

基于 LVDS 接口 10.1 寸 IPS 液晶屏显示彩色块实验

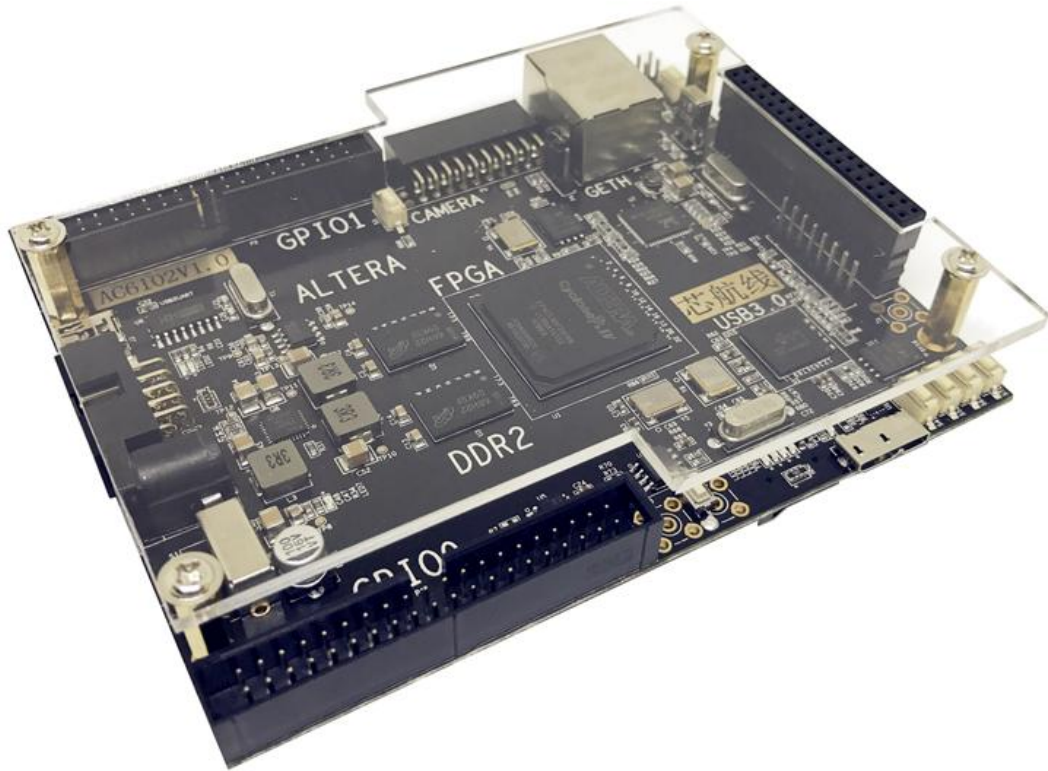
- 一、基于 LVDS 接口 10.1 寸 IPS 液晶屏显示彩色块实验..... 2
 - 1.1 准备材料 2
 - 1.2 显示效果图..... 3
 - 1.3 功能实现 3
 - 1.3.1 系统概述..... 3
 - 1.3.2 LVDS 编码..... 4
 - 1.3.3 ALTLVDS_TX 配置..... 6
 - 1.3.4 ALTLVDS_TX 接口说明..... 7
 - 1.3.5 10.1 寸 IPS 的液晶屏简介..... 8
 - 1.3.6 系统框图及板级验证 9

