多‘彩’娃娃

[redacted]

第一部分　设计概述

1.1 设计目的

本设计旨在开发一款具备多功能、高性能的数据采集卡，通过 FPGA 技术实现对不同类型信号的实时采集、处理和传输，满足广泛的工业需求。该采集卡支持 HDMI 视频、网络信号和 ADC 数据的同时采集，能够在不影响信号质量的前提下进行实时预览、处理和转换。设计目标是实现高效、低延迟的数据传输，并可将处理后的数据传输至上位机进行进一步的分析和展示。这种多信号采集卡能够显著提升信号处理效率，适用于需要实时性和可靠性的各类场景。

1.2 应用领域

该采集卡适用于广播、信息安全、数据分析等多个领域。在广播领域，采集卡可以采集并传输 HDMI 视频信号，用于实时直播和画面控制；在信息安全领域，通过网络信号采集和分析来监控流量特征，检测异常行为；在数据分析领域，采集卡用于获取 ADC 数据信号，以便对模拟数据进行数字化和深入分析。此外，该采集卡也适用于科学研究和实验室设备中，用于高精度数据记录和信号监测。该设计在多个行业中具有广泛的应用价值。

1.3 主要技术特点

本次设计的采集卡主要技术特点如下：

1.通过 AXI 仲裁和状态机实现 4 路视频的预览、任意两路视频的拼接和任意 1 路视频的显示。

2.结合状态机针对单个视频进行亮度调整、对比度调整、饱和度调整和浮雕特效。

3.采用 RS232 串口接收串口帧中的有效数据作为亮度调整、对比度调整、饱和度调整和浮雕特效算法的调整值，从而达到效果改善的目的。

4.将 RGB888 转化为 RGB565，同时实现 16 位宽的 UDP 传输，ADC 和网络信号仍采用 8 位宽进行 UDP 传输。

5.对于 2 路 ADC 则采用 256 位的 FFT 变换，同时采用双 RAM 进行时频域数据分别存储和 HDMI 显示。

1.4 关键性能指标

1. 设计可运行的最高频率（Fmax）

	Clock	Fmax	Requested Frequency	Slack
1	sys_clk	142.227MHz	50.000MHz	12.969
2	eth_rgmii_rxc_0	160.411MHz	125.000MHz	1.766
3	pixclk_in	151.768MHz	148.500MHz	0.145
4	ddr_ip_clk	119.918MHz	100.000MHz	1.661
5	ioclk0	1237.624MHz	400.000MHz	1.692
6	ioclk1	1237.624MHz	400.000MHz	1.692
7	sys_clk|adc_dac_inst/u_p…_pll_e3/CLKOUT1_Inferred	137.193MHz	35.001MHz	21.282
8	top5|eth_rgmii_rxc_1	117.536MHz	1.000MHz	991.492
9	sys_clk|u_pll/u_pll_e3/CLKOUT0_Inferred	184.026MHz	10.000MHz	94.566
10	sys_clk|u_pll/u_pll_e3/CLKOUT1_Inferred	120.438MHz	25.000MHz	31.697

2．资源利用率

Module Inst Name	LUT	FF	Distributed RAM	APM	DRM	IO	LUT CARRY	USCM
⊞ top5	14249	8692	2251	14.5	45	194	2989	4

	Logic Utilization	Used	Available	Utilization(%)
1	APM	14.5	84	18
2	IOCKDLY	2	40	5
3	FF	8270	64200	13
4	LUT	13596	42800	32
5	Distributed RAM	2251	17000	14
6	DLL	3	10	30
7	DQSL	8	18	45
8	DRM	43	134	33
9	FUSECODE	0	1	0
10	IO	186	296	63
11	IOCKDIV	1	20	5
12	IOCKGATE	4	20	20
13	IPAL	0	1	0
14	PLL	4	5	80
15	RCKB	0	24	0
16	SCANCHAIN	0	2	0
17	START	0	1	0
18	USCM	9	30	30
19	HSST	0	1	0
20	OSC	0	1	0
21	CRYSTAL	0	2	0
22	RESCAL	0	4	0
23	UDID	0	1	0
24	PCIE	0	1	0

