1 基于 ACM68013 模块的 OV5640 RGB 摄像头实验

1.1 系统整体设计

本次实验，我们将使用 OV5640 摄像头采集图像数据，然后在 AC620 开发板上外接一个 ACM68013 模块，通过该模块将图像数据传输至电脑，最终通过电脑端的上位机软件实时显示图像数据。系统的整体框图如下图 1-1 所示。

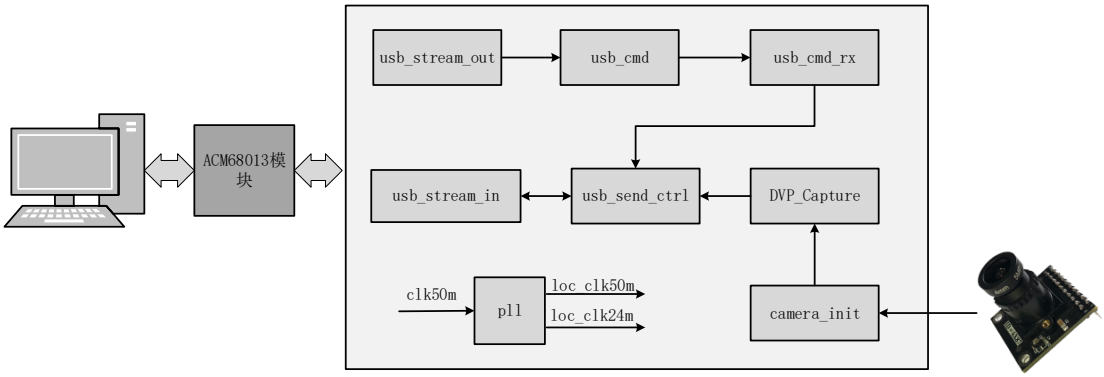

图 1-1 OV5640 RGB 摄像头采集 USB 传输整体设计框图

对于上图中各个模块的功能介绍如下：

1. PLL 模块：锁相环模块，生成本次实验每个模块所需要的工作时钟，其中输入时钟为 50M 的系统时钟，由开发板上的晶振提供，输出 24M 的时钟给到摄像头模块使用，输出 50M 的时钟给其他模块使用。
2. usb_stream_out 模块：USB 数据流发送控制模块，不断的将端点 2 中的数据读取出来，数据读取后直接作为端口输出。
3. usb_cmd 模块：接收转命令模块，对 USB 接收到的数据进行分析，提取出每个控制命令帧。
4. usb_cmd_rx 模块：指令转控制模块，将从接收转命令模块接收到的数据转换为相应的控制 USB 启动传输的信号。
5. camera_init 模块：摄像头初始化模块，完成对 OV5640 众多模式设置寄存器的写入操作。
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。

