**实验10 综合实验**

**简介**

该实验是本课程系列实验的最后一个，我们将通过本次实验学习几种通用接口，并要求读者完成一个综合实验题目，以达到对本系列实验复习巩固的目的。

**实验目的**

熟练掌握前面实验中的所有知识点

熟悉几种常用通信接口的工作原理及使用

独立完成具有一定规模的功能电路设计

**实验环境**

VLAB：vlab.ustc.edu.cn

FPGAOL: fpgaol.ustc.edu.cn（或Nexys4 DDR）

Logisim

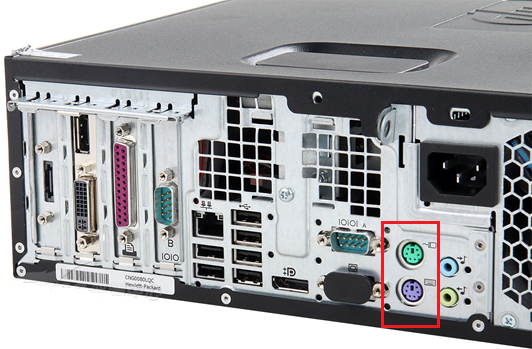
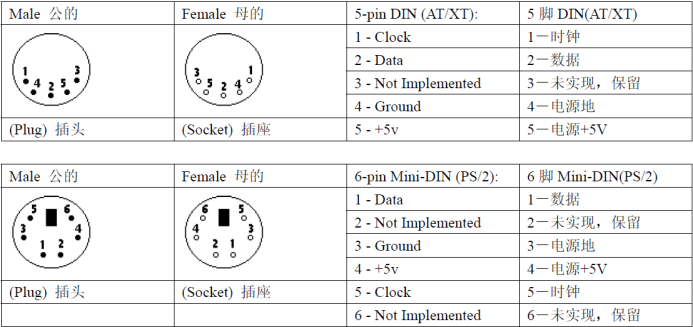
Vivado

自选外设

**实验步骤**

**Step1. PS2接口**

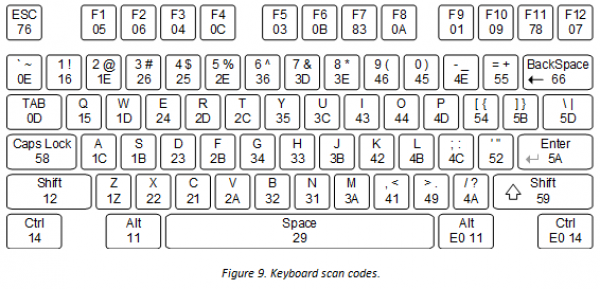
PS/2接口最初由IBM开发和使用，主要用于鼠标键盘的连接，鼠标一般为绿色接头，键盘则为紫色接头，如下图所示：

ps2接口主要用到了4根信号线（电源、地、时钟、数据），其中时钟和数据都是双向传输信号，因此能够实现数据的双向传输，当PS2设备向主机发送数据时，会首先检测时钟信号是否为高电平，然后连续发送11个时钟负脉冲，并在数据线上发送11bit的数据，包括起始位（1bit）、数据位（8bit）、校验位（1bit）和停止位（1bit），主机可在PS2\_CLK信号的下降沿对数据信号进行采样，其时序图如下所示：



此处，我们以PS2键盘为例讲解在Nexys4DDR开发板上的设计实现（由于Nexys4DDR开发板上有专门的芯片进行协议转换，因此对于部分USB接口的键盘也可以支持），键盘布局图如下所示：



在键盘按键、松开时都会向主机发送数据，例如当’a’被按下时，发送一个字节“1C”（称为通码），该按键松开时，发送两个字节“F0 1C”（称为断码）。

我们设计电路，接收PS2接口数据，并将最后接收到的16bit数据在LED上显示出来，由于PS2的时钟和数据信号相对于100MHz时钟来说变化非常缓慢，因此我们可通过前面实验中介绍的方法获取PS2接口时钟信号的下降沿，并在下降沿时刻对数据信号进行采样，其电路时序图如下所示：



