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

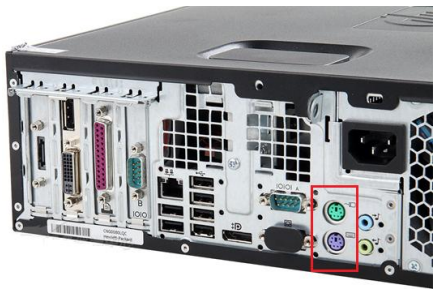
Vivado

自选外设

实验步骤

Step1. PS2 接口

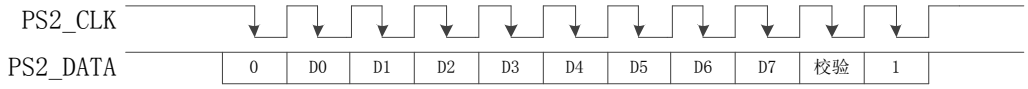
PS/2 接口最初由 IBM 开发和使用，主要用于鼠标键盘的连接，鼠标一般为绿色接头，键盘则为紫色接头，如下图所示：



Male 公的	Female 母的	5-pin DIN (AT/XT):	5 脚 DIN(AT/XT)
		1 - Clock	1—时钟
		2 - Data	2—数据
		3 - Not Implemented	3—未实现，保留
		4 - Ground	4—电源地
(Plug) 插头	(Socket) 插座	5 - +5v	5—电源+5V

Male 公的	Female 母的	6-pin Mini-DIN (PS/2):	6 脚 Mini-DIN(PS/2)
		1 - Data	1—数据
		2 - Not Implemented	2—未实现，保留
		3 - Ground	3—电源地
		4 - +5v	4—电源+5V
(Plug) 插头	(Socket) 插座	5 - Clock	5—时钟
		6 - Not Implemented	6—未实现，保留

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：

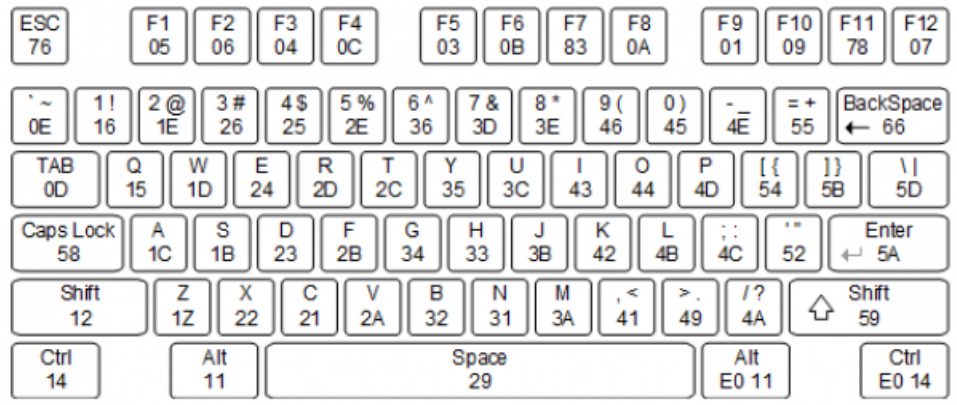
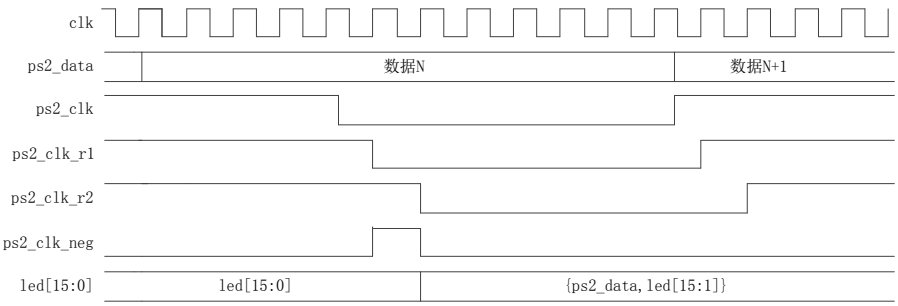


Figure 9. Keyboard scan codes.