下面将对各个模块的设计进行说明。

1.2 模块设计

1.2.1 USB 数据流发送控制模块

usb_stream_out 的功能是不断地将端点 2 中的数据读取出来，数据读取后直接作为端口输出。该模块的接口图如下图 1-2 所示。

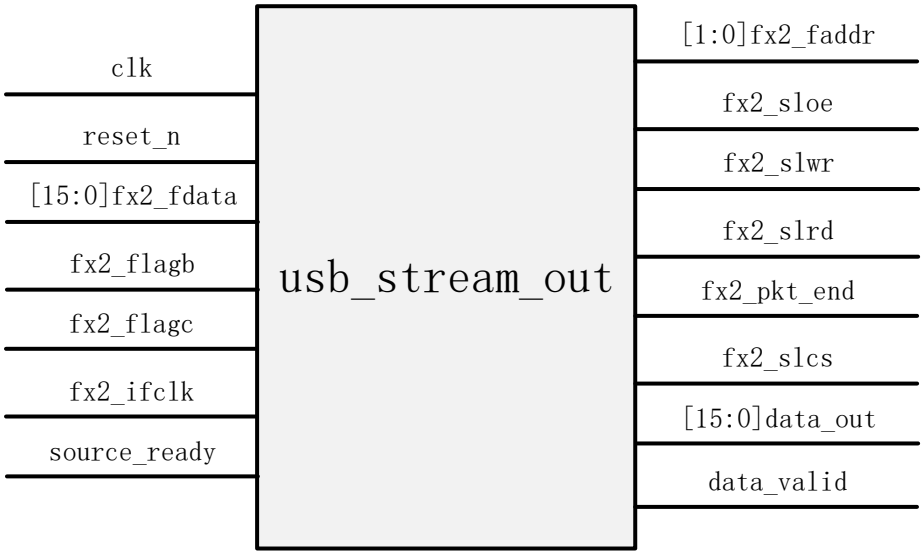

图 1-2usb_stream_out 模块的基本框图

对于模块的信号说明如下表 1-1 所示：

表 1-1usb_stream_out 模块接口信号表

信号名称	I/O	信号意义
clk	I	时钟信号
reset_n	I	复位信号，低电平有效
fx2_fdata[15:0]	I	FX2 型 USB2.0 芯片的 SlaveFIFO 的数据线
fx2_flagb	I	FX2 型 USB2.0 芯片的端点 2 空标志
fx2_flagc	I	FX2 型 USB2.0 芯片的端点 6 满标志
fx2_ifclk	I	FX2 型 USB2.0 芯片的接口时钟信号
source_ready	I	外部数据消费者数据接收允许信号，例如 FPGA 中的缓存 FIFO 中有足够的空间存储一帧 USB 数据，则允许从 Slave FIFO 中去读取数据
fx2_faddr[1:0]	O	FX2 型 USB2.0 芯片的 SlaveFIFO 的 FIFO 地址线
fx2_sloe	O	FX2 型 USB2.0 芯片的 SlaveFIFO 的输出使能信号，低电平有效

fx2_slwr	O	FX2 型 USB2.0 芯片的 SlaveFIFO 的写控制信号，低电平有效
fx2_slrd	O	FX2 型 USB2.0 芯片的 SlaveFIFO 的读控制信号，低电平有效
fx2_pkt_end	O	数据包结束标志信号
fx2_slcs	O	FX2 型 USB2.0 芯片的 SlaveFIFO 的片选信号，当 SLCS 输出高时，不可进行数据传输
data_out	O	经过 FPGA 接收了的 USB 数据
data_valid	O	经过 FPGA 接收了的 USB 数据有效标志信号

上表中以 FX2 开头的信号，都是与 FX2 进行连接的信号，实现对 FX2 的 SlaveFIFO 读操作。data_out 和 data_valid 是读取到的数据与数据有效信号，使用这两个信号，可以非常方便地将从 FX2 中读取到的数据存储到 RAM、FIFO 中，以实现用户自定义应用。

同时，考虑到基于 FX2 和 FPGA 进行实际应用开发时，系统对 USB 的数据传输速率和传输时间有要求，所以必须在 FPGA 这边主动去控制整个通信的吞吐率。为此该逻辑特意设置了一个名为 source_ready 的信号，当该信号有效时，才允许从 SlaveFIFO 中读取一包数据。

我们使用状态机的方式实现该模块的功能，定义状态如下所示，分别为空闲状态、读状态、等待状态。

```
parameter [1:0] stream_out_idle        = 2'd0;
parameter [1:0] stream_out_read        = 2'd1;
parameter [1:0] stream_out_wait        = 2'd2;
```

整体的状态转移图如下图 1-3 所示。

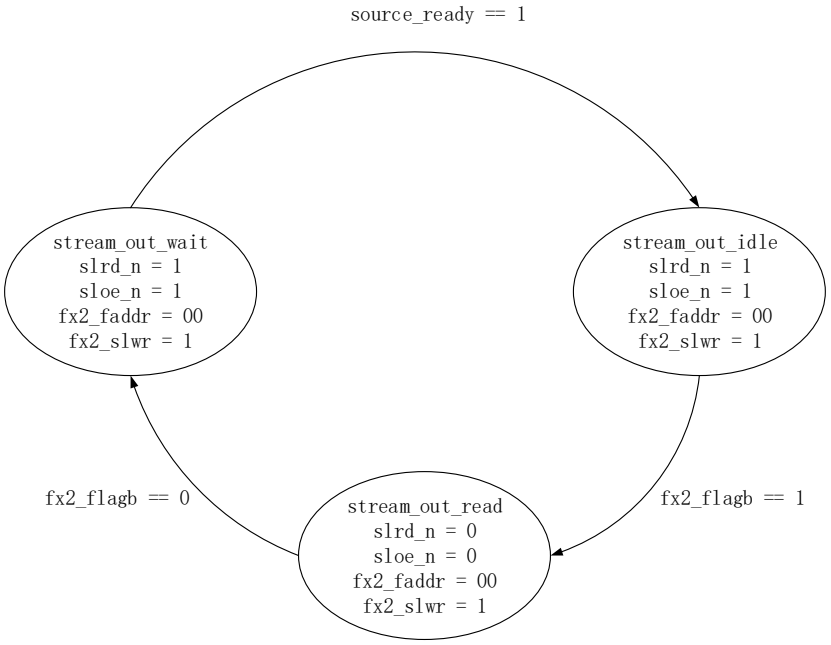

图 1-3 usb_stream_out 状态转移图

下面将对每个状态的代码设计进行简单的说明。

1. stream_out_wait 状态

在 stream_out_wait 状态的时候，等待数据使用逻辑允许从 USB 中读取新的数据，当

source_ready 信号为高电平的时候，也就是外部数据消费者数据允许接收数据，例如 FPGA 中的缓存 FIFO 中有足够的空间存储一帧 USB 数据，则允许从 Slave FIFO 中去读取数据，然后状态跳转至 stream_out_idle 状态等待读取数据，代码如下所示：

```
stream_out_wait:begin    //等待数据使用逻辑允许从 USB 中读取新的数据
    if(source_ready)
        next_stream_out_state = stream_out_idle;
    else
        next_stream_out_state = stream_out_wait;
end
```