前面提到过，在ps2接口发送数据时，每个数据包包含11个bit，但只有8个为有效数据，因此我们需要对接收到的数据进行计数，只取1~8位，示例代码如下所示：

module ps2\_test(

input clk,rst,ps2\_clk,ps2\_data,

output reg [15:0] led

);

reg ps2\_clk\_r1,ps2\_clk\_r2;

wire ps2\_clk\_neg;

reg [3:0] ps2\_clk\_cnt;

always@(posedge clk or posedge rst)

begin

if(rst)

ps2\_clk\_r1 <= 1'b1;

else

ps2\_clk\_r1 <= ps2\_clk;

end

always@(posedge clk or posedge rst)

begin

if(rst)

ps2\_clk\_r2 <= 1'b1;

else

ps2\_clk\_r2 <= ps2\_clk\_r1;

end

assign ps2\_clk\_neg = (ps2\_clk\_r1==1'b0)&&(ps2\_clk\_r2==1'b1);

always@(posedge clk or posedge rst)

begin

if(rst)

ps2\_clk\_cnt <= 4'd0;

else if(ps2\_clk\_neg)

begin

if(ps2\_clk\_cnt>=4'd10)

ps2\_clk\_cnt <= 4'd0;

else

ps2\_clk\_cnt <= ps2\_clk\_cnt + 4'd1;

end

end

always@(posedge clk or posedge rst)

begin

if(rst)

led <= 16'hFFFF;

else if(ps2\_clk\_neg)

begin

if((ps2\_clk\_cnt>=1)&&(ps2\_clk\_cnt<=8))

led <= { ps2\_data,led[15:1]};

end

end

endmodule

**Step2. 串口**

从广义上来说，采用串行接口进行数据通信的接口都可以称为串口，如SPI接口、IIC接口等，但我们所说的串口一般是指通用异步收发器（Universal Asynchronous Receiver/Transmitter），简称UART，主要包含RX、TX、GND三个接口信号，其中GND为共地信号，TX、RX负责数据的发送和接收。在嵌入式系统开发中，串口是一种必备的通信接口，在系统开发测试阶段和实际工作阶段都起着非常重要的作用。在Nexys4DDR开发板中，UART通信与USB烧写功能集成在了一个microUSB接口中，如下图所示。



用户将Nexys4DDR与PC相连，并上电之后，便可以在PC端的设备管理器中发现对应的串行接口。

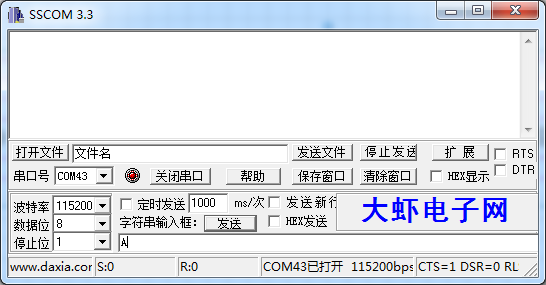


由于串行接口没有时钟信号，因此需要在收发两侧约定好一个特定的数据收发频率和数据格式。串口协议中支持的数据收发频率（又称波特率，bps）有多种，如9600、19200、115200、256000等，以115200为例，表示1s钟可以传送115200位的数据，本实验中，我们约定使用115200的波特率进行讲解和设计。串口的收发信号采用相同的数据格式，如下数据帧所示，当没有数据需要发送时，可以发送空闲帧，如下图空闲帧所示。



每一数据帧都包含“起始位+数据位+停止位”，两帧之间可以插入始终为高电平信号的空闲帧，根据协议，数据帧起始位为低电平、停止位为高电平，数据位长度可选择5~8中的任意数字，在数据位和停止位之间还可以包含奇偶校验位。为简单起见，我们本实验中选择“1位起始位+8位数据位+1位停止位”的数据帧结构，不使用奇偶校验功能。

首先，我们可以通过一个简单的环回测试来了解串口的使用。在FPGA内，将UART\_TX（C4引脚）输入到FPGA内的信号直接赋值给UART\_RX（D4引脚），在上位机看来即实时接收刚刚发送出去的数据。在上位机，我们可以使用任意一种串口工具进行调试，如超级终端、putty、SecureCRT等。如下图所示，需要特别注意串口号（与设备管理器中一致）、波特率（115200）、数据位（8）、停止位（1）等参数的设定。