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：

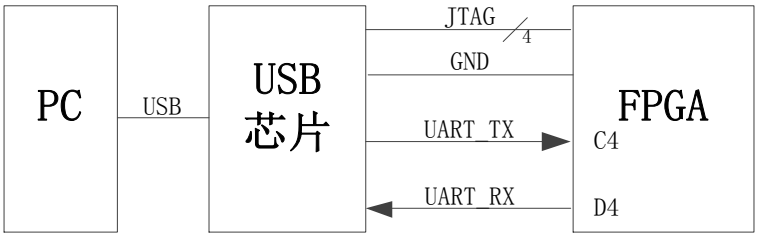
```
module ps2_test(
input   clk,rst,ps2_clk,ps2_data,
output reg  [15:0] led
    );
reg         ps2_clk_r1,ps2_clk_r2;
wire        ps2_clk_neg;
reg [3:0]   ps2_clk_cnt;

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
```

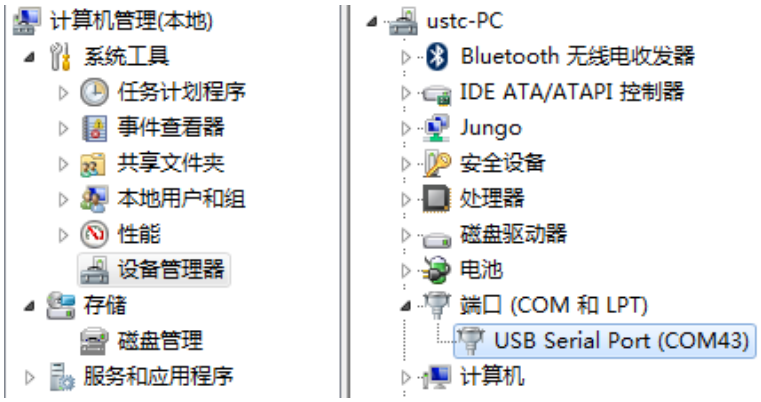
Step2. 串口

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

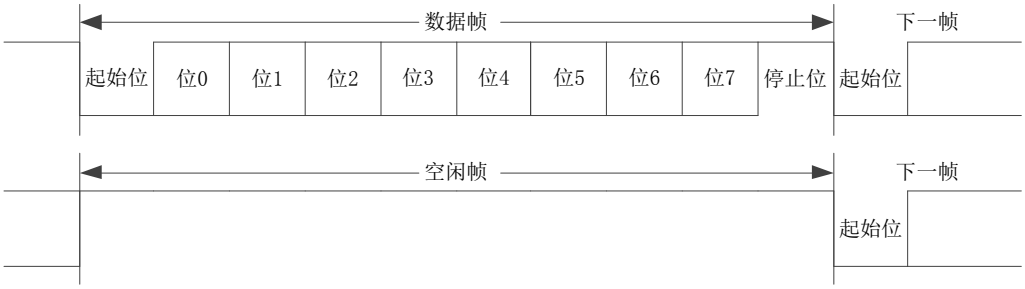
要的作用。在 Nexys4DDR 开发板中，UART 通信与 USB 烧写功能集成在了一个 microUSB 接口中，如下图所示。



