clk)
if(rx_done)begin
    data_str[3] <= #1 rx_data;
    data_str[2] <= #1 data_str[3];
    data_str[1] <= #1 data_str[2];
    data_str[0] <= #1 data_str[1];
end
```

最后判断得到的帧命令数据是否正确，当数据符合 D0 为 8'h55，D1 为 8'hA5，D7 为 8'hF0，则代表该数据格式正确，会生成一个指令正确信号 cmdvalid 输出到指令转控制模块，并将数据进行输出，代码如下所示：

```
always@(posedge Clk or negedge Reset_n)
if(!Reset_n) begin
    address <= #1 0;
    data <= #1 0;
    cmdvalid <= #1 0;
end else if(r_rx_done)begin
    if((data_str[0][7:0] == 8'h55) && (data_str[0][15:8] == 8'hA5) &&
(data_str[3][15:8] == 8'hF0))begin
        data[7:0] <= #1 data_str[3][7:0];
    end
end
```

```
data[15:8] <= #1 data_str[2][15:8];
data[23:16] <= #1 data_str[2][7:0];
data[31:24] <= #1 data_str[1][15:8];
address <= #1 data_str[1][7:0];
cmdvalid <= #1 1;

end
else
    cmdvalid <= #1 0; end
else
    cmdvalid <= #1 0;
```

1.2.2 指令转控制模块

指令转控制模块（usb_cmd_rx）将从接收转命令模块（usb_cmd）接收到的数据转换为相应的控制数据。本次实验需要一个启动传输寄存器，向该寄存器写入任意值便可以启动 USB 进行数据传输，该模块的基本结构框图如下图 1-3 所示。

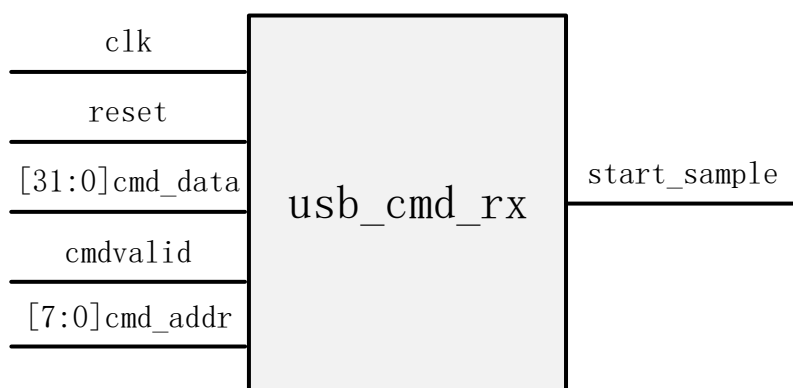


图 1-3 指令转控制模块结构框图

模块信号说明如下表 1-3 所示。

表 1-3 指令转控制模块信号说明表

信号名称	I/O	信号意义
clk	I	模块时钟信号
reset	I	模块复位信号，高电平有效
cmd_data[31:0]	I	写入到寄存器中的值
cmdvalid	I	命令有效标志信号
cmd_addr[7:0]	I	寄存器地址信号
start_sample	O	启动传输信号

本次实验设置启动传输寄存器的地址为0，当检测到cmd_addr为0的时候，产生启动传输信号 start_sample。代码如下所示：