其代码为：

module uart\_test1(

input uart\_tx,

output uart\_rx);

assign uart\_rx = uart\_tx;

endmodule

在发送窗口输入数据并发送，便可以在接收窗口收到同样的内容。

第二步，我们可以实现一个简单的数据接收模块，将UART\_TX发来的数据进行串并转换，并输出到LED灯上。模块工作流程可通过以下时序图来解释说明。



板载主时钟为100MHz，因此串行数据波特率为115200时，每个位持续约868个周期，我们用分频计数器进行计数，当接收信号为0时（起始位），分频计数器开始计数，计数值达到433时（起始位中间时刻），状态机从空闲状态跳转到接收状态，分频计数器在0~867循环计数，同时用位计数器进行位计数，可以看出当分频计数器值为“867”时，对应的就是串行接收信号对应位的最佳采样时刻（处于该位的中间时刻），通过位采样信号接收1bit的数据，保存到输出数据（8bit）的对应位中，在输出使能为高电平时将接收到的整个字节输出出去。

以下是接收模块的完整代码，共读者参考

module rx(

input clk,rst,

input rx,

output reg rx\_vld,

output reg [7:0] rx\_data

);

parameter DIV\_CNT = 10'd867;

parameter HDIV\_CNT = 10'd433;

parameter RX\_CNT = 4'h8;

parameter C\_IDLE = 1'b0;

parameter C\_RX = 1'b1;

reg curr\_state;

reg next\_state;

reg [9:0] div\_cnt;

reg [3:0] rx\_cnt;

reg rx\_reg\_0,rx\_reg\_1,rx\_reg\_2,rx\_reg\_3,rx\_reg\_4,rx\_reg\_5,rx\_reg\_6,rx\_reg\_7;

//reg [7:0] rx\_reg;

wire rx\_pulse;

always@(posedge clk or posedge rst)

begin

if(rst)

curr\_state <= C\_IDLE;

else

curr\_state <= next\_state;

end

always@(\*)

begin

case(curr\_state)

C\_IDLE:

if(div\_cnt==HDIV\_CNT)

next\_state = C\_RX;

else

next\_state = C\_IDLE;

C\_RX:

if((div\_cnt==DIV\_CNT)&&(rx\_cnt>=RX\_CNT))

next\_state = C\_IDLE;

else

next\_state = C\_RX;

endcase

end

always@(posedge clk or posedge rst)

begin

if(rst)

div\_cnt <= 10'h0;

else if(curr\_state == C\_IDLE)

begin

if(rx==1'b1)

div\_cnt <= 10'h0;

else if(div\_cnt < HDIV\_CNT)

div\_cnt <= div\_cnt + 10'h1;

else

div\_cnt <= 10'h0;

end

else if(curr\_state == C\_RX)

begin

if(div\_cnt >= DIV\_CNT)

div\_cnt <= 10'h0;

else

div\_cnt <= div\_cnt + 10'h1;

end

end

always@(posedge clk or posedge rst)

begin

if(rst)

rx\_cnt <= 4'h0;

else if(curr\_state == C\_IDLE)

rx\_cnt <= 4'h0;

else if((div\_cnt == DIV\_CNT)&&(rx\_cnt<4'hF))

rx\_cnt <= rx\_cnt + 1'b1;

end

assign rx\_pulse = (curr\_state==C\_RX)&&(div\_cnt==DIV\_CNT);

always@(posedge clk)

begin

if(rx\_pulse)

begin

case(rx\_cnt)

4'h0: rx\_reg\_0 <= rx;

4'h1: rx\_reg\_1 <= rx;

4'h2: rx\_reg\_2 <= rx;

4'h3: rx\_reg\_3 <= rx;

4'h4: rx\_reg\_4 <= rx;

4'h5: rx\_reg\_5 <= rx;

4'h6: rx\_reg\_6 <= rx;