3.功耗

```
***********
* Summary *
***********
```

Power Summary					
Total On Chip Power	Static Power	External Power	Junction Temperature	Thermal Margin	Power Margin
1.6493	0.364472	0	52.5283	47.4717	2.84262

1.5 主要创新点

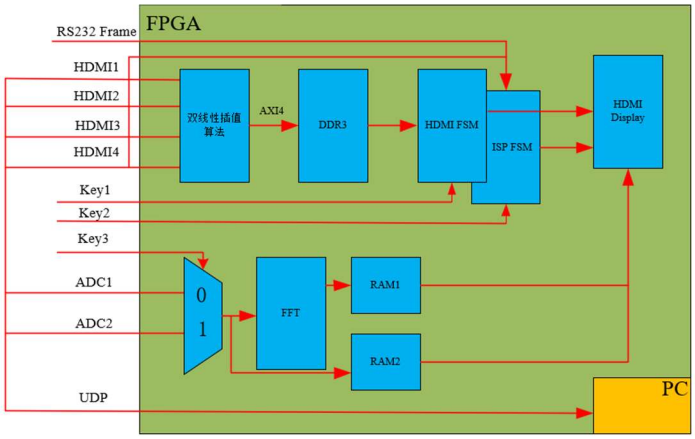
1.通过 AXI4 仲裁和状态机实现 4 路视频的预览、任意两路视频的拼接和任意 1 路视频的显示。

2.采用 RS232 串口接收串口帧中的有效数据作为亮度调整、对比度调整、饱和度调整和浮雕特效算法的调整值，从而达到效果改善的目的。

3.对于 2 路 ADC 则采用 256 位的 FFT 变换，同时采用双 RAM 进行时频域数据分别存储和 HDMI 显示。

第二部分 系统组成及功能说明

2.1 整体介绍



本次设计的模块流程图如上图所示，采集的 4 路 1920*1080@60 的 HDMI 信号通过双线性插值算法缩小为 960*540@60，然后通过 AXI4 仲裁方式写入 DDR3 中，各路视频的预览显示、单个视频显示还是两路视频的拼接显示则通过 HDMI FSM 状态机（该模块在 top5.v 的主函数部分）进行控制，控制信号扫描按键 key1（实际代码为 key_flag1[5]）进行产生。对于单路视频的亮度调整、对比度调整、饱和度调整和浮雕特效的转换则通过 ISP FSM（该模块在 top5.v 的 hdmi_loop_inst 部分）状态机进行控制，控制信号扫描按键 key2(实际代码为 key_flag1[6])进行产生。采集的 ADC 信号经过 256 位的 FFT 变换后和原 ADC 信号分别存储在 RAM1 和 RAM2 中，最后同时通过 HDMI 进行显示。HDMI 信号、ADC 信号和网络信号（UDP 信号）通过多路选择的方式将数据进行存储在 FIFO 中，当存储到一定个数后，通过 UDP 上传到上位机做进一步处理。对于 ISP 中涉及到的算法的补偿参数则通过接收上位机的串口帧的有效数据来实现。

2.2 各模块介绍

1. 双线性插值算法、亮度调整、对比度调整、饱和度调整和浮雕特效算法：

1) 双线性插值算法：

将输入的图像处理进行暂存，然后对每一行数据进行分别插值后保存在 RAM 中，完成两行的线性插值后，读出存入 ram 的第一次插值的数值，进行第二次线性插帧。

2) 亮度调整、对比度调整：

亮度和对比度的调整采用线性算法 $rgb\_out = rgb\_in * alpha + beta$ 实现，其中 alpha 控制对比度，beta 控制亮度。详细的代码实现放在附录。

3) 饱和度调整：

饱和度的调整基于原像素值和灰度图的加权组合。

```
//RGB颜色空间，饱和度调节
//原图和灰度图进行加权组合即可改变图像的饱和度
//Y = 0.2989*R + 0.5870*G + 0.1140*B;
//R_new = -Y * value + R * (1+value);
//G_new = -Y * value + G * (1+value);
//B_new = -Y * value + B * (1+value);
//value 范围 (-1~1)
```