```
always@(posedge clk or posedge reset)
begin
```

```

start_sample <= 1'b0;
end
else if(cmdvalid)begin
    case(cmd_addr)
        0: start_sample <= 1'b1;
        default;;
    endcase
end
else
    start_sample <= 1'b0;

```

1.2.3 USB 数据输出控制模块

USB 输出控制模块（usb_send_ctrl）主要用于启动 USB 数据传输以及清除 USB FIFO 中的数据方便下一次进行传输，该模块的基本框图如下图 1-4 所示。

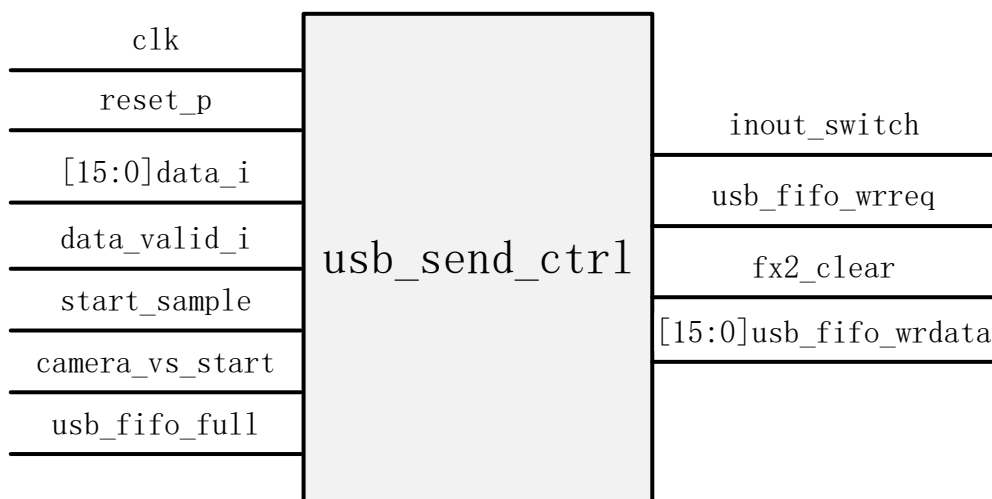


图 1-4 USB 数据输出控制模块结构框图

模块信号说明如下图 1-4 所示。

表 1-4 USB 输出控制模块信号说明表

信号名称	I/O	信号意义
clk	I	模块时钟信号
reset_p	I	模块复位信号，高电平有效
data_i[15:0]	I	16 位的摄像头采集到的数据
data_valid_i	I	摄像头采集到的数据有效信号
start_sample	I	启动数据传输信号
camera_vs_start	I	摄像头采集到的一副图像的起始脉冲信号
usb_fifo_full	I	usb_stream_in 模块中 FIFO 的满信号
inout_switch	O	0: read, 由 PC 向 FPGA 下发指令 1: write, 由 FPGA 向 fx2 芯片继而向 PC 上传数据
usb_fifo_wrreq	O	usb_stream_in 模块中 FIFO 的写请求信号
fx2_clear	O	USB 芯片中的清除信号

usb_fifo_wrdata[15:0]	O	需要交由 usb_stream_in 模块进行传输的 16 位图像数据
-----------------------	---	-------------------------------------

我们可以通过状态机实现该模块需要的功能，定义状态如下所示：

localparam IDLE	= 2'd0;	//空闲状态
localparam RESET_USB	= 2'd1;	//复位 USB 状态
localparam DATA_SEND_START	= 2'd2;	//数据发送开启状态
localparam DATA_SEND_WORKING	= 2'd3;	//数据发送状态

下面将对每个状态需要实现的功能以及相应代码的讲解。

1. IDLE 状态

在空闲状态的时候，将 fx2_clear 拉高，清除 USB 芯片中残留的数据，当检测到启动传输 start_sample 信号之后，将 inout_switch 拉高，代表此时需要由 FPGA 向电脑端传输数据，并且进入到复位 USB 的状态。代码如下所示：

```
if(start_sample) begin
    inout_switch <= 1'b1;
    state <= RESET_USB;
    fx2_clear <= 1'b1;
end
else begin
    inout_switch <= 1'b0;
    fx2_clear <= 1'b1;
    state <= state;
end
```

2. RESET_USB 状态

复位 USB 状态是为了清除 USB 中遗留的数据，当进入该状态之后，USB 复位计数器 rst_usb_cnt 开始计数，计数到一定值之后，拉低 fx2_clear，使 USB 内遗留数据能够充分清除。代码如下所示：

```
if(rst_usb_cnt >= 20'hffff0)begin
    rst_usb_cnt <= 0;
    state <= DATA_SEND_START;
end
else if(rst_usb_cnt >= 20'h7fff0)begin
    rst_usb_cnt <= rst_usb_cnt + 1'd1;
    fx2_clear <= 1'b0;
end
else
    rst_usb_cnt <= rst_usb_cnt + 1'd1;
```

3. DATA_SEND_START 状态

进入 DATA_SEND_START 状态之后，当检测到一副图像最开始的信号 camera_vs_start 到来之后，进入 DATA_SEND_WORKING 状态进行数据传输，

代码如下所示：

```
if(camera_vs_start) begin
    state <= DATA_SEND_WORKING;
end
else
    state <= state;
```

4. DATA_SEND_WORKING 状态

进入 DATA_SEND_WORKING 状态之后，当检测到上位机一段时间之内没有读取 USB 中数据的时候，跳回 IDLE 状态，等待下一次的传输。代码如下所示：