2. stream_out_idle 状态

stream_out_idle 状态是指 slrd_n 和 sloe_n 均为高电平的闲置状态。只要端点 2 为空的标志 fx2_flagb 为低电平（被激活），状态机将处于 stream_out_idle 状态，fx2_flagb 变成高电平（端点 2 非空）之后，状态机将从 stream_out_idle 状态跳转至 stream_out_read 状态，开始读取数据，代码如下所示：

```
stream_out_idle:begin    //等待 FX2 的 Slave FIFO 中端点 2 非空
    if(fx2_flagb == 1'b1)
        next_stream_out_state = stream_out_read;
    else
        next_stream_out_state = stream_out_idle;
end
```

3. stream_out_read 状态

在 stream_out_read 状态中，FPGA 将连续从端点 2 的 FIFO 中读取数据。当 fx2_flagb 变成低电平，也就是端点 2 的 FIFO 为空之后，跳转回 stream_out_wait 状态，代码如下所示：

```
stream_out_read:begin    //从端点 2 中读取数据
    if(fx2_flagb == 1'b0)
        next_stream_out_state = stream_out_wait;
    else
        next_stream_out_state = stream_out_read;
end
```

在读取数据的时候，需要激活 slrd_n 信号和 sloe_n 信号才能成功读取数据，slrd_n 信号和 sloe_n 信号都是低电平有效，代码如下所示：

```
//产生读 Slave FIFO 数据请求信号
always@(*)begin
```

```
if((current_stream_out_state==stream_out_read)&(fx2_flagb==1'b1))
begin
        slrd_n = 1'b0;
        sloe_n = 1'b0;
    end else begin
        slrd_n = 1'b1;
        sloe_n = 1'b1;
    end
    end
```

1.2.2 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 1-4 所示。

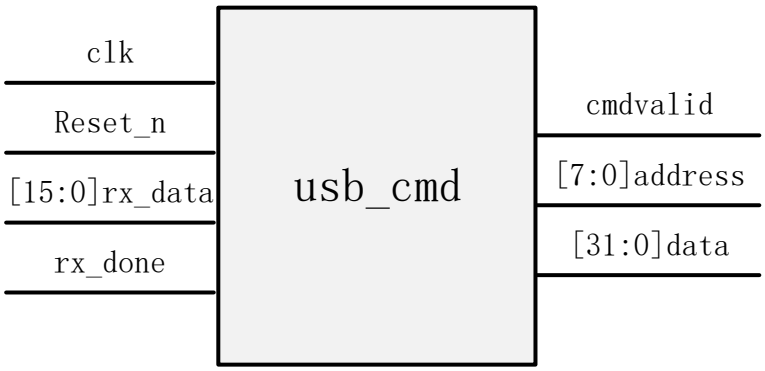

图 1-4 接收转命令模块框图

模块信号说明如下表 1-2 所示

表 1-2 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	配置摄像头开始采集的寄存器地址信号
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 1-3 所示：

表 1-3 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];
        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;
```

1.2.3 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 1-5 所示。

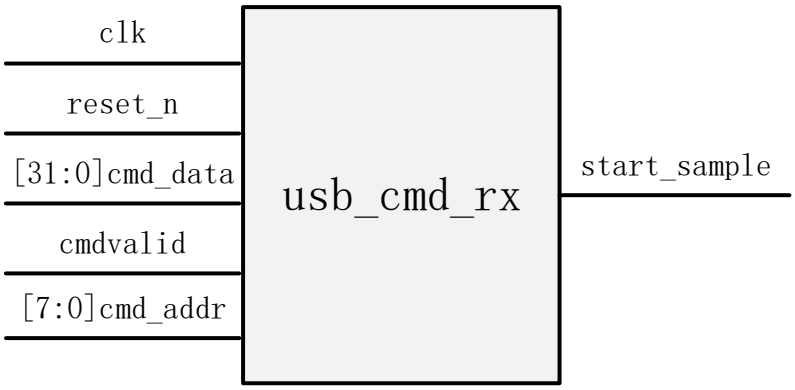

图 1-5 指令转控制模块结构框图

模块信号说明如下表 1-4 所示。

表 1-4 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_n	I	模块复位信号，低电平有效
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 negedge reset_n)
if(!reset_n)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;
```