一、基于 LVDS 接口 10.1 寸 IPS 液晶屏显示彩色块实验

注意：本实例对应工程文件名称为 lvds_lcd.rar

1.1 准备材料

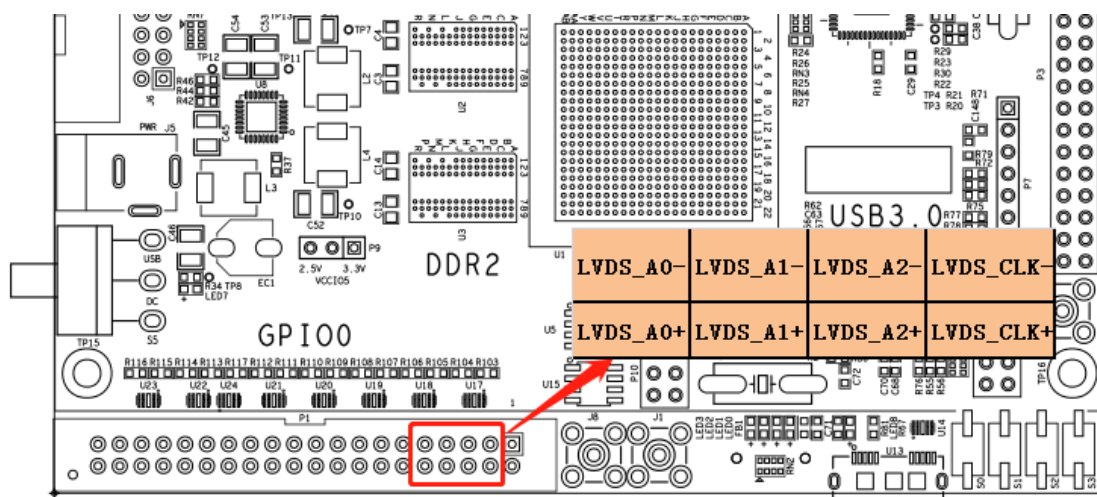
1. 6102 开发板一套



2. LVDS 接口 10.1 寸 IPS 液晶屏一个



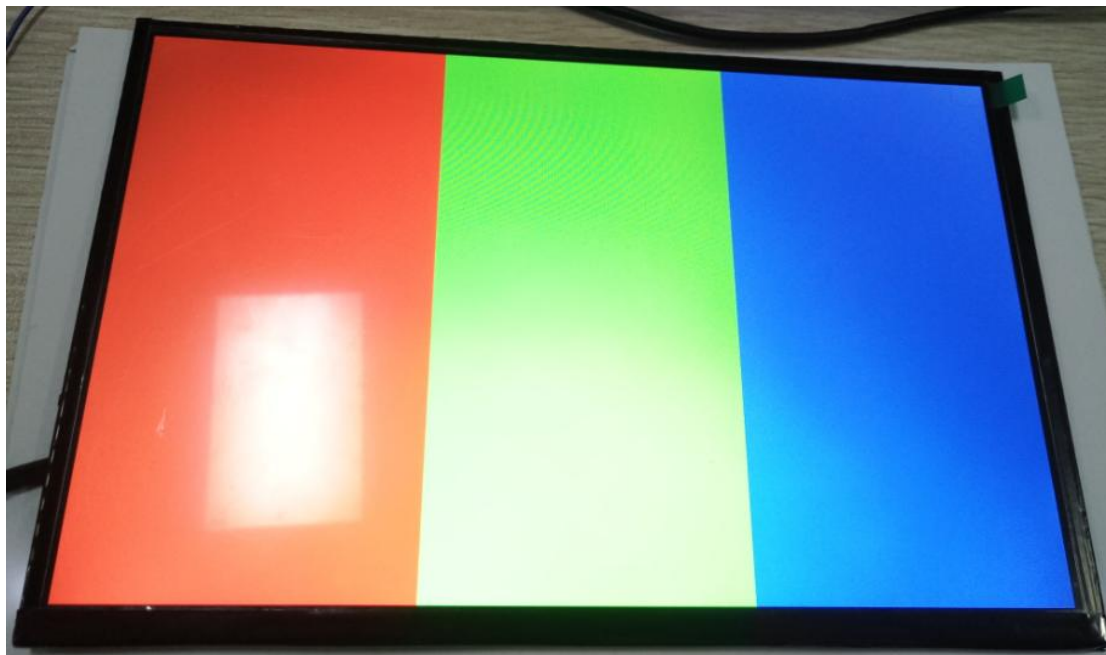
3. 硬件连接示意图



实际接线顺序

LVDS_A0- (白色)	LVDS_A1- (蓝色)	LVDS_A2- (白色)	LVDS_CLK- (蓝色)
LVDS_A0+ (蓝色)	LVDS_A1+ (白色)	LVDS_A2+ (蓝色)	LVDS_CLK+ (白色)

1.2 显示效果图



1.3 功能实现

1.3.1 系统概述

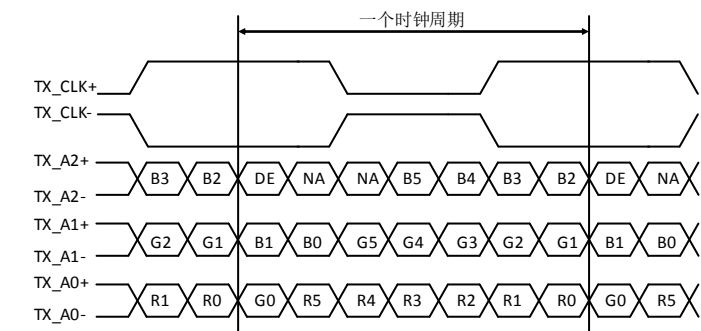
整个系统就是先通过一个锁相环产生一个 83.5MHz 的工作时钟供 vga_ctrl 模块和 lcdlvs 模块使用，vga_ctrl 模块是用来产生 vga（1280x800 的像素）时序并显示红绿蓝三个彩条，lcdlvs 模块里面调用了 altlvs tx 的 IP 核，用来将产生的 vga 时序信号按照 lvs 的编码