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。

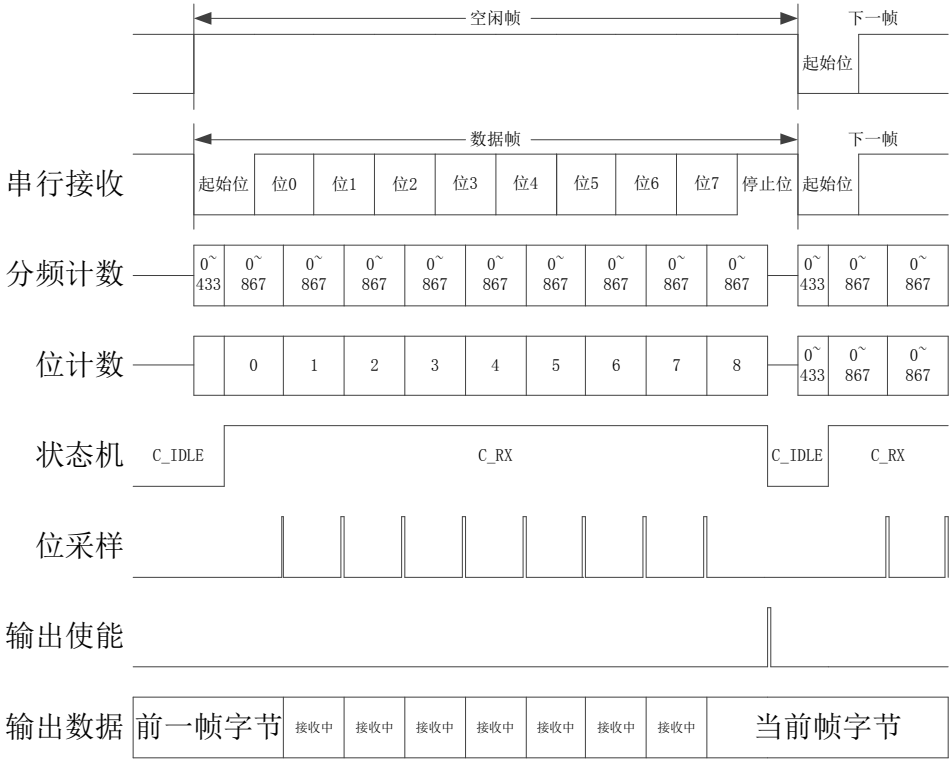
首先，我们可以通过一个简单的环回测试来了解串口的使用。在 FPGA 内，将 UART_TX（C4 引脚）输入到 FPGA 内的信号直接赋值给 UART_RX（D4 引脚），在上位机看来即实时接收刚刚发送出去的数据。在上位机，我们可以使用任意一种串口工具进行调试，如超级终端、putty、SecureCRT 等。如下图所示，需要特别注意串口号（与设备管理器中一致）、波特率（115200）、数据位（8）、停止位（1）等参数的设定。