1.2.4 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 1-6 所示。

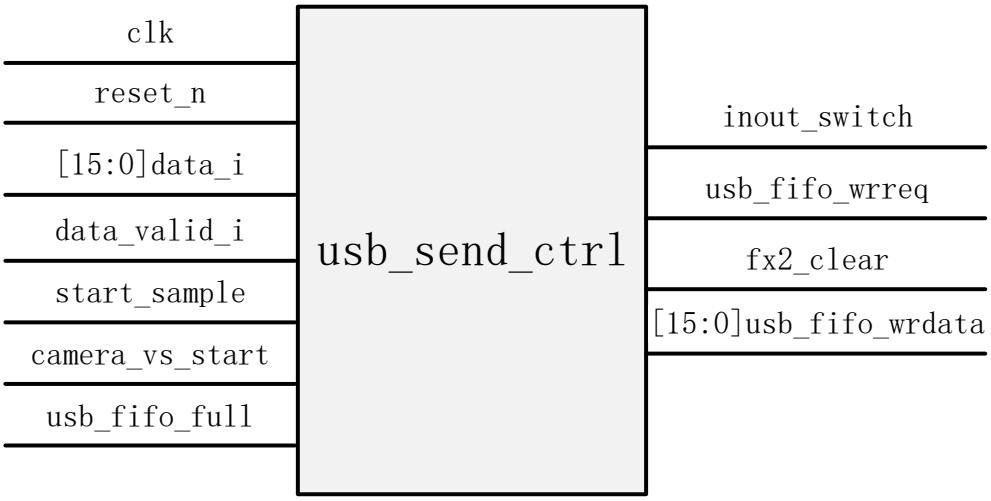

图 1-6 USB 数据输出控制模块结构框图

模块信号说明如下表 1-5 所示。

表 1-5 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_n	I	模块复位信号，低电平有效
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'hffffff)
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 negedge reset_n)
if(!reset_n)
    read_cnt <= 32'd0;
else if(read_cnt >= 32'hffffff)
    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 negedge reset_n)
if(!reset_n)
    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
```

1.2.5 USB 数据流发送控制模块

usb_stream_in 模块主要用于实现 FPGA 通过 FX2 芯片发送数据给 FPGA，实现一些常见的输入型应用，例如 FPGA 实时采集 ADC 数据，然后经由 FX2 发送到 PC，再由 PC 进一步处理，或者 FPGA 接收摄像头数据，经由 FX2 传给 PC，由 PC 进行图像显示等，其模块图如下图 1-7 所示。

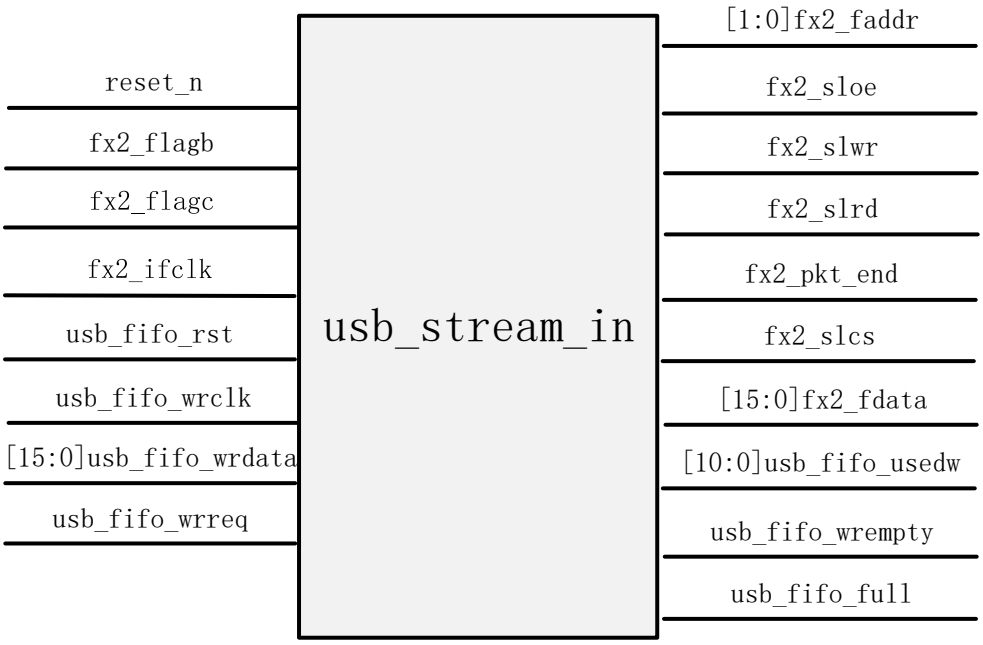

图 1-7usb_stream_in 模块结构图

usb_stream_in 模块的信号说明如下表 1-6 所示。

表 1-6 usb_stream_in 模块信号说明表

信号名称	I/O	信号意义
reset_n	I	复位信号，低电平有效
usb_fifo_rst	I	usb_stream_in 模块内部 FIFO 的清除信号
fx2_flagb	I	FX2 型 USB2.0 芯片的端点 2 空标志
fx2_flagc	I	FX2 型 USB2.0 芯片的端点 6 满标志
fx2_ifclk	I	FX2 型 USB2.0 芯片的接口时钟信号
usb_fifo_wrclk	I	usb_stream_in 模块内部 FIFO 的写入时钟信号
usb_fifo_wrdata [15:0]	I	usb_stream_in 模块内部 FIFO 的写数据信号
usb_fifo_wrreq	I	usb_stream_in 模块内部 FIFO 的写使能信号
fx2_fdata[15:0]	O	FX2 型 USB2.0 芯片的 SlaveFIFO 的数据线
fx2_faddr[1:0]	O	FX2 型 USB2.0 芯片的 SlaveFIFO 的 FIFO 地址线
fx2_sloe	O	FX2 型 USB2.0 芯片的 SlaveFIFO 的输出使能信号，低电平有效
fx2_slwr	O	FX2 型 USB2.0 芯片的 SlaveFIFO 的写控制信号，低电平有效
fx2_slrd	O	FX2 型 USB2.0 芯片的 SlaveFIFO 的读控制信号，低电平有效
fx2_pkt_end	O	数据包结束标志信号

usb_fifo_usedw[10:0]	O	usb_stream_in 模块内部 FIFO 的写入计数信号
usb_fifo_wrempty	O	usb_stream_in 模块内部 FIFO 为空信号
usb_fifo_full	O	usb_stream_in 模块内部 FIFO 写满信号

usb_stream_in 模块功能的实现：FPGA 监控端点 6 的满标志（fx2_flagc），当 fx2_flagc 为高电平的时候，FPGA 会连续将数据写入到端点 6 的 FIFO 内，当 fx2_flagc 为低电平，也就是端点 6 的 FIFO 写满之后，FPGA 将暂停写操作。

本次设计我们可以通过状态机的方式实现 usb_stream_in 模块的功能，定义状态如下所示，分别是空闲状态和写状态。