方式发送到 10.1 寸 IPS 液晶屏上。

1.3.2 LVDS 编码

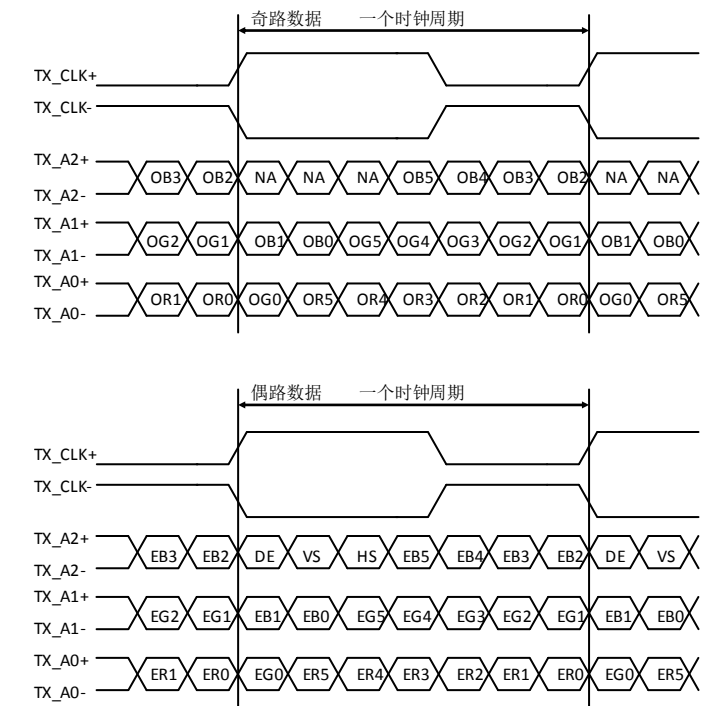
- 1) LVDS 编码标准有两种，一种 JEIDA 的标准，一种是 VESA 的标准。
- 2) LVDS 数据格式有单通道 6bit 数据格式、双通道 6bit 数据格式、单通道 8bit 数据格式 A、单通道 8bit 数据格式 B、双通道 8bit 数据格式

1.3.2.1 单通道 6bit 数据格式



图中 NA 的意思是未使用。此例为控制信号仅使用 DE 的模式，未使用行同步信号 HS 和场同步信号 VS。当控制信号为 DE+行场同步信号模式时，图中的两个 NA 更换为场同步信号 VS 和行同步信号 HS。

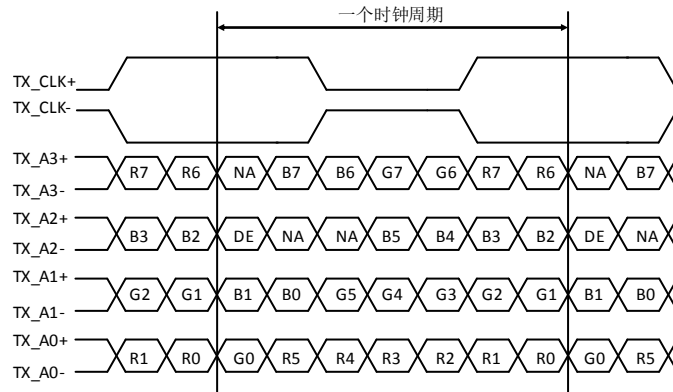
1.3.2.2 双通道 6bit 数据格式



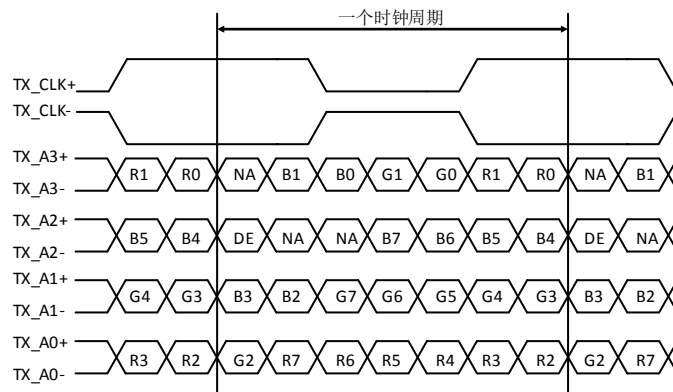
双通道 6bit LVDS 发送电路使用两片四通道 LVDS 发送芯片,从图中可以看出，双路 6bit