4) 浮雕特效算法：

浮雕特效算法基于遍历整幅灰度图像，每个点的像素值使用相邻像素值之差来替代，以获得图像的边缘特征，再加上固定数值，这样就能得到浮雕效果，具体的公式如下：

$$dst(x, y) = src(x, y) - src(x+1, y) + TH$$

说明：dst是目标图像，src是源图像，TH是固定常数的阈值。

2. AXI 仲裁控制：

基于 AXI 的仲裁控制主要原理是在每个视频流的写通道 WLAST 结束之后，开始下一个通道的写入。主要的代码实现原理见附录。

3. HDMI FSM：

对于 HDMI FSM 实现的控制主要通过检测按键信号进行状态的跳变。主要的状态流程如下：从 0-1-2-3-4-5-6-7-8-9-10-0…按顺序跳转。发生跳转的条件是检测到高电平的按键信号。

```
//procedure: mode_cnt
    //------0->3 : sigle  video
    //-------  4 : four   video
    //---------5 : two videos -->> A and B
    //---------6 : two videos -->> A and C
    //---------7 : two videos -->> A and D
    //---------8 : two videos -->> B and C
    //---------9 : two videos -->> B and D
    //--------10 : two videos -->> C and D
```

4. ISP FSM：

对于四种图像处理算法的状态机实现原理同 HDMI FSM。同样发生跳转的条件是检测到高电平的按键信号。

5. ADC+FFT + HDMI：

采用的 FFT IP 核由比赛官方提供，考虑到之后的 HDMI 同时展示频谱图和时域图，所以采用 256 点 FFT 进行变换。将输入的音频数据进行 FIFO 缓存应对跨时钟，由于提供的 FFT IP 核仅支持数据的连续输入，所以采用 FFT CTRL 模块控制输入到 FFT_IP 的数据个数。数据经过 FFT 变换之后，进行取模获得频域幅度 。取模采用 JPL 和牛顿迭代的 Verilog 实现,具体见附录。

6. UDP 传输：

本次采用的 UDP FPGA 接收和发送设计基于正点原子的代码和紫光提供的 UDP 例程，部分稍作修改。对于发送端，其输入的数据首先通过存入 eth_adnet_pkt 和 eth_img_pkt 模块中的 FIFO 进行缓存，然后当 FIFO 中缓存的数据个数达到以太网要发送的字节数时，开始发送。最终通过 UDP 传输的是 8bit 的 ADC 和网络数据、32bit 的图像数据。最终 PC 通过 SOCKET 套接字进行接收。

7. RS232 接收串口帧：

由于亮度调整、对比度调整、饱和度调整和浮雕特效算法中涉及到调整效果的控制，所以在此基于串口帧接口上位机发送的串口数据，作为各个算法的调整值。串口帧的格式如下：

```
//-------------------------------------------------------------------------------------------------------------//
//The construction of uart frame:|  start  |  data type | data1 | data2  |        data_sum_check[7:0]          |    end  |//
//-------------------------------|  'h0a   |  (1,2,3,4) | 8B    |   8B   |  'h05 + (1,2,3,4) + data1 + data2   |   'hfa  |//
//-------------------------------------------------------------------------------------------------------------//
```

8’h0a 作为帧头，data type 分别对应四种算法，算法的调整值 = data1 + data2，然后进行数据校验，将传输的帧头、data type、data1、data2 进行相加，取低八位与此时接收到的串口数据进行比对，正确则进入帧尾 8’hfa 的校验。帧尾校验成功后输出成功的标志信号，当检测到该标志信号，则对调整值进行替换（初始调整值默认为 0）。

第三部分　完成情况及性能参数

基础要求（共 30 分）

1、 实现 1 路网络、1 路 HDMI 视频信号、1 路 ADC 信号到以太网的采集（20 分）；完成
2、 能进行不同采集功能的切换（10 分）。完成

加分项（共 70 分）

1、 能进行多块板卡进行级联，同时进行 4 路以上网络信号的采集和分析，并能尽可能降低采集延迟（15 分）；
2、 能进行 4 路以上 HDMI 信号采集及预览，并能对任意一个视频进行单独显示，对任意 2 个视频进行拼接显示，要求能实时切换，且切换过程不黑屏（15 分）；完成
3、 能进行 2 路以上的 ADC 信号采集，并对 ADC 信号进行频域和时域变换存储及显示（5 分）；完成
4、 能将数据通过 PCIE 或者光口网卡进行采集（如使用光口，PC 需要光纤网卡），并尽可能增大带宽、降低传输延迟（10 分）；
5、 其他功能：能支持 4 种以上的视频效果展示，如字符叠加、亮度调整等；（每种效果 5 分，共计 20 分）；完成
6、 其他采集功能（5 分）。UART采集数据完成