4'h7: rx\_reg\_7 <= rx;

endcase

end

end

always@(posedge clk or posedge rst)

begin

if(rst)

begin

rx\_vld <= 1'b0;

rx\_data <= 8'h55;

end

else if((curr\_state==C\_RX)&&(next\_state==C\_IDLE))

begin

rx\_vld <= 1'b1;

rx\_data <= {rx\_reg\_7,rx\_reg\_6,rx\_reg\_5,rx\_reg\_4,rx\_reg\_3,rx\_reg\_2,rx\_reg\_1,rx\_reg\_0};

end

else

rx\_vld <= 1'b0;

end

endmodule

按照同样的思路，我们可以设计出发送模块，此处不再详细展开，给出完整源代码共读者参考学习。

module tx(

input clk,rst,

output reg tx,

input tx\_ready,

output reg tx\_rd,

input [7:0] tx\_data

);

parameter DIV\_CNT = 10'd867;

parameter HDIV\_CNT = 10'd433;

parameter TX\_CNT = 4'h9;

parameter C\_IDLE = 1'b0;

parameter C\_TX = 1'b1;

reg curr\_state,next\_state;

reg [9:0] div\_cnt;

reg [4:0] tx\_cnt;

reg [7:0] tx\_reg;

always@(posedge clk or posedge rst)

begin

if(rst)

curr\_state <= C\_IDLE;

else

curr\_state <= next\_state;

end

always@(\*)

begin

case(curr\_state)

C\_IDLE:

if(tx\_ready==1'b1)

next\_state = C\_TX;

else

next\_state = C\_IDLE;

C\_TX:

if((div\_cnt==DIV\_CNT)&&(tx\_cnt>=TX\_CNT))

next\_state = C\_IDLE;

else

next\_state = C\_TX;

endcase

end

always@(posedge clk or posedge rst)

begin

if(rst)

div\_cnt <= 10'h0;

else if(curr\_state==C\_TX)

begin

if(div\_cnt>=DIV\_CNT)

div\_cnt <= 10'h0;

else

div\_cnt <= div\_cnt + 10'h1;

end

else

div\_cnt <= 10'h0;

end

always@(posedge clk or posedge rst)

begin

if(rst)

tx\_cnt <= 4'h0;

else if(curr\_state==C\_TX)

begin

if(div\_cnt==DIV\_CNT)

tx\_cnt <= tx\_cnt + 1'b1;

end

else

tx\_cnt <= 4'h0;

end

always@(posedge clk or posedge rst)

begin

if(rst)

tx\_rd <= 1'b0;

else if((curr\_state==C\_IDLE)&&(tx\_ready==1'b1))

tx\_rd <= 1'b1;

else

tx\_rd <= 1'b0;

end

always@(posedge clk or posedge rst)

begin

if(rst)

tx\_reg <= 8'b0;

else if((curr\_state==C\_IDLE)&&(tx\_ready==1'b1))

tx\_reg <= tx\_data;

end

always@(posedge clk or posedge rst)

begin

if(rst)

tx <= 1'b1;

else if(curr\_state==C\_IDLE)

tx <= 1'b1;

else if(div\_cnt==10'h0)

begin

case(tx\_cnt)

4'h0: tx <= 1'b0;

4'h1: tx <= tx\_reg[0];

4'h2: tx <= tx\_reg[1];

4'h3: tx <= tx\_reg[2];

4'h4: tx <= tx\_reg[3];

4'h5: tx <= tx\_reg[4];

4'h6: tx <= tx\_reg[5];

4'h7: tx <= tx\_reg[6];

4'h8: tx <= tx\_reg[7];

4'h9: tx <= 1'b1;