LVDS 发送芯片数据输出格式与单路 6bit LVDS 发送芯片数据输出格式是相同的，只不过一路传送奇数像素 RGB 数据，另一路传送偶数像素 RGB 数据。OR0、OR1、…中的“O”代表奇数像素，ER0、ER1、…中的“E”代表偶数像素。

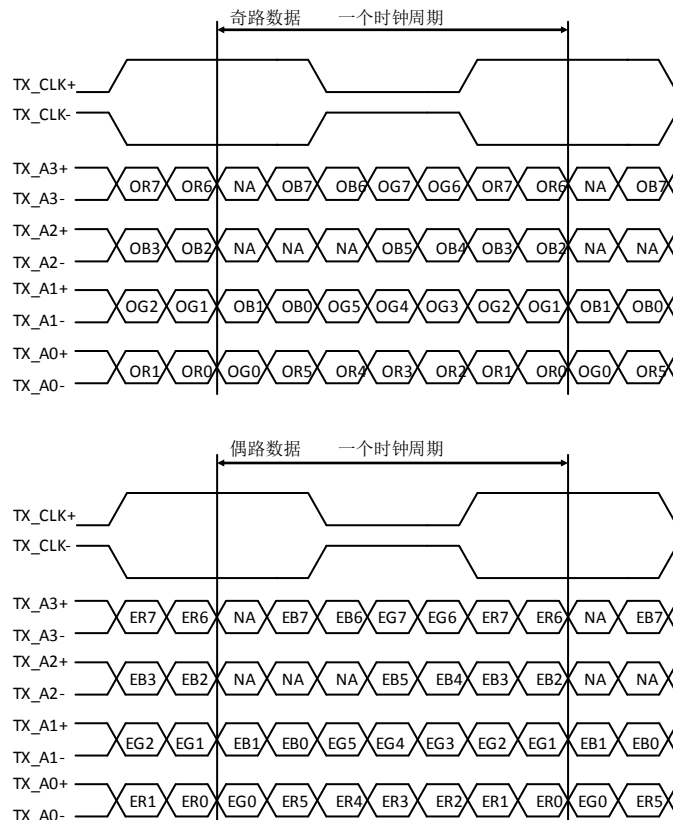
1.3.2.3 单通道 8bit 数据格式 A



1.3.2.4 单通道 8bit 数据格式 B

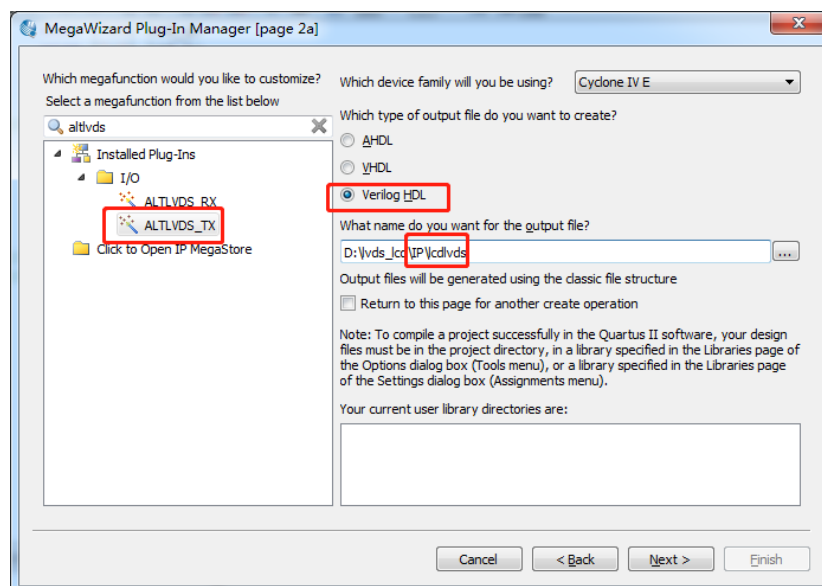


1.3.2.5 双通道 8bit 数据格式



1.3.3 ALTLVDS_TX 配置

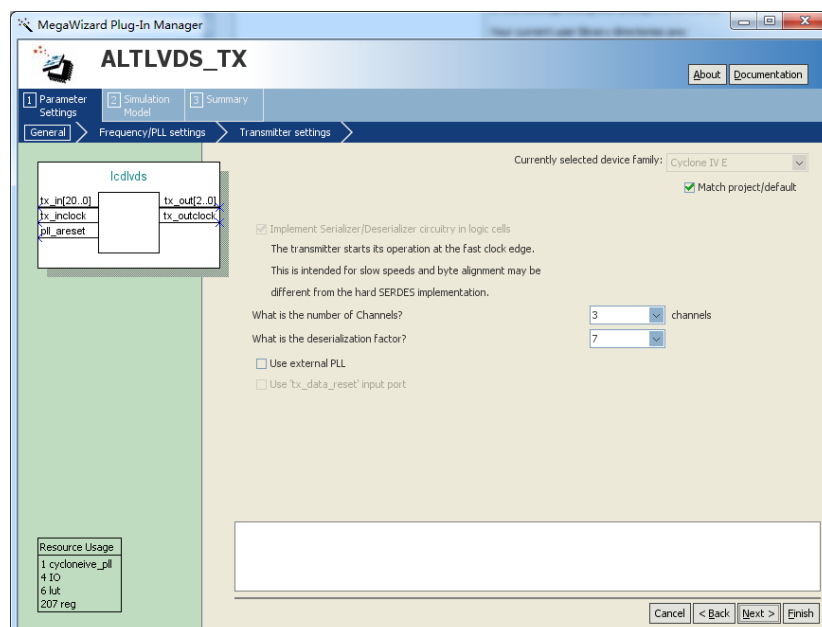
在 IP 核向导搜索框内输入 altlvds



在 “What is the number of Channels?” 后面的下拉栏中选择 LVDS 的通道数为 3。

在 “What is the deserialization factor?” 后面的下拉栏中选择串化因子为 “7”。

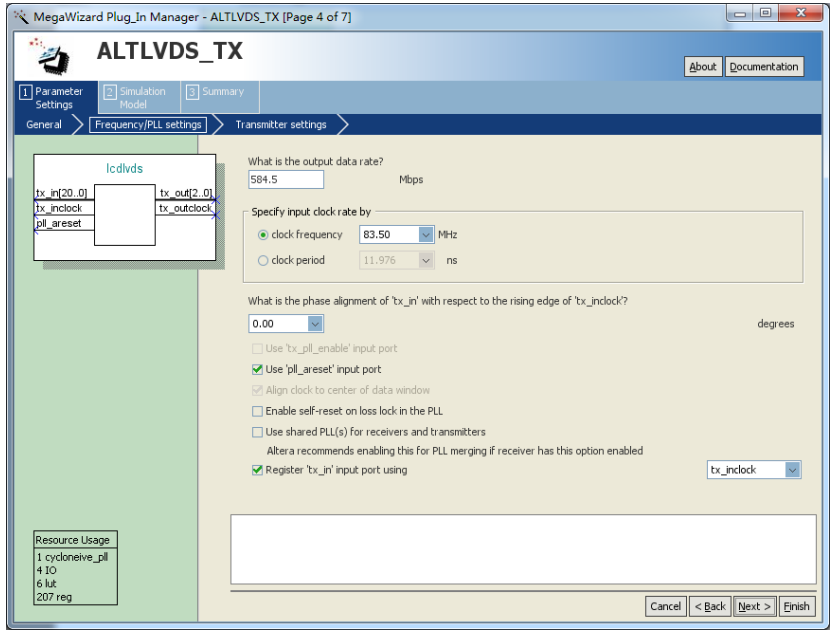
如图所示：



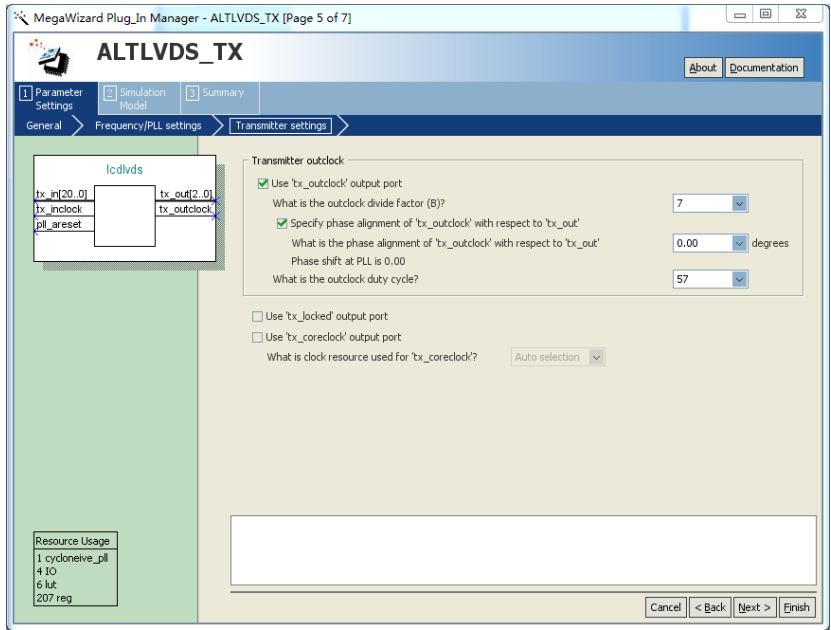
点击 NEXT 后进入下一步，配置频率，按照如下来配置：

在 “What is the output data rate?” 下面输入单通道的 LVDS 数据速率为 “584.5” Mbps。