第四部分 总结

4.1 可扩展之处

1. 拓展四路视频的不同效果的分开实现，可以考虑采用四路 UART 采集和 ISP FSM 的控制并行实现，同时需要优化面积。
2. 目前整体功能还是针对于单板，需要拓展 SFP、PCIE 来实现拓扑接口，来达到大带宽和低延迟的传输。
3. 拓展四路网络信号的采集和分析。

4.2 心得体会

通过此次多路多功能采集卡设计竞赛，我们在技术实现、设计思路和系统架构方面获得了宝贵的实践经验，也深刻理解了 FPGA 在多信号处理中的重要性。整个设计过程中，我们不仅提高了自己在硬件电路和数字信号处理方面的知识，还对数据采集和实时信号处理有了新的认识和理解。

此次竞赛让我们认识到，设计一个系统并不仅仅是满足功能需求，还需要综合考虑系统的可扩展性、兼容性和可靠性。为了实现多信号采集，我们需要确保系统的架构足够灵活，以便在将来根据不同需求进行扩展。因此，在设计中我们采取了模块化的思维，将不同的信号采集通道独立设计，使系统更具可扩展性。此外，选择 FPGA 作为核心处理模块也增加了系统的灵活性，可以通过编程对不同信号进行处理和切换。系统设计不仅仅是技术实现，更是一个不断权衡和优化的过程。

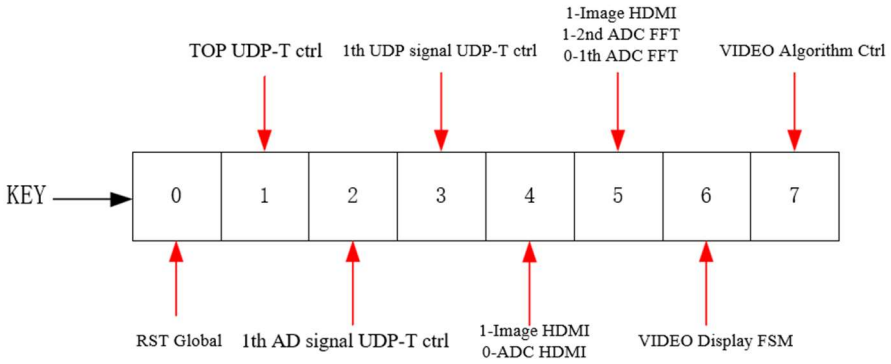
第五部分 参考文献

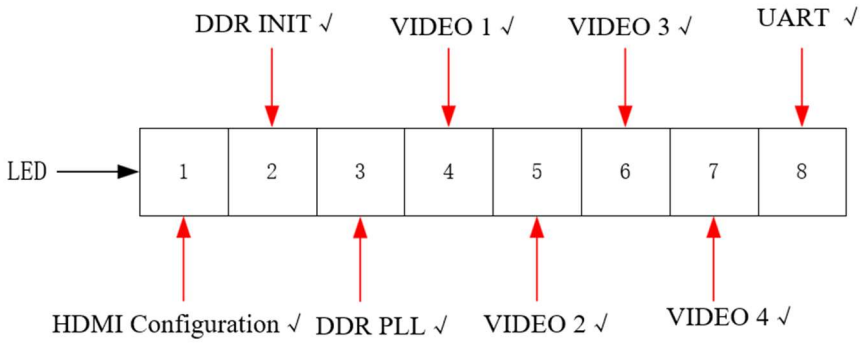
1. 正点原子的例程
2. 紫光的例程

第六部分 附录

以下只是列出了功能部分代码的思想，全部的代码可以在 GitHub 找到，这里给出地址：https://github.com/misslin516/FPGA-competition-2024