```
parameter stream_in_idle   = 1'b0;
parameter stream_in_write  = 1'b1;
```

状态转移图如下图 1-8 所示，下面我们对每个状态的代码设计和功能进行说明。

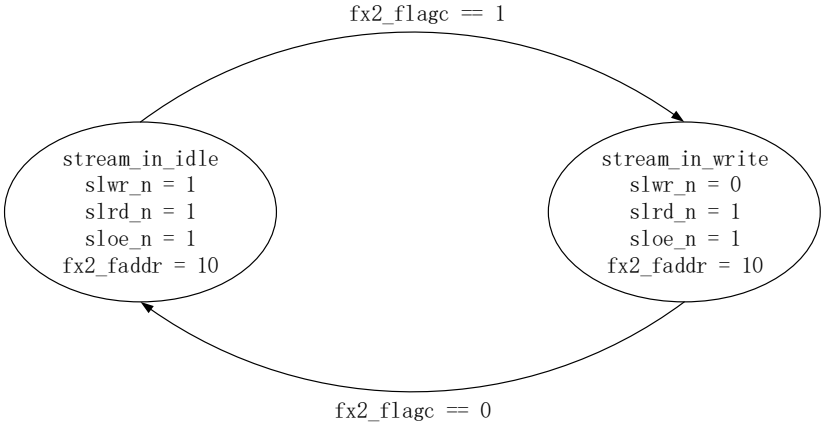

图 1-8usb_stream_in 模块状态转移图

1. stream_in_idle 状态

stream_in_idle 是指 slwr_n 处在高电平时的闲置状态，只要端点 6 的满标志（fx2_flagc）为低电平（被激活），则状态会处于 stream_in_idle 状态。slwr_n 信号产生的代码如下所示：

```
//write control signal generation
always@(*)begin
    if((current_stream_in_state==stream_in_write)&(fx2_flagc==1'b1))
        slwr_n <= 1'b0;
    else
        slwr_n <= 1'b1;
end
```

在 fx2_flagc 变成高电平的时候，代表此时端点 6 的 FIFO 还未写满，可以向 FIFO 中写入数据，状态机将从 stream_in_idle 状态跳转至 stream_in_write 状态，代码如下所示：

```
stream_in_idle:begin
```

```
    if(fx2_flagc == 1'b1)
        next_stream_in_state = stream_in_write;
    else
        next_stream_in_state = stream_in_idle;
end
```

2. stream_in_write 状态

在 stream_in_write 状态中，将会激活 slwr_n 信号，FPGA 将会持续向端点 6 的 FIFO 中写入数据，在 fx2_flagc 变成低电平的时候（FIFO 被写满），状态机将返回到 stream_in_idle 状态，代码如下所示：

```
stream_in_write:begin
    if(fx2_flagc == 1'b0)
        next_stream_in_state = stream_in_idle;
    else
        next_stream_in_state = stream_in_write;
    end
```

然后增加一个双时钟 FIFO，输入和输出位宽为 16，深度为 1024，随后将 FIFO 例化至代码中，并进行信号连接，代码如下所示：