在“Specify input clock rate by” 下面的“Clock frequency”后面输入 LVDS 时钟频率为“83.50” MHz。因为在这里我们 VGA 用的是 1280x800 的分辨率，所以时钟位 83.50MHz
LVDS 数据速率 = LVDS 时钟频率 * 串化因子，即 584.5Mbps = 7 * 83.50MHz。



点击 NEXT 后进入下一步，配置发送相关的设置，按照如下配置即可：



具体的详细用法可以点击界面的右上角的 Documentation 里面的 IP 核手册来参考。

1.3.4 ALTLVDS_TX 接口说明

```

lcdlvds lcdlvds_inst (
    .pll_areset ( !g_rst_n          ),
    .tx_in      ( lvds_adjt         ),
    .tx_inclock ( clk_83m           ),
    .tx_out     ( lvds_data         ),
    .tx_outclock( lvds_clk          )
);

```

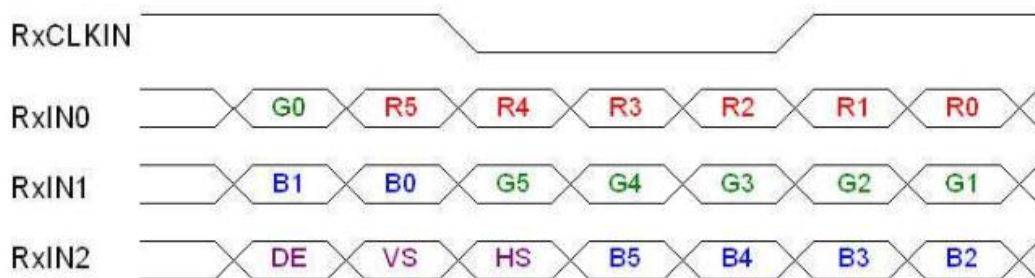
信号名称	方向	功能
pll_areset	input	复位信号，高电平有效
tx_in	input	LVDS 并行发送数据输入
tx_inclock	input	LVDS 并行发送数据的同步时钟输入