项目的按键和 led 显示灯的对应和功能如下：





1. 亮度调整、对比度调整的代码实现：

亮度调整：

```
assign {R, G, B} = img_data_i;
always@(posedge clk or posedge reset) begin
    if(reset) begin
        valid_d1 <= 0;
        {R_add_a, G_add_a, B_add_a} <= 0;
    end else begin
        valid_d1 <= valid_i;
        R_add_a <= R + {adjust_val[8], adjust_val};
        G_add_a <= G + {adjust_val[8], adjust_val};
        B_add_a <= B + {adjust_val[8], adjust_val};
    end
end


always@(posedge clk or posedge reset) begin
    if(reset) begin
        valid_d2 <= 0;
        {R_new, G_new, B_new} <= 0;
    end else begin
        valid_d2 <= valid_d1;
        R_new <= R_add_a[9] ? 0 : R_add_a[8] ? 255 : R_add_a[8:0];
        G_new <= G_add_a[9] ? 0 : G_add_a[8] ? 255 : G_add_a[8:0];
        B_new <= B_add_a[9] ? 0 : B_add_a[8] ? 255 : B_add_a[8:0];
    end
end
```

对比度调整：

```
assign {R, G, B} = img_data_i;
always@(posedge clk or posedge reset) begin
    if(reset) begin
        valid_d1 <= 0;
        {R_m_a, G_m_a, B_m_a} <= 0;
    end else begin
        valid_d1 <= valid_i;
        R_m_a <= R*adjust_val;
        G_m_a <= G*adjust_val;
        B_m_a <= B*adjust_val;
    end
end


always@(posedge clk or posedge reset) begin
    if(reset) begin
        valid_d2 <= 0;
        {R_new, G_new, B_new} <= 0;
    end else begin
        valid_d2 <= valid_d1;
        R_new <= R_m_a[16] ? 255 : R_m_a[15:8];
        G_new <= G_m_a[16] ? 255 : G_m_a[15:8];
        B_new <= B_m_a[16] ? 255 : B_m_a[15:8];
    end
end
```

2. AXI 仲裁的 WLAST 实现控制部分：

```
    else if(DDR_INIT_DONE)begin
        case(arbitration_wr_state)
            M0_WRITE:
            begin
                if(M0_AXI_WLAST) begin//第一次传输完成，对第二个主机进行传输
                    arbitration_wr_state <= M1_WRITE;
                end
                else if( (!(wfifo0_state == 'd3)) && (wfifo0_rd_water_level < M_AXI_BRUST_LEN) && (wr_addr_cnt0 < wr_addr_max - M_AXI_BRUST_LEN * 8 )) begin
                    arbitration_wr_state <= M1_WRITE;
                end
            end
            M1_WRITE:
            begin
                if(M1_AXI_WLAST) begin
                    arbitration_wr_state <= M2_WRITE;
                end
                else if( (!(wfifo1_state == 'd3)) && (wfifo1_rd_water_level < M_AXI_BRUST_LEN)&& (wr_addr_cnt1 < wr_addr_max - M_AXI_BRUST_LEN * 8 )) begin
                    arbitration_wr_state <= M2_WRITE;
                end
            end
            M2_WRITE:
            begin
                if(M2_AXI_WLAST) begin
                    arbitration_wr_state <= M3_WRITE;
                end
                else if((!(wfifo2_state == 'd3)) && (wfifo2_rd_water_level < M_AXI_BRUST_LEN)&& (wr_addr_cnt2 < wr_addr_max - M_AXI_BRUST_LEN * 8 )) begin
                    arbitration_wr_state <= M3_WRITE;
                end
            end
            M3_WRITE:
            begin
                if(M3_AXI_WLAST) begin
                    arbitration_wr_state <= M0_WRITE;
                end
                else if((!(wfifo3_state == 'd3)) && (wfifo3_rd_water_level < M_AXI_BRUST_LEN)&& (wr_addr_cnt3 < wr_addr_max - M_AXI_BRUST_LEN * 8 )) begin
                    arbitration_wr_state <= M0_WRITE;
                end
            end
        endcase
    end
    else begin
        arbitration_wr_state <= arbitration_wr_state;
    end
end
```