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_re
g_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;
```

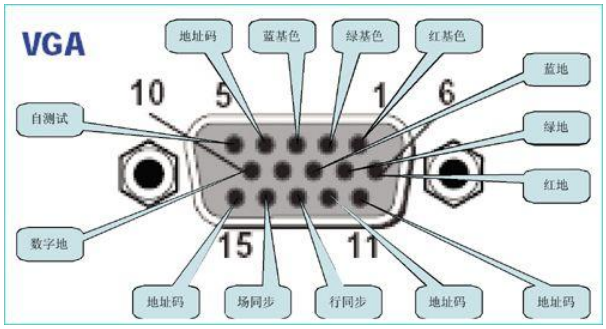
```
reg [9:0]   div_cnt;
reg [4:0]   tx_cnt;
reg [7:0]   tx_reg;
always@(posedge clk or posedge rst)
begin
    if(rst)
        curr_state  <= C_IDLE;
    else
        curr_state  <= next_state;
end
always@(*)
begin
    case(curr_state)
        C_IDLE:
            if(tx_ready==1'b1)
                next_state  = 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
```

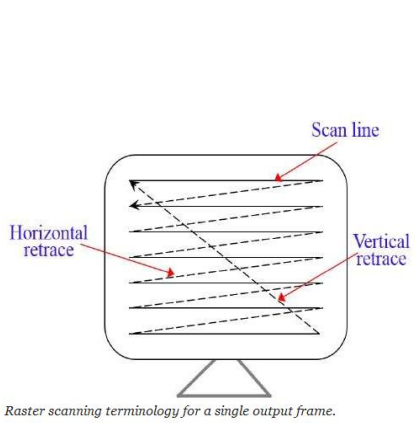
```
end
endmodule
```

Step3. VGA 接口

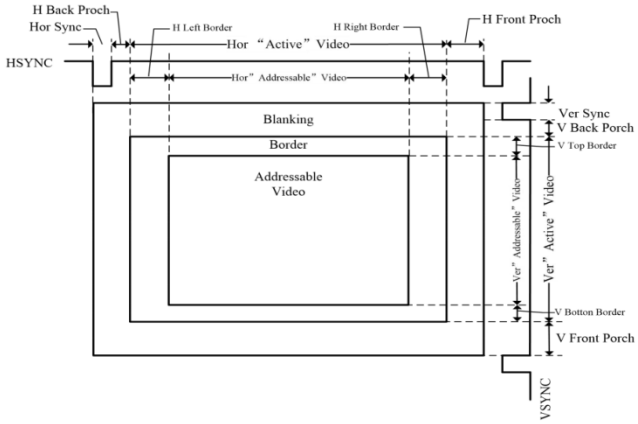
VGA（Video Graphics Array）即视频图形阵列，是 IBM 在 1987 年推出的使用模拟信号的一种视频传输标准，这种接口不支持热插拔，也不能传输音频信号，RGB 三色均为模拟信号，通过电流大小表征颜色值，这种接口在如今看来有些过时，但仍然是应用最为广泛的视频接口标准。接口实物如下图所示，左侧的称为 VGA 公头、右侧的称为 VGA 母头。



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。

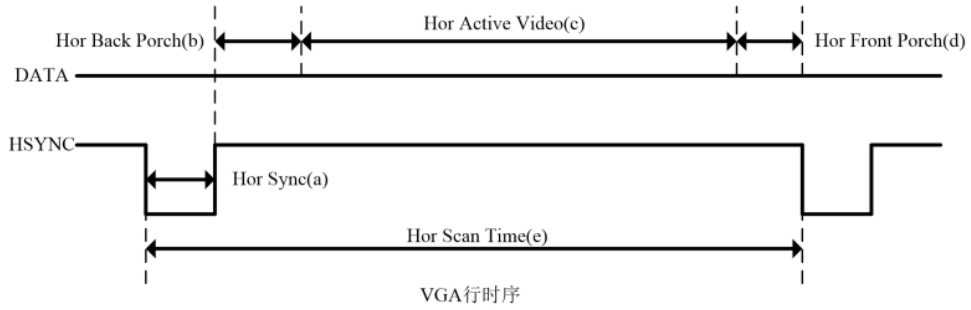


Raster scanning terminology for a single output frame.



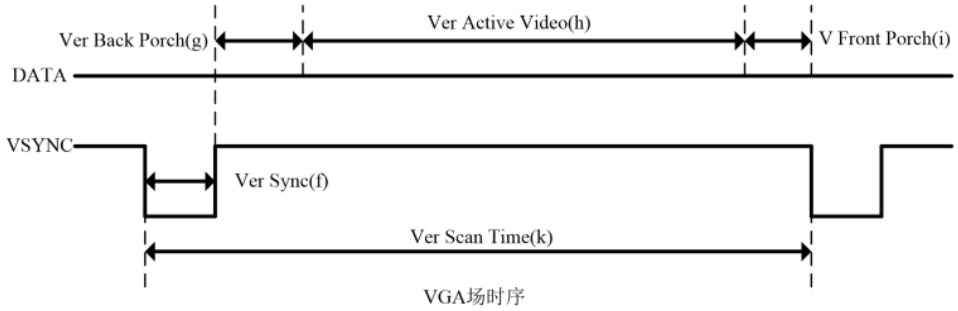
行扫描通过 H_SYNC 信号来控制，每个行扫描周期分为 4 个阶段，分为 a（行同步）,b（行消隐）,c（行视频有效）,d（行前肩）四段，

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，：



VGA行时序

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场时序

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。

VGA 常用分辨率时序参数

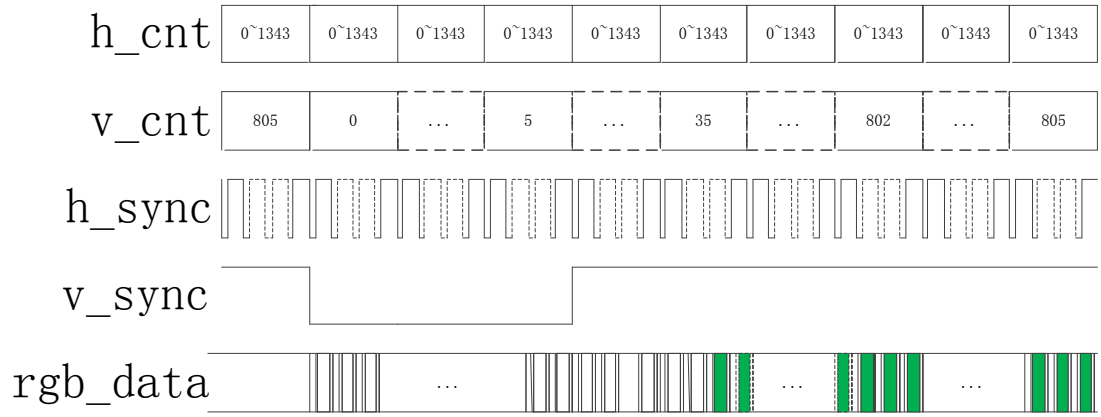
显示模式	时钟	行时序参数(单位：像素)					列时序参数(单位：行)				
	/MHz	a	b	c	d	e	f	g	h	i	k
640x480@60Hz	25.175	96	48	640	16	800	2	33	480	10	525
800x600@60Hz	40	128	88	800	40	1056	4	23	600	1	623
1024x768@60Hz	65	136	160	1024	24	1344	6	29	768	3	806
1280x720@60Hz	74.25	40	220	1280	110	1650	5	20	720	5	750
1280x1024@60Hz	108	112	248	1280	48	1688	3	38	1024	1	1066
1920x1080@60Hz	148.5	44	148	1920	88	2200	5	36	1080	4	1125

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）。



注：rgb_data只在行同步处于C段且列同步处于H段时有数据（图中绿色部分），其余时刻均为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	读取指定地址的数据，并显示

		f0	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：