tx_out	output	LVDS 串行发送数据输出
tx_outclock	output	LVDS 串行发送数据的同步时钟输出

因为 altlvds_tx 的 IP 核是从时钟上升沿开始发送，这里的传化因子位 7，所以高电平期间传输 4bit 数据，低电平期间传输 3bit 数据，总共三个数据通道，所以 tx_in[6:0]对应通道 tx_out[0]，tx_in[13:7]对应通道 tx_out[1]，tx_in[20:14]对应通道 tx_out[2]。

1.3.5 10.1 寸 IPS 的液晶屏简介

在这里我们用的 10.1 寸 IPS 的液晶屏是单通道 6bit 数据格式，液晶屏的数据输入格式如图所示：

6.2 The Input Data Format



信号名称	描述
R5-R0	6bit 红色数据，R5(MSB)
G5-G0	6bit 绿色数据，G5(MSB)
B5-B0	6bit 蓝色数据，B5(MSB)
RxCLKIN	数据时钟输入
DE	Data Enable(DE),数据有效
VS	场同步
HS	行同步

所以根据 ALTLVDS_TX 这个 IP 核的输入，所以我们要把 21bit 的数据串化起来送几区，就是这样的格式

DE	VS	HS	B5	B4	B3	B2	B1	B0	G5	G4	G3	G2	G1	G0	R5	R4	R3	R2	R1	R0
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