endcase

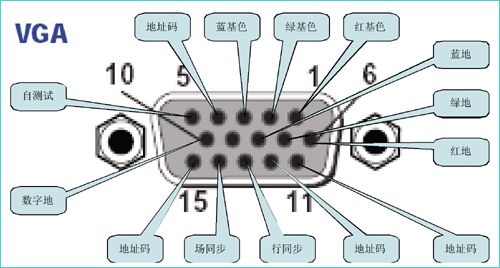
end

end

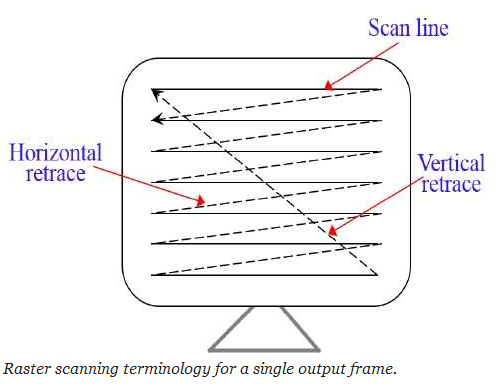
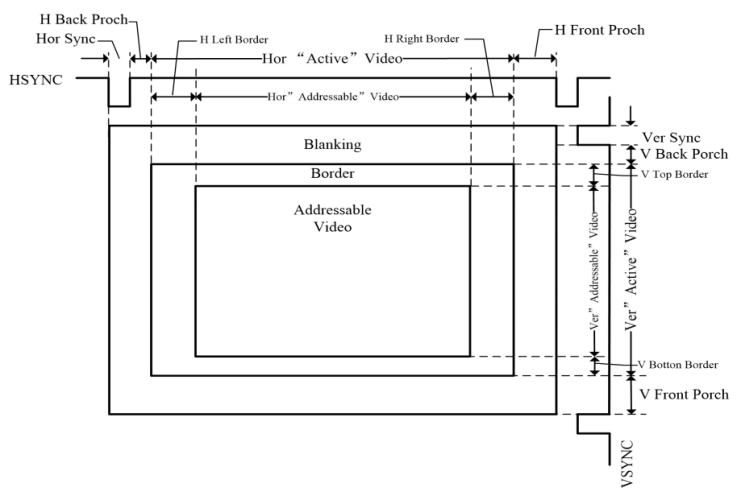
endmodule

**Step3. VGA接口**

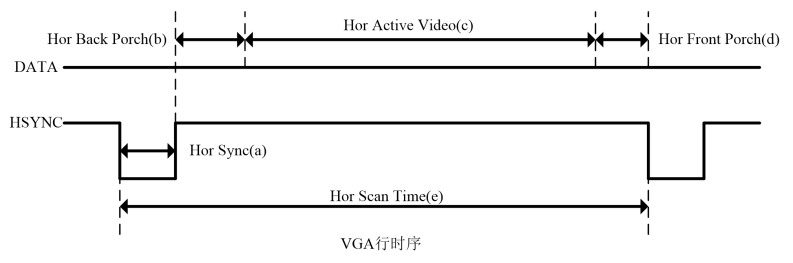
VGA（Video Graphics Array）即视频图形阵列，是IBM在1987年推出的使用模拟信号的一种视频传输标准，这种接口不支持热插拔，也不能传输音频信号，RGB三色均为模拟信号，通过电流大小表征颜色值，这种接口在如今看来有些过时，但仍然是应用最为广泛的视频接口标准。接口实物如下图所示，左侧的称为VGA公头、右侧的称为VGA母头。

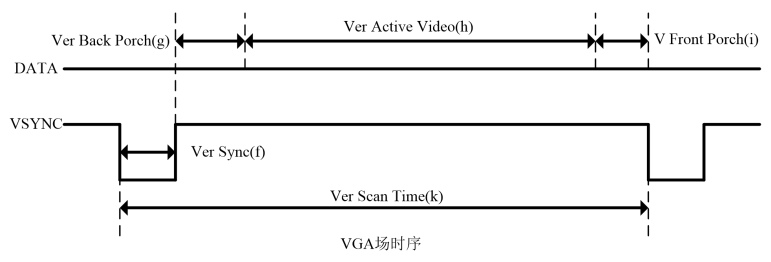
其显示原理如下图所示，通过行扫描（H\_SYNC）和列扫描（V\_SYNC）信号控制RGB数据，进行逐点扫面显示。