```
if(read_cnt >= 32'hfffffff)
begin
    state <= IDLE;
end
else
    state <= DATA_SEND_WORKING;
```

当状态处于 DATA_SEND_WORKING 状态，并且usb_stream_in模块中FIFO写满之后（usb_fifo_full 为高），read_cnt 开始计数，当计数一段时间之后，usb_fifo_full还处于高电平，说明此时USB中的数据并未读出去，也就是说此时上位机处于关闭状态，当重新打开上位机的时候，会从 IDLE 状态开始，清除USB中的数据，重新进行数据传输，这样上位机在进行图像显示的时候才不会出现错位的情况。read_cnt产生的代码如下所示：

```
always@(posedge clk or posedge reset_p)
if(reset_p)
    read_cnt <= 32'd0;
else if(read_cnt >= 32'hfffffff)
    read_cnt <= 32'd0;
else if((state == DATA_SEND_WORKING) && usb_fifo_full)
    read_cnt <= read_cnt +1'd1;
else
    read_cnt <= 32'd0;
```

当状态处于 DATA_SEND_WORKING 状态，并且摄像头采集到的数据有效（data_valid_i为高）的情况下，产生usb_stream_in模块中FIFO的写请求信号，并且将此时的数据 data_i 写入到 FIFO 中，最终交由 USB 进行传输，代码如下所示：

```
always@(posedge clk or posedge reset_p)
if(reset_p)
    usb_fifo_wrreq <= 1'd0;
else if((state == DATA_SEND_WORKING) && data_valid_i)
```