```
fifo fifo(
    .aclr(usb_fifo_rst),
    .data(usb_fifo_wrdata),
    .rdclk(fx2_ifclk),
    .rdreq(~slwr_n),
    .wrclk(usb_fifo_wrclk),
    .wrreq(usb_fifo_wrreq),
    .q(data_out1),
    .rdempty(empty),
    .rdfull(),
    .rdusedw(),
    .wrempty(),
    .wrfull(usb_fifo_full),
    .wrusedw(usb_fifo_usedw)
    );
```

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，关于 DVP_Capture 模块和 camera_init 模块的设计请自行查看 620 开发板资料中盘 A_AC620V2 开发板标准配套资料\01_教材文档中 02A 文档中的相关内容，这里设置摄像头输出的图片分辨率为“1280*720”。

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”，该软件位于压缩包 ac620_fx2_stream_in_ov5640_rgb 中，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 硬件连接

读者将 OV5640 摄像头和 USB 模块连接在开发板上，然后编译工程，将生成的 sof 文件下载至开发板上，进行验证，sof 文件下载成功之后，如下图 1-9 所示，开发板上的 LED0 和 LED1 将会被点亮，LED0 被点亮说明 PLL 锁相环工作正常，LED1 被点亮说明摄像头初始化完成。

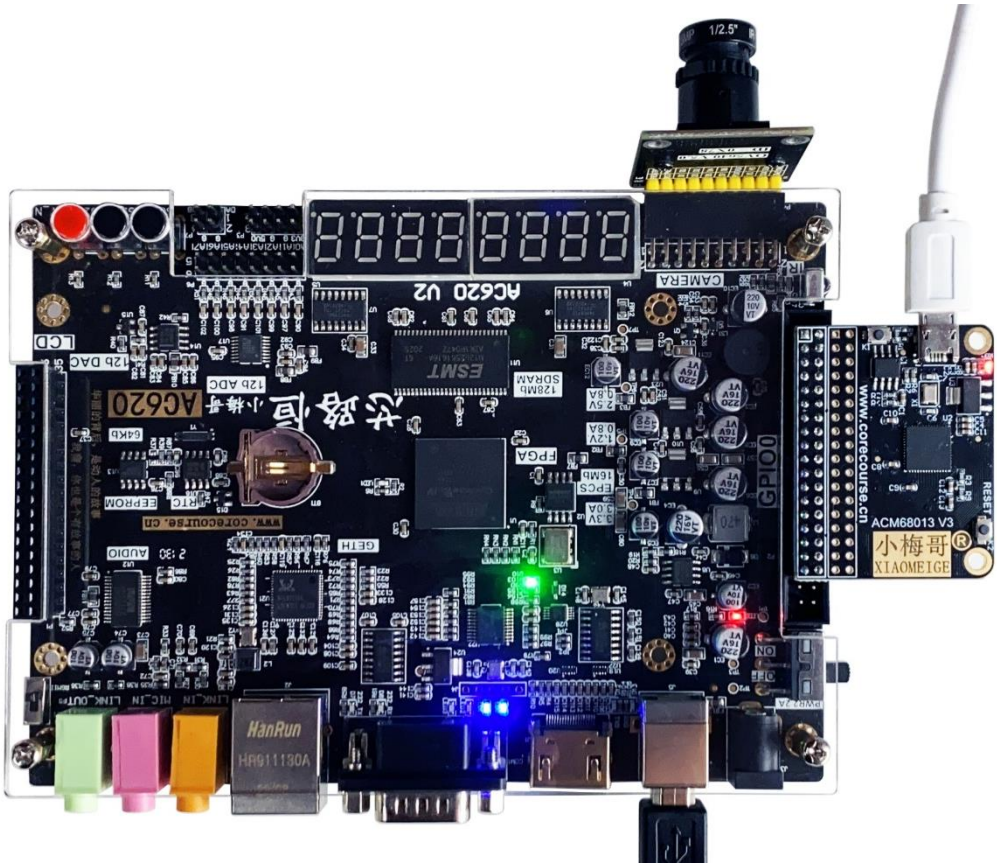

图 1-9 sof 文件下载成功之后示意图

1.3.2 烧写固件

打开 CyControl 软件，点击设备，烧写我们工程文件夹下的“slave_for_adc.iic”文件至 EEPROM 中，如下图 1-10 所示。

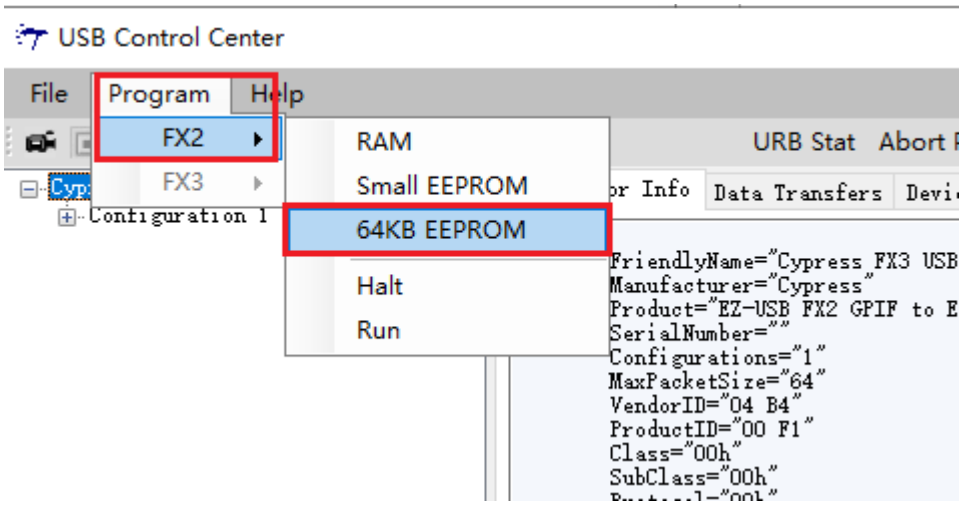

图 1-10 烧写固件

然后在弹出的对话框中选择需要烧写的固件，如下图 1-11 所示。

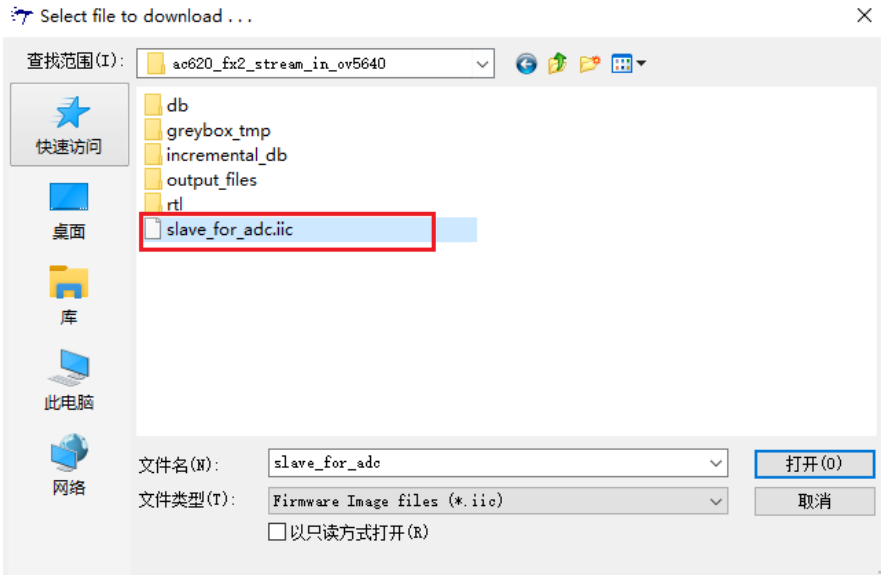

图 1-11 选择需要烧写的固件