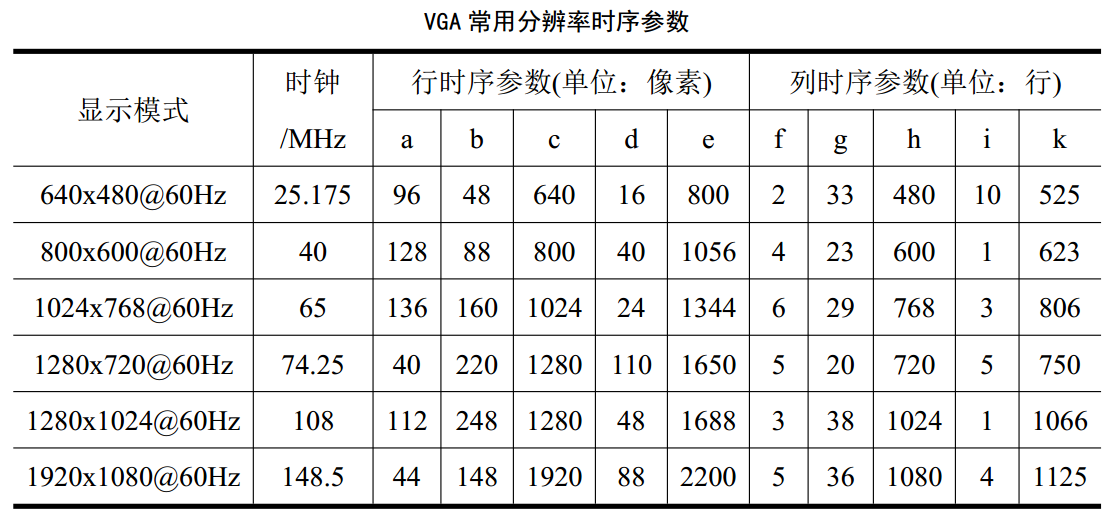
行扫描通过H\_SYNC信号来控制，每个行扫描周期分为4个阶段，分为a（行同步）,b（行消隐）,c（行视频有效）,d（行前肩）四段，其中只有c段对应显示器上一行的显示区域，除c段外，其余时段的RGB数据都应为0，时序如下图所示，：



列扫描信号通过V\_SYNC信号来控制，每个列扫描周期也分为4个阶段，分别为f（场同步）,g（场消隐）,h（场视频有效）,i（场前肩）四段，除h段外，其余时段的RGB数据都应该为0。



VGA时序对各参数都有特定的要求，以下是各常见分辨率的参数值，特别注意，行时序参数以像素为单位，列（场）时序参数是以行为单位。



我们以1024x768@60Hz为例进行介绍，根据上表中的参数要求，每一行应包含1344个像素（136+160+1024+24），每一列包含806个行周期（6+29+768+3），因此更新一帧需要1344\*806个周期，当刷新频率为60Hz时，要求时钟频率为1344\*806\*60=64995840，约合65MHz。

为在Nexys4DDR开发板上实现分辨率为1024\*768的VGA控制逻辑，我们需要一个65MHz的时钟，因此需要借助时钟管理单元（Clocking Wizard），用板载的100MHz时钟生成一个65MHz的时钟。模块接口如下所示

module clk\_wiz\_0

(

// Clock in ports 100MHz

input clk\_in1,

// Clock out ports 65MHz

output clk\_out1,

// Status and control signals

input reset,

output locked

);

然后使用65MHz作为时钟，实现VGA控制逻辑，h\_cnt从0~1343循环计数，v\_cnt则每一个h\_cnt计数周期加1，从0~805循环计数，通过两个计数器生成h\_sync（当h\_cnt在0~135时为0，其余时刻为1）、v\_sync（当v\_cnt在0~5时为0，其余时刻为1）信号，以及rgb\_data信号（当h\_cnt处于C段且v\_cnt处于H段时，rgb\_data为有效视频数据，其余时刻都为0）。



其完整代码如下所示：

module vga\_ctrl(

input clk,rst,//clk=65MHz

//output [9:0] h\_addr,v\_addr,

//output rd\_vld,

input [11:0] rd\_data, //r[3:0],g[3:0],b[3:0]

output reg hs,vs,

output [11:0] vga\_data);

parameter H\_CNT = 11'd1343; //136+160+1024+24=1345

parameter V\_CNT = 11'd805; //6+29+768+3=806

reg [10:0] h\_cnt,v\_cnt;

reg h\_de,v\_de;//data\_enable

always@(posedge clk)

begin

if(rst)

h\_cnt <= 11'd0;

else if(h\_cnt>=11'd1343)

h\_cnt <= 11'd0;

else

h\_cnt <= h\_cnt + 11'd1;

end

always@(posedge clk)

begin

if(rst)

v\_cnt <= 11'd0;

else if(h\_cnt==11'd1343)

begin

if(v\_cnt>=11'd805)