```
begin
    usb_fifo_wrreq <= 1'd1;
    usb_fifo_wrdata <= data_i;
end
else
begin
    usb_fifo_wrreq <= 1'd0;
    usb_fifo_wrdata <= 16'd0;
end
```

模块设计完成之后，只需要在顶层文件中对各个模块之间的接口信号进行连接，完整的顶层文件代码请自行查看例程文件。

1.3 板级验证

经过以上工作，代码设计部分的任务已经全部完成，接下来就可以进行板级验证了。本次实验我们提供有相应的上位机软件“小梅哥 USB 摄像头”，下载完程序之后，便可以打开该上位机软件查看接收到的数据是否正确。

1.3.1 系统所需硬件

1. ACZ702 开发板一块
2. USB 线一根
3. ACM68013 模块一个
4. Type-C 下载线一根
5. OV5640 摄像头一个
6. 电源线一根（可选）

1.3.2 硬件连接

本次设计系统硬件连接如下图 1-5 所示：

1. 使用 Type-C 线连接开发板调试接口（靠近电源接口）和电脑 USB 口
2. 将开发板电源拨码开关拨到对应侧
3. ACM68013 模块连接至 40 pin 的排针上，靠右连接，1 脚和 1 脚对应
4. 使用 USB 线连接 ACM68013 模块和电脑
5. 将摄像头连接至开发板的摄像头接口上

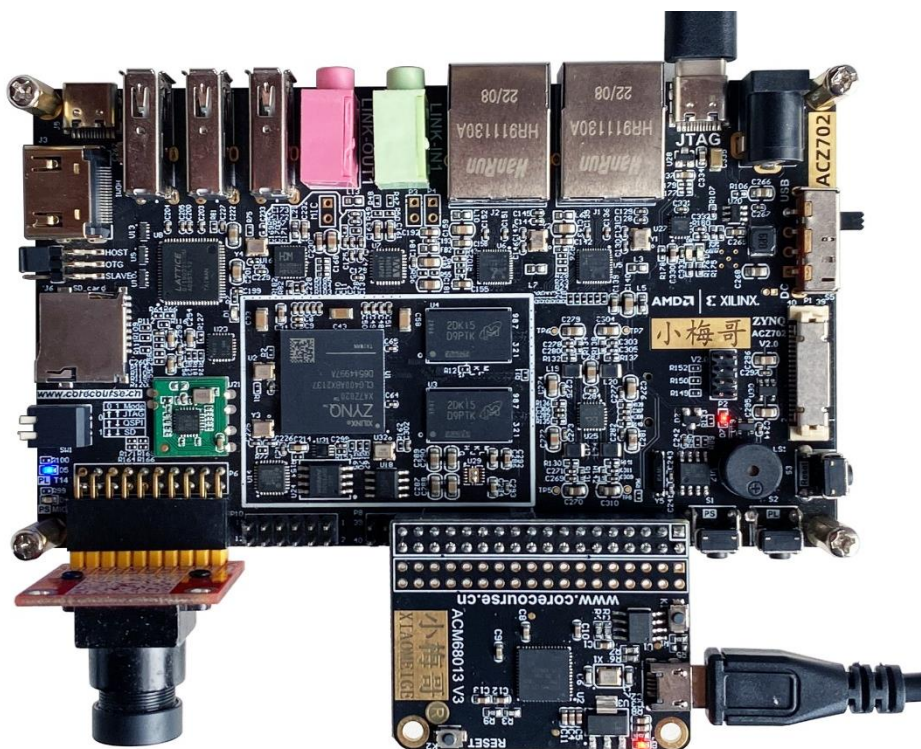


图 1-5 硬件连接图

1.3.3 功能验证

功能验证步骤如下所示：

1. 全编译工程得到 bit 文件。
2. 使用 bit 文件配置 FPGA 开发板，下载完成之后，开发板右边的 PL 侧的 LED 灯会被点亮，如下图 1-6 所示，此时说明摄像头初始化成功。

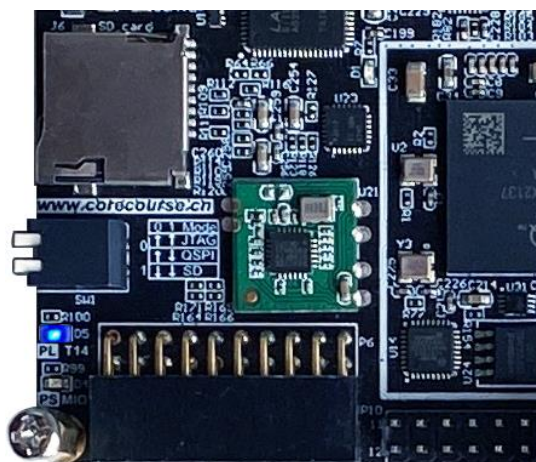


图 1-6 程序下载成功之后，LED 灯被点亮

3. 使用上位机软件显示图像。

店铺：<https://xiaomeige.taobao.com>

技术博客：<http://www.cnblogs.com/xiaomeige/>

官方网站：www.corecourse.cn

技术群组：

“小梅哥 USB 摄像头”上位机最新下载链接：