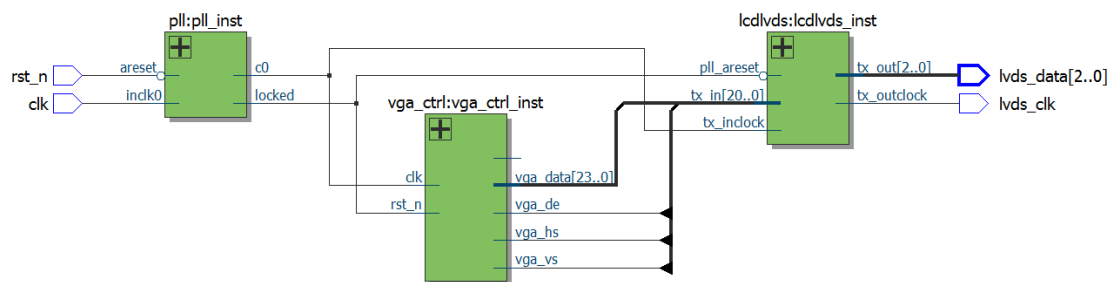
所以代码中按照这样的方式将数据拼接起来

```
wire[20:0]lvds_txin= {lcd_de,lcd_vs,lcd_hs,lcd_b[7:2],lcd_g[7:2],lcd_r[7:2]};
```

根据这个液晶屏的数据输入时序图(左边两个数据是在时钟高电平,中间三个数据是低电平,右边两个数据又是高电平),而 altlvds_tx 的 IP 核是从时钟上升沿开始发送,所以这拼接的数据还需要调整,所以我们把右边的 2bit 数据要拼接到左边来,所以就按照如下格式来调整才能正确的发送数据

```
wire[20:0]lvds_adjt={lvds_txin[15:14],lvds_txin[20:16],  
                    lvds_txin[8 :7 ],lvds_txin[13:9 ],  
                    lvds_txin[1 :0 ],lvds_txin[6 :2 ]};
```

1.3.6 系统框图及板级验证



系统 RTL 示意图

模块说明:

pll.v: 产生 vga_ctrl 模块和 lcdlvds 模块工作时钟

vga_ctrl.v: 产生 vga 时序和彩色块

lcdlvds.v: 将产生的 vga 时序信号按照 lvds 的编码方式发送到 10.1 寸 IPS 液晶屏上

lvds_lcd.v: 顶层模块

vga_ctrl 模块代码:

```
module vga_ctrl(  
    clk,  
    rst_n,  
    vga_clk,  
    vga_vs,  
    vga_hs,  
    vga_de,  
    vga_data  
);  
    input clk, rst_n;  
    output vga_clk, vga_de;
```

```

output vga_vs, vga_hs;
output reg [23:0] vga_data;

//1280 x 800 @ 60Hz; Pixel Clock = 83.500 MHz(12ns)
`define HS_A      128    //H_Sync_Time
`define HS_B      200    //H_Back_Porch + H_Left_Border
`define HS_C      1280   //Hor Addr Time(real pixel)
`define HS_D      72     //H_Right_Border + H_Front_Porch
`define HS_E      1680   //H_Total_Time

`define VS_A      6      //V_Sync_Time
`define VS_B      22     //V_Back_Porch + V_Top_Border
`define VS_C      800    //Ver Addr Time(real pixel)
`define VS_D      3      //V_Bottom_Border + V_Front_Porch
`define VS_E      831    //V_Total_Time

parameter PART_L = `HS_C / 3;
parameter PART_M = `HS_C / 3;
parameter PART_R = `HS_C - PART_L - PART_M;

reg [10:0] cnt_vs; //VGA 场扫描计数器
reg [10:0] cnt_hs; //VGA 行扫描计数器
wire en_show;      //图像显示有效区

always @ (posedge clk or negedge rst_n)
begin
    if (!rst_n)
        cnt_hs <= 'd0;
    else
        if (cnt_hs < `HS_E - 1'b1)
            cnt_hs <= cnt_hs + 1'b1;
        else
            cnt_hs <= 'd0;
end

assign vga_hs = (cnt_hs > `HS_A - 1'b1);

always @ (posedge clk or negedge rst_n)
begin
    if (!rst_n)
        cnt_vs <= 'd0;
    else
        if (cnt_hs == `HS_E - 1'b1)
            if (cnt_vs < `VS_E - 1'b1)