v\_cnt <= 11'd0;

else

v\_cnt <= v\_cnt + 11'd1;

end

end

always@(posedge clk)

begin

if(rst)

h\_de <= 1'b0;

else if((h\_cnt>=296)&&(h\_cnt<=1319))

h\_de <= 1'b1;

else

h\_de <= 1'b0;

end

always@(posedge clk)

begin

if(rst)

v\_de <= 1'b0;

else if((v\_cnt>=35)&&(v\_cnt<=802))

v\_de <= 1'b1;

else

v\_de <= 1'b0;

end

always@(posedge clk)

begin

if(rst)

hs <= 1'b1;

else if(h\_cnt<=11'd135)

hs <= 1'b0;

else

hs <= 1'b1;

end

always@(posedge clk)

begin

if(rst)

vs <= 1'b1;

else if(v\_cnt<=11'd5)

vs <= 1'b0;

else

vs <= 1'b1;

end

assign vga\_data = ((v\_de==1)&&(h\_de==1))? rd\_data : 12'h0;

endmodule

顶层模块负责调用时钟模块和VGA控制模块，如下所示：

module top(

input clk,rst,[11:0] rd\_data,

output hs,vs,[11:0] vga\_data);

wire clk\_65m,lock;

clk\_wiz\_0 clk\_wiz\_0(

.clk\_in1 (clk),

.clk\_out1 (clk\_65m),

.reset (rst),

.locked (lock)

);

vga\_ctrl vga\_ctrl(

.clk (clk\_65m),

.rst (~lock),

.rd\_data (rd\_data),

.hs (hs),

.vs (vs),

.vga\_data (vga\_data)

);

endmodule

关于VGA接口更详细的介绍，读者可进一步在网上搜索，此处提供一个介绍较为详细的文章供参考：

https://www.cnblogs.com/liujinggang/p/9690504.html

**实验练习(综合实验内容自定，练习仅作为选题参考)**

**题目1** 在FPGAOL平台上，利用串口终端等外设，实现简单的Shell功能，例如：在串口协议基础上，实现一个读写命令解析功能，如下表所示，功能电路接收以ASCII码格式发来的命令，并根据命令类型做出合适的响应。

|  |  |  |  |
| --- | --- | --- | --- |
| **命令** | **格式** | **示例** | **说明** |
| 写命令 | w addr data | >w 0 f0 | 将数据写入指定地址，无返回值。 |
| 读命令 | r addr | >r 0  f0 | 读取指定地址的数据，并显示在终端中。 |
| 其它命令 | xxx xxx xx | >x y  E! | 除读写命令外，均为非法命令，返回“E！” |