```
>w 0 f0      >向地址0写入数据f0
>r 0         >从地址0读取数据
f0           返回地址0的数据（f0）
>w 1 ff      >向地址1写入数据ff
>r 1         >从地址1读取数据
35           返回地址1的数据（35）
>test        >其它命令
E!           返回错误提示：E!
```

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。

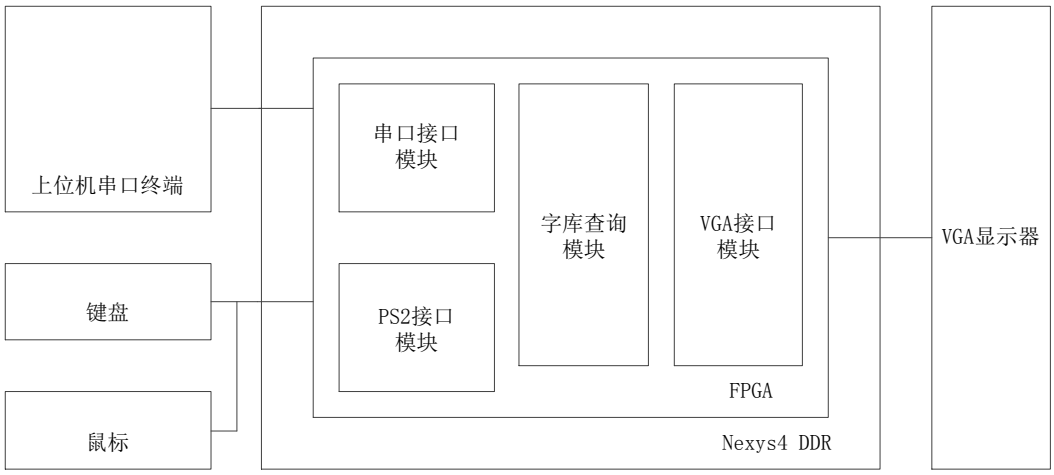
d. 通过串口向 FPGA 发送字符信息，FPGA 接收到数据之后进行处理，通过编码转换、字库查询，将所接受到的字符以像素形式显示在 VGA

显示器上。

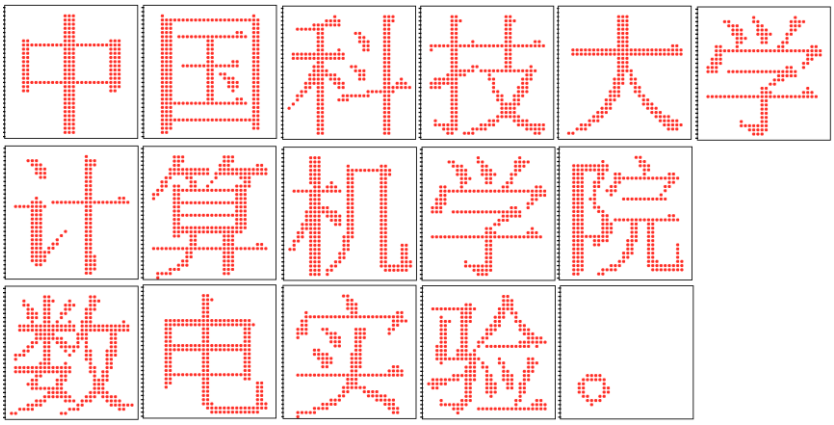
e. VGA 显示器上有闪烁的光标，用于表示当前输入位置

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