烧写成功之后，软件会显示“Programming successed”，如下图 1-12 所示，然后按一下 USB 模块上的 RESET 按键。

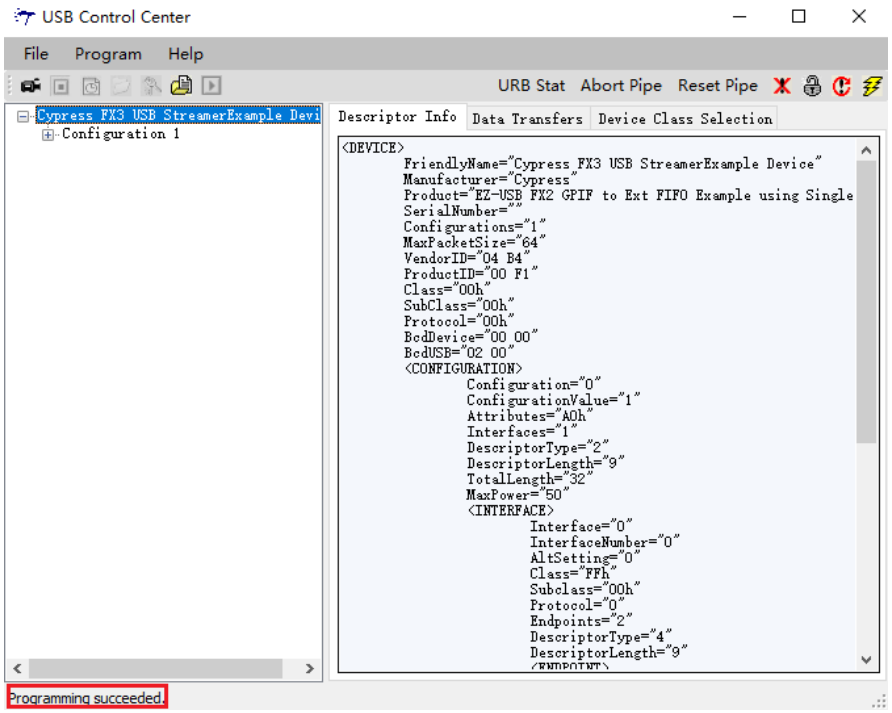

图 1-12 固件烧写成功之后示意图

1.3.3 上位机软件显示图像

打开盘 A ACX720 开发板标准配套资料\05_常用软件下的“小梅哥 USB 摄像头”上位机，初始界面如下图 1-13 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，相关内容可以在盘 A_AC620V2 开发板标准配套资料\01_教材文档\02A 文件中查看。

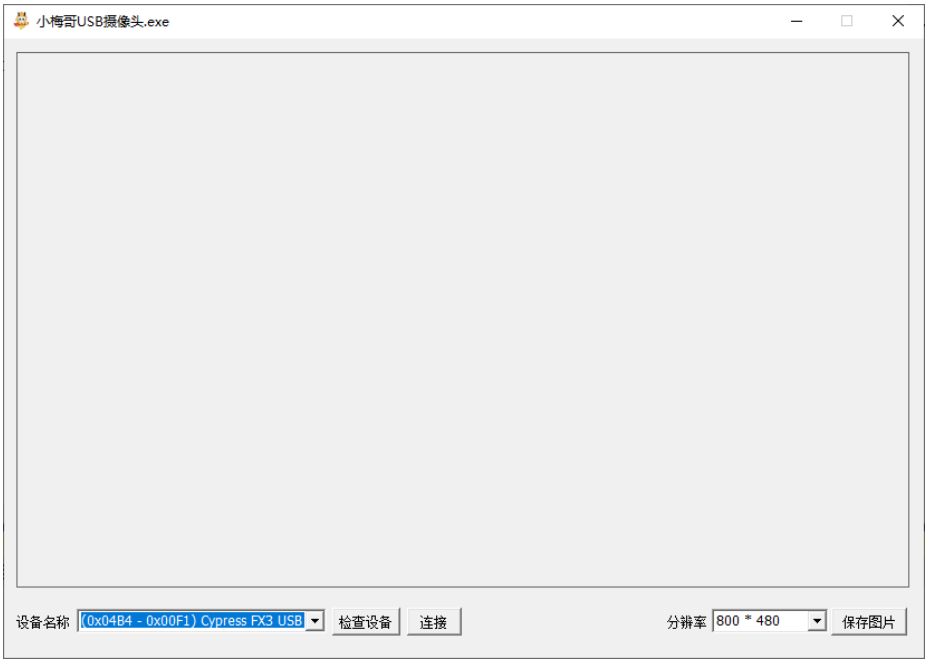

图 1-13 小梅哥 USB 摄像头上位机初始界面

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 1-14 所示。

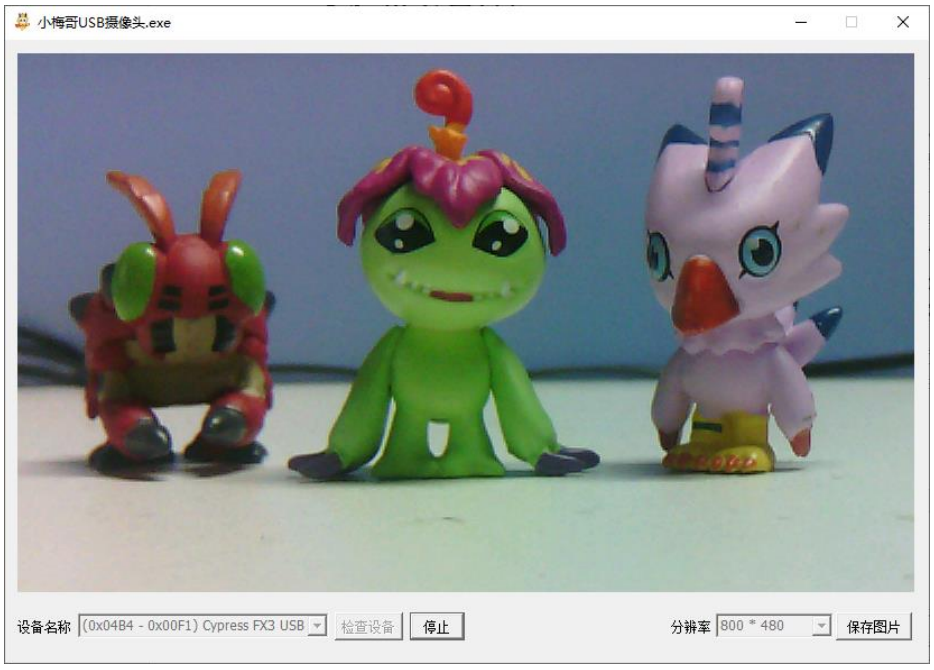

图 1-14 上位机显示摄像头采集到的图像数据

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。