3. JPL 和牛顿迭代的 Verilog 实现

   JPL 开根号：

```
always@(posedge clk) begin
    if(syn_rst) begin
       dataa_reg <= 'd0 ;
       datab_reg <= 'd0 ;
    end else begin
        if(valid_in) begin
            dataa_reg <= dataa;
            datab_reg <= datab;
        end
    end
end


//check the number is positive or not,if negative ,trans it into positive
assign dataa_abs = (dataa_reg[31] == 1'b1) ? (31'd0-dataa_reg[N-2:0]) : dataa_reg[N-2:0] ;
assign datab_abs = (datab_reg[31] == 1'b1) ? (31'd0-datab_reg[N-2:0]) : datab_reg[N-2:0] ;

always @(posedge clk)
   begin
        if(dataa_abs > datab_abs)
             begin
                  dataabs_max <= dataa_abs ;
                  dataabs_min <= datab_abs ;
                  absmin_3 <= {1'b0,datab_abs}+{datab_abs,1'b0} ;
             end
        else
          begin
                  dataabs_max <= datab_abs ;
                  dataabs_min <= dataa_abs ;
                  absmin_3 <= {1'b0,dataa_abs}+{dataa_abs,1'b0} ;
             end
   end

 always @(posedge clk)
   begin
        if(absmin_3 > {1'b0,dataabs_max})
             ampout <= {1'b0,dataabs_max} - {4'b0,dataabs_max[N-2:3]} + {2'b0,dataabs_min[N-2:1]} ;
        else
          ampout <= {1'b0,dataabs_max} + {4'b0,dataabs_min[N-2:3]} ;
   end
```

牛顿迭代开根号：

```
//------------------------------------------------------------------------------

//        迭代计算过程

//------------------------------------------------------------------------------
        generate
            genvar i;
                for(i=r_width-1;i>=1;i=i-1)
                    begin:U
                        always@(posedge clk or posedge    rst)
                            begin
                                if(rst)
                                    begin
                                        D[i] <= 0;
                                        Q_z[i] <= 0;
                                        Q_q[i] <= 0;
                                        ivalid_t[i] <= 0;
                                    end
                                else    if(ivalid_t[i+1])
                                    begin
                                        if(Q_z[i+1]*Q_z[i+1] > D[i+1])
                                            begin
                                                Q_z[i] <= {Q_q[i+1][q_width:i],1'b1,{{i-1}{1'b0}}};
                                                Q_q[i] <= Q_q[i+1];
                                            end
                                        else
                                            begin
                                                Q_z[i] <= {Q_z[i+1][q_width:i],1'b1,{{i-1}{1'b0}}};
                                                Q_q[i] <= Q_z[i+1];
                                            end
                                        D[i] <= D[i+1];
                                        ivalid_t[i] <= 1;
                                    end
                                else
                                    begin
                                        ivalid_t[i] <= 0;
                                        D[i] <= 0;
                                        Q_q[i] <= 0;
                                        Q_z[i] <= 0;
                                    end
                                end
                    end
        endgenerate
          -
//------------------------------------------------------------------------------

//    计算余数与最终平方根

//------------------------------------------------------------------------------
        always@(posedge    clk or posedge    rst)
            begin
                if(rst)
                    begin
                        data_o <= 0;
                        data_r <= 0;
                        o_vaild <= 0;
                    end
                else    if(ivalid_t[1])
                    begin
                        if(Q_z[1]*Q_z[1] > D[1])
                            begin
                                data_o <= Q_q[1];
                                data_r <= D[1] - Q_q[1]*Q_q[1];
                                o_vaild <= 1;
                            end
                        else
                            begin
                                data_o <= {Q_q[1][q_width:1],Q_z[1][0]};
                                data_r <= D[1] - {Q_q[1][q_width:1],Q_z[1][0]}*{Q_q[1][q_width:1],Q_z[1][0]};
                                o_vaild <= 1;
                            end
                    end
                else
                    begin
                        data_o <= 0;
                        data_r <= 0;
                        o_vaild <= 0;
                    end
            end
//------------------------------------------------------------------------------
```