[小梅哥 USB 摄像头上位机（RGB）](#)

<http://www.corecourse.cn/forum.php?mod=viewthread&tid=29305>

打开软件之后，初始界面如图 1-7 下所示。点开软件之后，首先观察设备名称的开头是否为“(0x04B4 - 0x00F1) Cypress FX3 USB”，如果不是，请检查 USB 线的连接，是否安装 USB 驱动以及是否下载了 USB 芯片的固件，固件请烧写工程文件夹下的“slave_for_adc.iic”文件，如果“slave_for_adc.iic”固件实验不成功，请烧写“slave_for_adc_clk_not.iic”固件。

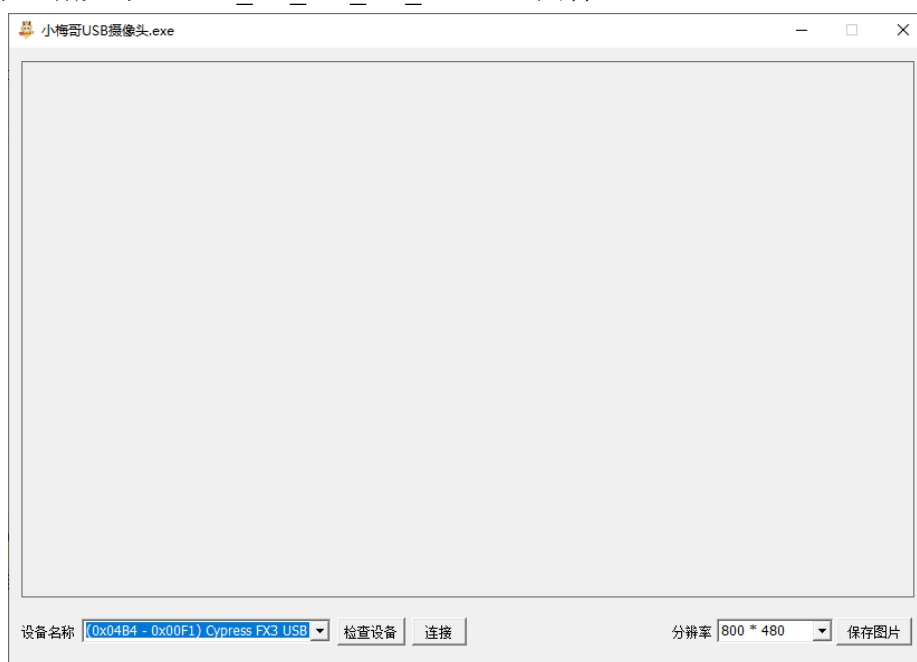


图 1-7 小梅哥 USB 摄像头上位机初始界面

检测到设备之后，点击连接，便可以看到摄像头采集到的实时图像，如下图 1-8 所示。

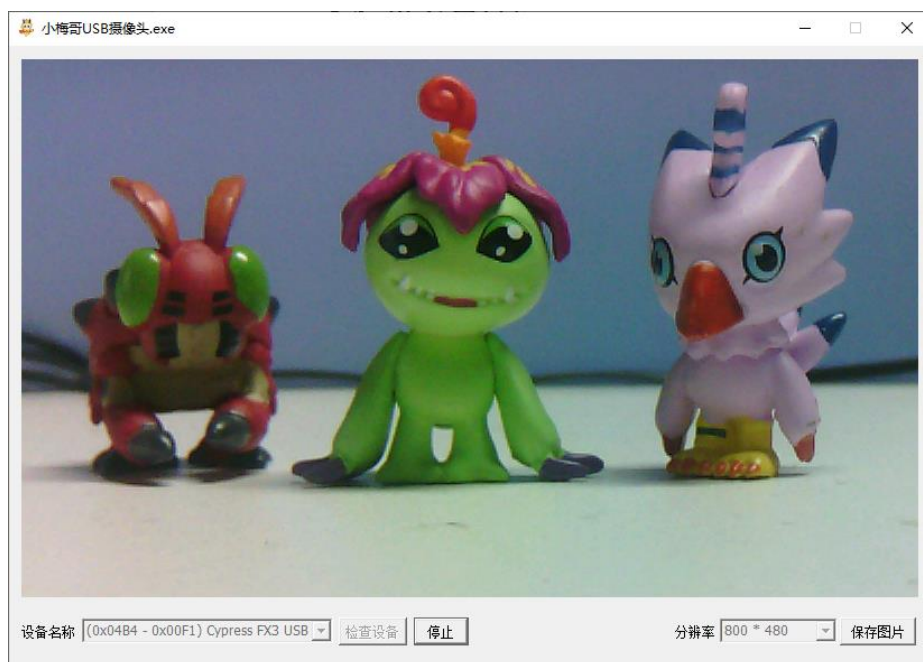


图 1-8 上位机显示摄像头采集到的图像数据

1.4 思考与总结

本节实验介绍了通过 USB2.0 传输 OV5640 采集的图像数据，并且通过相关的上位机软件实时显示摄像头采集到的数据。

本次实验需要注意的是，在传输图像数据的时候需要考虑到 USB 底层采用的数据编码方式为 8b-10b 编码，因此有效数据带宽只有 80%，也就是 48MB/S。另外，由于 480Mbps 的传输速度由挂在同一个 USB 根节点上的所有设备共享，因此当同一个根节点下，还有其他 USB 设备时，传输带宽会进一步分散，所以留给单一设备的有效传输带宽是很难达到 48MB/S 的，实测一般在 10~30MB/S 左右。本实验中设置的分辨率为“800*480”，帧率 15fps，其需要的 USB 的传输速度大约为 11 MB/s ($((800*480*2*15) / (1024*1024))$)，在 USB2.0 传输速度范围之内，理论上可以达到更高的分辨率，比如 1280*720，但是在实际测试过程中发现，不同的电脑会出现丢包的情况，而 800*480 的分辨率基本都不会出现丢包的情况，所以我们默认设置 800*480 的分辨率，用户如果感兴趣的话，可以自行修改分辨率进行尝试。