运行结果如下所示：



注意：网页端的串口终端在收到回车键‘\n’后才会向FPGA发送整串的ASCII码数据。

**题目2** 在FPGAOL平台上实现一个简单的片上系统（如LC3、RISC-V、MIPS等指令集），并提供能够正确运行的实例程序，最好能通过串口进行交互式操作。

**题目3** 利用串口、PS2接口、VGA接口等在Nexys4DDR开发板或FPGAOL平台上完成以下功能：

a. 完成串口接口模块，能正确的与上位机之间收发数据

b. 完成VGA接口模块，能在VGA显示器上显示文字、图像或动画

c. 完成PS2接口模块，能正确的接收到ps2接口外设发送的各种数据，并显示出来。

d. 通过串口向FPGA发送字符信息，FPGA接收到数据之后进行处理，通过编码转换、字库查询，将所接受到的字符以像素形式显示在VGA显示器上。

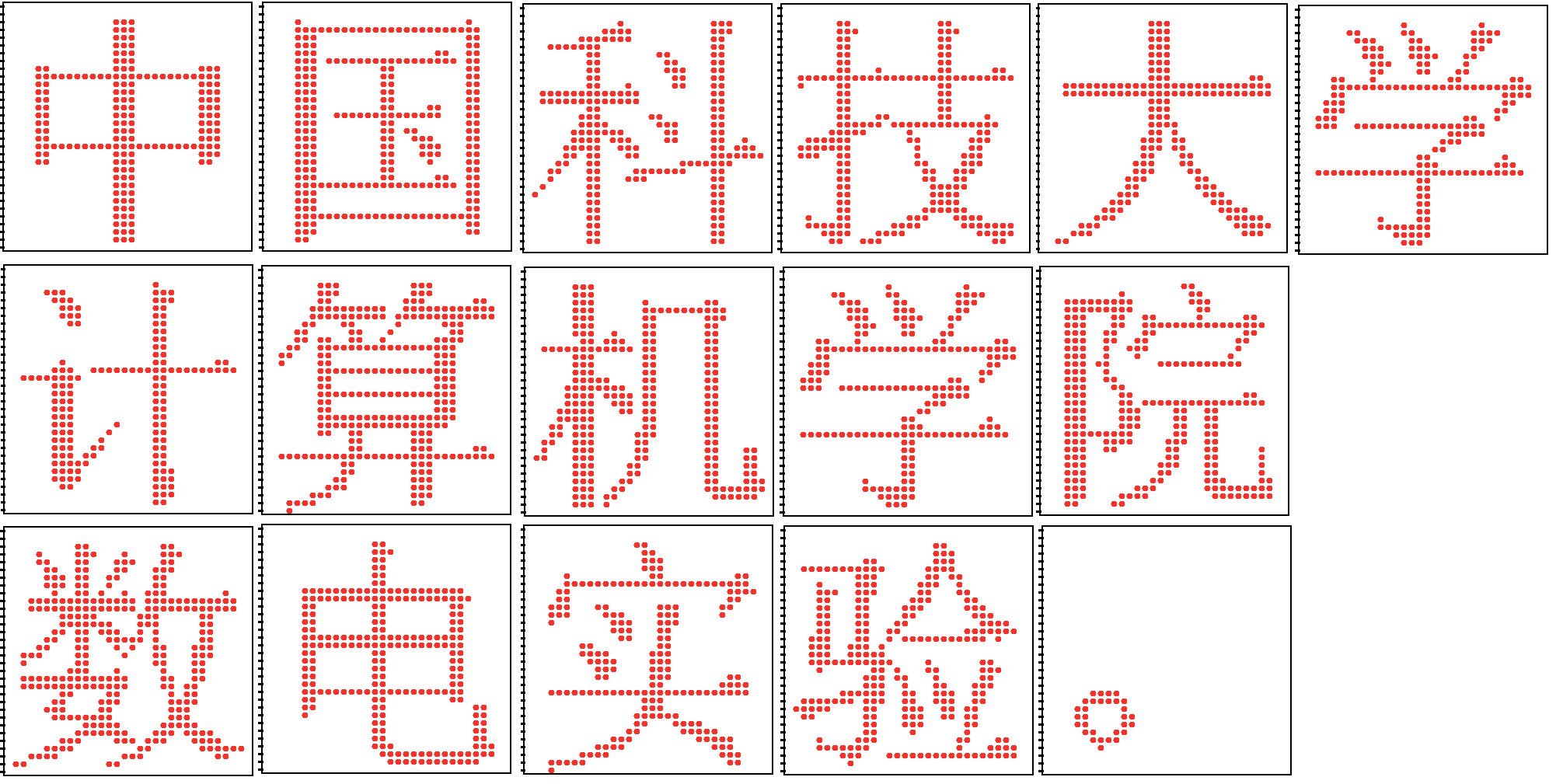
e. VGA显示器上有闪烁的光标，用于表示当前输入位置

f. 对于回车、换行、删除等特殊字符能够正确处理

g. 用键盘(+鼠标)代替串口，完成d~f功能



**题目4** 在Logisim中或者在FPGA开发板上实现逻辑电路，通过LED点阵实现汉字的循环显示。要求至少循环显示十个汉字，汉字内容及机内码的形式保存在ROM中，控制电路顺序读取数据，完成机内码到区位码的转换，通过查询字库，获取16\*16的像素数据，最终显示在LED点阵上。



**题目5** 利用所学知识完成功能电路的设计，选题、内容、方案均由自己确定，可使用外设。要求有一定原创性、有自己的核心代码、电路功能完整，运行稳定，文档详细。

**总结与思考**

1.请总结本次实验的收获

2.请评价本次实验的难易程度

3.请评价本次实验的任务量

4.请为本次实验提供改进建议