```

```

        cnt_vs <= cnt_vs + 1'b1;
    else
        cnt_vs <= 'd0;
    else
        cnt_vs <= cnt_vs;
end

assign vga_vs = (cnt_vs > `VS_A - 1'b1);

//图像显示有效区
assign en_show = (`HS_A + `HS_B - 1'b1 < cnt_hs) && (cnt_hs < `HS_A
+ `HS_B + `HS_C - 1'b1)
&&(`VS_A + `VS_B - 1'b1 < cnt_vs) && (cnt_vs < `VS_A +
`VS_B + `VS_C - 1'b1);
assign vga_de = en_show;
assign vga_clk = clk;

always @ (posedge clk or negedge rst_n)
begin
    if (!rst_n)
        vga_data <= 24'h000000;
    else
        if (en_show)
            if ((`HS_A + `HS_B - 1'b1 < cnt_hs) && (cnt_hs < `HS_A +
`HS_B - 1'b1 + PART_L))
                vga_data <= 24'hFF0000; //红色彩条
            else
                if ((cnt_hs < `HS_A + `HS_B - 1'b1 + PART_L + PART_M))
                    vga_data <= 24'h00FF00; //绿色彩条
                else
                    vga_data <= 24'h0000FF; //蓝色彩条
            else
                vga_data <= 24'h000000;
        end
    end

endmodule

```

顶层模块例化

```

/*
 *   LVDS 数据速率 = LVDS 时钟频率 * 串化因子
 */
module lvds_lcd(
    clk,

```

```

rst_n,
lvds_clk,
lvds_data
);

input clk;
input rst_n;
output lvds_clk;
output [2:0] lvds_data;

wire clk_83m;    //83.5Mhz
wire g_rst_n;

pll pll_inst(
    .areset ( !rst_n    ),
    .inclk0 ( clk       ),
    .c0      ( clk_83m   ),
    .locked  ( g_rst_n   )
);

wire lcd_de;
wire lcd_hs;
wire lcd_vs;
wire [7:0] lcd_r;
wire [7:0] lcd_g;
wire [7:0] lcd_b;
vga_ctrl vga_ctrl_inst(
    .clk      (clk_83m          ),
    .rst_n    (g_rst_n         ),
    // .vga_clk(),
    .vga_vs   (lcd_vs          ),
    .vga_hs   (lcd_hs          ),
    .vga_de   (lcd_de          ),
    .vga_data  ({lcd_r, lcd_g, lcd_b})
);

wire[20:0]lvds_txin={lcd_de,lcd_vs,lcd_hs,lcd_b[7:2],lcd_g[7:2],lcd_r
[7:2]};

wire [20:0] lvds_adjt = {lvds_txin[15:14],lvds_txin[20:16], //4_3
                        lvds_txin[8 :7 ],lvds_txin[13:9 ],
                        lvds_txin[1 :0 ],lvds_txin[6 :2 ]};

lcdlvds lcdlvds_inst (
    .pll_areset ( !g_rst_n      ),

```

```

.tx_in      ( lvds_adjt      ),
.tx_inclock ( clk_83m        ),
.tx_out     ( lvds_data      ),
.tx_outclock( lvds_clk       )
);

```

endmodule

引脚分配:

在这里有人会有个疑问, lvds 不是差分对传输么, 为什么代码里面写的不是差分对的逻辑, 其实我们在分配引脚的时候讲 IO 电平标准设置成 LVDS, 默认的信号名称后面会多出来“(n)”, 这个就是对应的差分对的负线, 不带“(n)” 就是对应的差分对的正线, 于是我们按照下面来分配引脚即可

Named: * Edit: [X] [Y]						
Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
in clk	Input	PIN_G1	1	B1_N3	3.3-V LVTTTL	
out lvds_clk	Output	PIN_M19	5	B5_N0	LVDS	
out lvds_data[2]	Output	PIN_N19	5	B5_N1	LVDS	
out lvds_data[1]	Output	PIN_N18	5	B5_N0	LVDS	
out lvds_data[0]	Output	PIN_R19	5	B5_N1	LVDS	
out lvds_data[2](n)	Output	PIN_N20	5	B5_N1	LVDS	
out lvds_data[1](n)	Output	PIN_N17	5	B5_N0	LVDS	
out lvds_data[0](n)	Output	PIN_R18	5	B5_N1	LVDS	
in rst_n	Input	PIN_L8	1	B1_N2	3.3-V LVTTTL	
out lvds_clk(n)	Output	PIN_M20	5	B5_N0	LVDS	
<<new node>>						

板级验证效果图:

