第一篇

入门拜师篇——接口基础

本篇将引领您初见数字江湖，数字系统由功能逻辑系统核心和外部接口共同组成，本篇将引领您从外部认识一个数字系统，主要是认识常见的几种接口。这是一个数字系统与外部通信的接口，可以把本篇当做接口江湖通关指南。

第1章

串　口

CHAPTER 1

1.1　技能简介

技能名称：串口。全称为串行接口或串行通信接口(通常指COM口)，是一种常见的数据通信扩展接口。此乃初入江湖必备技能，与江湖中各位大侠的初次交流全靠此技，可谓神技也。

1. 串口特点

串口是将字节拆分成一位一位的形式，在一条传输线上逐位发送，具有传输线路少、传输距离远、传输成本低、传输控制比并口通信复杂的特点。

2. 传输方式

单工方式：数据只能沿着一个方向传输，无法反向。

半双工方式：数据传输可以双向传输，但需要进行分时控制。

全双工方式：数据传输可以双向同时进行(一般采用此工作方式)。

3. 串口通信协议及引脚定义

按照协议，串口可以分为RS232、RS422、RS485等，本案例主要介绍与RS232相关的内容。

串口一般采用DB9接口，如图1-1所示，图1-1(a)为串口母头，图1-1(b)为串口公头。

(a) 母头　　(b) 公头

图1-1　串口DB9接口

以图 1-1(b)为参照，串口公头左上第一个引脚为 1 号脚，右上第一个脚为 5 号脚，左下第一个脚为 6 号脚，右下第一个脚为 9 号脚。而母头的引脚顺序则与公头有区别，公头的 5 脚为母头的 1 脚，公头的 1 脚为母头的 5 脚，公头的 6 脚为母头的 9 脚，公头的 9 脚为母头的 6 脚。见表 1-1。

表 1-1　DB9 引脚定义

引脚序号	简　称	引脚定义
1	CD	载波检测
2	RXD	接收数据（数据流向：终端到 PC）
3	TXD	发送数据（数据流向：PC 到终端）
4	DTR	数据终端准备好
5	GND	信号地线
6	DSR	数据准备好
7	RTS	请求发送
8	CTS	清除发送
9	RI	响铃指示器

其中，最常用的信号线为 RXD、TXD、GND，其他信号作为握手信号，不用它们也可以实现串口数据通信。

4. 串口通信实验目的及实验内容

学习用硬件描述语言设计完成基本接口。

掌握 RS232 接口的基本工作原理和控制方法。

通过使用 VHDL 语言完成 RS232 接口的数据发送和接收，并通过串口调试工具完成验证。

1.2　见招拆招

此乃串口技能核心要领，学习与理解的程度将直接影响其发挥的功力，如果运用得当将会事半功倍，当遇到问题一筹莫展时，运用此技也可能会出现奇迹！

串口数据帧的格式如图 1-2 所示。

图 1-2　串口数据帧格式

信号线上存在两种逻辑状态：逻辑 1(高电平)、逻辑 0(低电平)。在发送端无数据发送(闲置)时，数据线应保持在逻辑 1 的状态。

起始位(START)：发送端是通过发送一个起始位信号，来启动一组数据传输的。当发送端向数据线发送一个逻辑 0 信号时，表示发送数据即将开始。

数据位(DATA)：在起始位之后就是要传输的数据位，数据位一般采用 8 位为一个帧数据单位，低位(LSB)在前，高位(MSB)在后。

校验位(PARITY)：此位为一个特殊的数据位，用来检测接收到的数据位是否有误，一般采用奇偶校验，但是在使用时通常取消此位。

结束位(STOP)：当 8 位数据传输完后即为结束位，用一个逻辑高电平表示数据传输的结束。

帧：从起始位到结束位的时间间隔称为一帧。

串口波特率：指的是信号被调制以后在单位时间内的变化次数，比如每秒钟传送 9600 个二进制位，这时串口波特率为 9600b/s。

1.3　牛刀小试

串口通信大致可以分为 4 个设计模块：顶层模块、波特率发生器、接收模块和发送模块。其中顶层模块是整体框架，其他 3 个模块的关系如图 1-3 所示。

图 1-3　串口通信的功能模块

1. 顶层模块

顶层模块是串口结构的总体框架，包含了串口的所有功能模块：波特率发生器、接收模块和发送模块。

1) 顶层接口

顶层接口代码如下：

```
entity uart is
port(
                clk : in std_logic;                          -- 100MHz 时钟
                rst : in std_logic;
                rx : in std_logic;
                LEDRX: out std_logic_vector(7 downto 0);    -- 接收 rx 数据显示到数码管
                tx : out std_logic
        );
end entity;
```

接口信号说明如下：

clk：系统输入时钟频率 100MHz。

rst：复位信号。

rx：串口数据接收端口。

LEDRX：将接收到的 8 位串口数据显示到数码管。

tx：串口数据发送端口。

2）波特率时钟发生器

（1）计算分频系数。

本实验的串口采用全双工工作模式，波特率采用的是 115 200b/s，系统时钟 100MHz，通过正确的分频系数计算出所需要的串口波特率的时钟。

为了与 115 200 波特率一致，分频系数算式如下，

```
100000000/115200 = 867
signal cnt                    :integer range 0 to 867 : = 0;
signal uart_clk               :std_logic;
```

signal cnt：用来记录系统时钟的周期个数，系统时钟每经过一个时钟周期则 cnt 计数加 1（cnt＜＝cnt＋1），当计算到与分频系数相同的时候，这时则需要产生一个波特率时钟上升沿，同时 cnt 计数清零。

signal uart_clk：代表波特率时钟。

（2）计数器进程。

计数器进程是用来处理计数器两个状态：计数器重置和计数状态。

计数器重置：当经过一个系统时钟的上升沿时，如果计数器计满一个分频系数或者波特率的使能信号为 0，计数器将被置 0。

计数状态：当经过一个系统时钟的上升沿，如果不满足计数器重置状态，则计数器自动增 1。

计数器进程代码如下：

```
bps:process(rst,clk)
begin
                        if rst = '0' then
                            cnt < = 0;
                        elsif (clk'event and clk = '1')     then   -- 时钟计数器
                                if cnt  =  867   then
                                    cnt < = 0;
                                else
                                    cnt < = cnt + 1;
                                end if;
                        end if;
end process;
------------------------ 波特率发生器
process(clk,rst)
begin
                if rst = '0' then
                    uart_clk < = '0';
                    elsif   clk'event and clk = '1' then
                        if (cnt = 867) then
                        uart_clk < = '1';                          -- 波特率高电平
                        else
                        uart_clk < = '0';                          -- 波特率低电平
                        end if;
```

```
                end if;
end process;
```

(3) 波特率时钟产生进程。

该进程是用来产生波特率时钟的高低电平变化,包含两个状态:高电平和低电平。

当以系统时钟为基准的计数器数值与分频系数一致时则将 uart_clk 置为高电平,否则为低电平。

2. 串口接收模块

对串口接收模板的状态分析如下:

串口的接收模块是一个将串行数据转化成为并行数据的过程。当检测到接收信号后(一位低电平信号),每经过一个波特率时钟周期,系统会将 rx 接收到的数据存入寄存器中,把 8 位寄存器全部存满,然后检验下一位是否为停止位(rx 为高电平),如果检测到高电平信号(即停止位),就把接收到的 8 位数据发送给 8 位数码管,检查接收的数据是否正确。

为了表示每个阶段执行不同的命令,需要添加一个计数器来表示不同的阶段,在每一个波特率时钟的上升沿进行跳转,每计数一次跳转到下一个阶段,其中 0 阶段用来检测起始信号,1～8 阶段用来接收 rx 接收的数据,9 阶段用来检测是否为停止位。

```
signal cnt2     : integer range 0 to 9;                    -- 状态跳转计数器
signal rx8bit   : std_logic_vector(7 downto 0);            -- 接收数据的寄存器
```

将串行数据转化成为并行数据的过程代码如下:

```
process(rst,clk)
    begin
        if rst = '0' then
            rx8bit < = (others = >'1');
        elsif uart_clk'event and uart_clk = '1' then
            case cnt2 is
                when 0  = > if rx = '0' then
                                                cnt2 < = cnt2 + 1;
                                                end if;
                when 1  = > rx8bit(0)< = rx;
                                                cnt2 < = cnt2 + 1;
                when 2  = > rx8bit(1)< = rx;
                                                cnt2 < = cnt2 + 1;
                when 3  = > rx8bit(2)< = rx;
                                                cnt2 < = cnt2 + 1;
                when 4  = > rx8bit(3)< = rx;
                                                cnt2 < = cnt2 + 1;
                when 5  = > rx8bit(4)< = rx;
                                                cnt2 < = cnt2 + 1;
                when 6  = > rx8bit(5)< = rx;
                                                cnt2 < = cnt2 + 1;
                when 7  = > rx8bit(6)< = rx;
                                                cnt2 < = cnt2 + 1;
                when 8  = > rx8bit(7)< = rx;
```

```
                                                    cnt2 <= cnt2 + 1;
                when 9  =>      if rx = '1' then
                                                LEDRX <= rx8bit;
                                                end if;
                                                cnt2 <= 0;
                when others => cnt2 <= 0;
            end case;
        end if;
    end process;
```

3. 串口发送模块

1）状态分析

串口发送模块相对于接收模块操作比较简单，就是将从接收端发送过来的并行数据转换成串行数据发送出去，其实就是串口接收模块的逆过程。

发送端需要发送 10 位信号表示一个完整的串口数据帧，在空闲状态时，tx 端口应该一直保持为高电平状态，当接收端数据接收完毕后，发送端开始工作，首先在第一个波特率时钟周期，向外发送一个低电平信号作为开始发送的标志位。然后将接收端缓存中的 8 位数据分为 8 个波特率时钟周期传输给 tx 端口，当 8 位数据发送完毕后，在经过一个波特率时钟周期，将 tx 置为高电平，作为结束发送的标志位。

数据向外发送进程即为将并行信号转为串行信号的过程，同接收过程一样，需要定义一个计数器（cnt1）表示不同的发送阶段，功能与接收模块一致。

状态跳转计数器代码如下：

```
process(rst,clk)
begin
    if rst = '0' then
        cnt1 <= 0;
    elsif uart_clk'event and uart_clk = '1' then
        if cnt1 = 9 then
            cnt1 <= 0;
        else
            cnt1 <= cnt1 + 1;
        end if;
    end if;
end process;
```

将并行数据转化成为串行数据的过程代码如下：

```
process(rst,clk)
    begin
        if rst = '0' then
            tx <= '0';
        elsif clk'event and clk = '1' then
            case cnt1 is
                when 0  => tx <= '0';          -- 起始位，发送低电平
                when 1  => tx <= tx8bit(0);
                when 2  => tx <= tx8bit(1);
                when 3  => tx <= tx8bit(2);
```

```
                when 4  => tx <= tx8bit(3);
                when 5  => tx <= tx8bit(4);
                when 6  => tx <= tx8bit(5);
                when 7  => tx <= tx8bit(6);
                when 8  => tx <= tx8bit(7);
                when 9  => tx <= '1';          -- 停止位,发送高电平
                when others => tx <= '0';
            end case;
        end if;
    end process;
```

为了便于观察发送模块的实验结果,在代码中将八位二进制数“10000000”保存到发送模块计数器 tx8bit 中,然后发送端口会将二进制数 10000000 发送到串口监视器,十六进制显示为 80,观察串口监视器的结果即可验证。

至此,串口的所有模块全部实现。

2) 实验验证

可借助串口调试助手验证自己的实验结果,将波特率设置为 115 200b/s,数据位 8 位,无校验位,停止位 1 位。可以观察到数据接收窗口中不断接收串口 tx 发送的数据,这与设计一致。tx 端口发送的数据如图 1-4 所示。同时在串口调试工具的数据发送串口中发送十六进制的 44,即二进制的 01000100,观察实验板上的 LED 灯是否与发送的数据一致(1 高电平,0 低电平),rx 端口接收到的数据如图 1-5 所示。

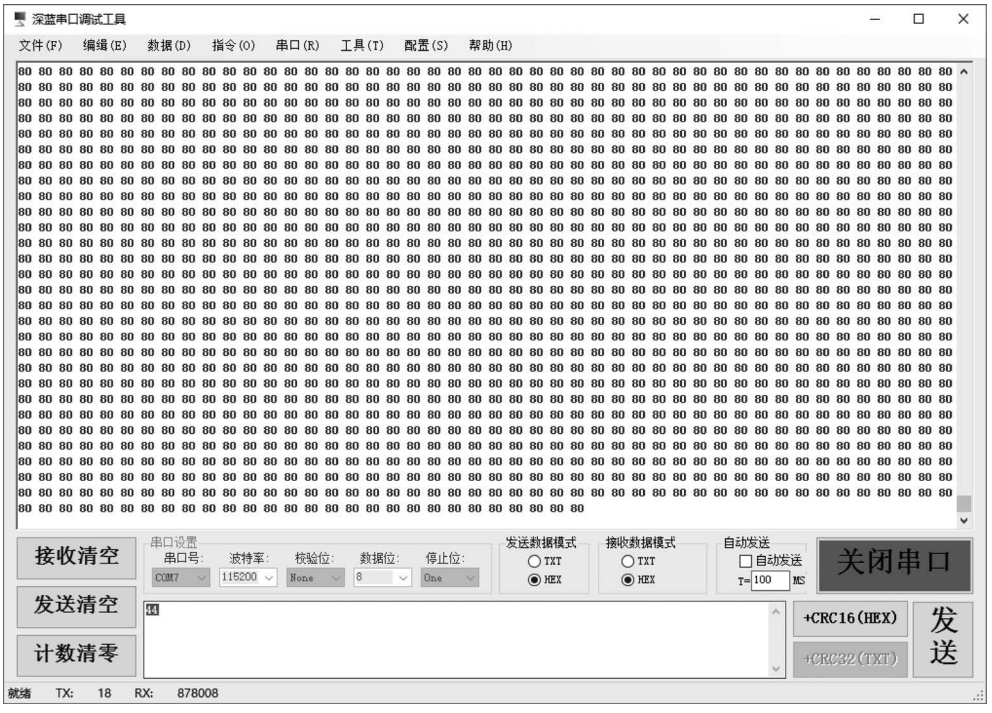


图 1-4　tx 端口发送的数据

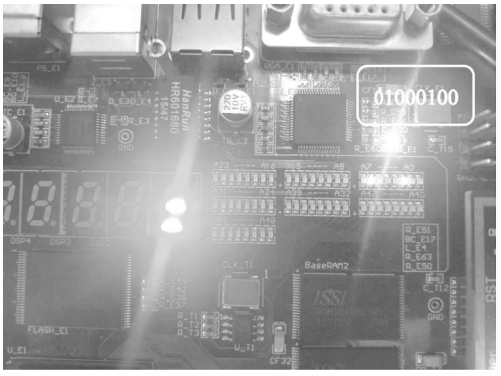


图 1-5　rx 端口接收到的数据

1.4　自身修炼

在完成以上串口实验后，可以尝试通过这种方式向串口发送其他数据，例如通过实验板中的按键输入键值等，或者将接收到的串口数据显示在数码管上。

第2章

PS/2接口(键盘)

CHAPTER 2

2.1 技能简介

技能名称：PS/2 接口。

搭配武器：键盘。

此技能配合武器使用方能产生巨大威力，该武器就是传说中的键盘，因此本技能往往配合键盘搭配使用。

1. PS/2 键盘工作方式

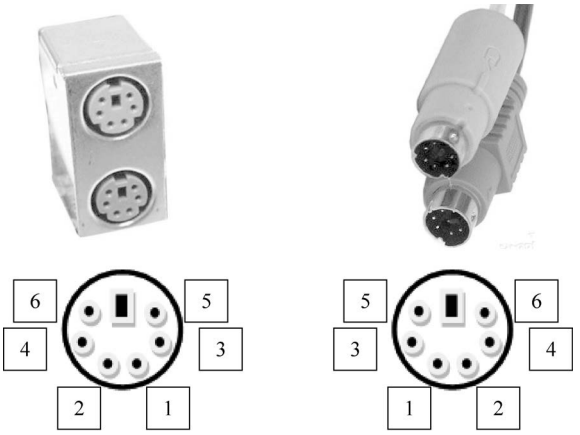
键盘是一个简单的矩阵按键的集合，通过按下不同的按键产生不同的命令，内部处理器负责监视哪个按键被按下或者被释放，同时把编码发送给键盘内部的译码器，经过译码之后通过 PS/2 接口发送给 PC，而 PS/2 接口将数据串行地发送给 PC。

2. PS/2 接口传输方式

传输方式：同步串行传输。

3. 接口引脚定义

采用 6 脚 mini-DIN 连接器，该连接器在封装上更小巧，如图 2-1 所示，图 2-1(a)为 PS/2 接口母头及其引脚顺序示意图，图 2-1(b)为 PS/2 接口公头及其引脚顺序示意图。PS/2 6 脚连接器引脚定义见表 2-1。



(a) 母头　　(b) 公头

图 2-1　6 脚 mini-DIN 连接器

表 2-1　PS/2 6 脚连接器引脚定义

引　脚　号	引 脚 定 义
1	数据位(Keyboard data)
2	未定义,保留(Reserved)
3	地(GND)
4	电源(5VDC)
5	时钟(Keyboard Clock)
6	未定义,保留(Reserved)

虽然 PS/2 接口设置有 6 个引脚,但是真正使用的也只有其中的 4 个。其中,数据位是用来与 PC 进行串行传输数据的,地即为地线,电源为 5V 供电,时钟为同步时钟信号。

4. PS/2 接口的实验内容及目的

(1) 学习并掌握 PS/2 接口的通信协议原理。
(2) 通过原理的学习完成 PS/2 键盘接口的模块设计。
(3) 实现使用 PS/2 接口并用开发板的数码管显示键盘扫描码。

2.2　见招拆招

本技能上手较快,通过对每一步招式的详细介绍,目的是使你完成修炼,达到技能与武器合二为一的境界。

1. PS/2 数据帧的格式

图 2-2 所示为键盘扫描时序图。

START	DATA1	DATA2	DATA3	DATA4	DATA5	DATA6	DATA7	DATA8	PARITY	STOP
起始位	数据 1bit	数据 1bit	数据 1bit	数据 1bit	数据 1bit	数据 1bit	数据 1bit	数据 1bit	奇偶校验	停止位

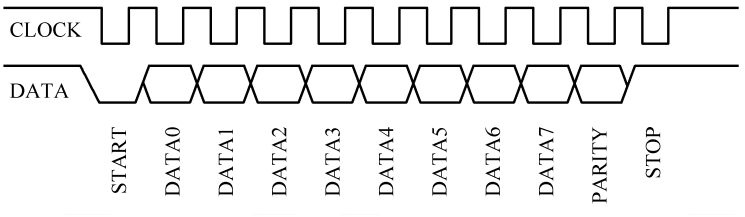


图 2-2　键盘扫描时序图

2. 帧格式解析

(1) 起始位：代表一帧数据发送的开始标志位，当检测到起始位时，才会进行数据扫描。

(2) 数据位 1～8：代表传输的有效数据，即扫描产生的有效数据。

(3) 奇偶校验：判断发送的数据是否有误，如果数据位中 1 的个数是偶数，则校验位就为 1；如果数据位中 1 的个数是奇数，校验位为 0。一旦数据有误则此帧无效。

(4) 停止位：代表一帧的结束，当接收到此信号，则停止扫描采样。

当按下键盘上的某一按键时，就会产生一组连续的波形，通过键盘的时钟采样就会产生唯一的编码，该编码称为通码，当按键弹起时，同样会产生一组编码，该编码称为断码。所有的通码都是 1 字节，而断码需要 2 字节，断码是在通码之前加入 F0。

例如，当按下 F 键，这时键盘就会按照上面的帧格式产生如下一组数据：

起始位：0；

数据位(8bit)：0010(2) 1011(B)；

奇偶校验位：0；

停止位：1；

F 键的 16 进制通码即为 2B；

F 键的 16 进制断码即为 F02B。

2.3　牛刀小试

本次实验的内容就是将键盘上的按键通码的 16 进制内容显示在两个数码管上。

PS/2 键盘接口设计可以分为 3 个设计模块：顶层模块、键盘数据处理模块、数码管模块。关键模块如图 2-3 所示。

1. 顶层模块

顶层模块是 PS/2 键盘接口的整体框架，它包含了 PS/2 键盘接口内部的数据处理模块元件例化，及为了保证能显示 1 字节的通码，元件例化生成的两个 7 段数码管。

图 2-3　PS/2 键盘接口设计模块

顶层接口代码如下：

```
entity top is
port(
        datain: in std_logic;
        clkin: in std_logic;      -- 11.0592M
        fclk: in std_logic;      -- 100M
        rst_in: in std_logic;
        seg0: out std_logic_vector(6 downto 0);
        seg1: out std_logic_vector(6 downto 0)
);
end top;
```

接口信号说明如下。

datain：键盘的数据输入接口。

clkin：PS/2 同步时钟接口。

fclk：系统时钟。

rst_in：系统复位。

seg0：数码管。

seg1：数码管。

2. 键盘数据处理模块

本模块是 PS/2 接口与键盘进行数据处理的核心模块，在本模块内需要对几个关键信号进行处理，包括开始位、数据位的采集、奇偶校验位、停止位。

键盘数据处理模块接口代码如下：

```
entity Keyboard is
port (
            datain, clkin : in std_logic;
            fclk, rst : in std_logic;
            scancode : out std_logic_vector(7 downto 0)
    );
end Keyboard;
```

接口信号说明如下。

datain：键盘数据输入接口。

clk：键盘同步时钟输入接口。

fclk：系统时钟。

rst：系统复位。

scancode：扫描码信号。

1) 同步时钟下降沿的判断

因为规定同步时钟下降沿的时刻对数据进行采样,所以需要对键盘同步时钟进行电平判断,以确定数据采样时间点。由于系统时钟频率高于同步时钟,可以使用系统时钟对采样时钟进行电平信号采集,连续采样两个时间点判断电平高低,如果第一个采样点为高电平,第二个采样点为低电平,则开始对数据进行采样。

关键信号:clk,clk1,clk2。

clk1:用来保存第一个采样点。

clk2:用来保存第二个采样点。

clk:当 clk1 的采样数据为 1,clk2 的采样电平为 0 时,此时为同步时钟的下降沿,对数据进行采样,同时把 clk 赋为'1'。即,每当 clk='1'的时候,开始对键盘输入数据进行采样。

2) 数据帧的采样处理

键盘的一帧数据是由 11 个 bit 组成,因此需要按照数据帧的格式对数据判断并进行采样。这里采用状态机设计,通过设计 13 个状态表示数据帧处理的每一个过程。

```
type state_type is (delay, start, d0, d1, d2, d3, d4, d5, d6, d7, parity, stop, finish) ;
```

其中,delay 代表初始状态,start 代表数据帧的开始位,d0～d7 代表键盘数据,parity 代表奇偶校验位,stop 代表停止位,finish 代表数据接收完毕,将扫描码发送出去阶段。

这里定义一个 code 信号作为 8 位数据位的缓存,当执行到停止位阶段时,把 code 信号发送给 scancode,以保证数据的完整性。

首先系统进入 delay 状态,start 作为下一个状态,接下来就是数据帧中每一位的信号判断,在 start 状态中对开始位进行判断,当键盘数据为 0 时则进入下一个状态,否则回到初始状态。依次类推进行每一位的判断。

3) 奇偶校验位的处理

为了保证数据接收的正确性,需要对接收到的 8 位数据进行奇偶校验,需要加入一个信号来保存奇偶校验的结果。

```
signal odd     :     std_logic;——奇偶校验位
```

而键盘自己产生的奇偶校验把校验位也加入到数据中进行奇校验,而数据帧采样只是对 8 位数据进行校验,属于偶校验,因此需要对比系统的校验位与 odd 位的值,如果两个校验位结果不同,则数据正确,否则返回到初始阶段。

键盘数据处理模块代码如下:

```
case state is
                    when delay =>
                        state <= start;
                    when start =>
                        if clk = '1' then
                            if data = '0' then
                                state <= d0;
                            else
                                state <= delay;
                            end if;
                        end if;
```

```
when d0 =>
    if clk = '1' then
        code(0) <= data;
        state <= d1;
        ⋮
WHEN parity =>
    IF clk = '1' then
        if (data xor odd) = '1' then
            state <= stop;
        else
            state <= delay;
        end if;
    END IF;
WHEN stop =>
    IF clk = '1' then
        if data = '1' then
            state <= finish;
        else
            state <= delay;
        end if;
    END IF;
```

3. 数码管模块

本模块负责接收扫描码，同时进行数据解析并显示到 7 段数码管上。需要注意的是，1 字节的扫描码需要两个译码器进行译码并用两个 7 段数码管显示（其中一个数码管负责高 4 位译码，另一个数码管对低四位进行译码，再合并成一个通码），因此在顶层模块要同时例化两个数码管元件。

```
entity seg7 is
port(
code: in std_logic_vector(3 downto 0);
seg_out : out std_logic_vector(6 downto 0)
);
end seg7;
```

接口信号说明如下。

code：负责接收扫描码。

seg_out：显示译码后的扫描码。

至此，PS/2 接口的招式全部介绍完毕，可通过对比键盘扫描码对照表验证实验的正确性。

2.4 自我修炼

传说中还有另一种武器常常与键盘一起出现，它就是传说中的鼠标。通过查阅鼠标的数据发送规则完成鼠标的 PS/2 接口实验，通过 7 段数码管显示鼠标位置的坐标。

第3章

PS/2接口(鼠标)

CHAPTER 3

3.1 技能简介

技能名称：PS/2 接口。

搭配武器：鼠标。

PS/2 接口常常也同鼠标搭配使用，键盘和鼠标均为行走江湖常见武器。

1. PS/2 鼠标工作方式

PS/2 鼠标一般为两键或者两键加滚轮(三键)的结构，同时配有定位器。通过对鼠标的移动或者按下不同的按键产生不同的数据发送给主机进行控制。

2. 接口传输方式

PS/2 鼠标接口为双向同步串行协议，与键盘传输协议一致。

3. 接口引脚定义

与键盘接口引脚定义相同。

4. PS/2 鼠标接口的实验内容及目的

(1) 学习并掌握 PS/2 鼠标接口的通信协议原理。

(2) 使用 FPGA 实现 PS/2 鼠标接口。

(3) 结合数码管完成鼠标光标位置坐标的显示。

3.2 见招拆招

鼠标也是接口江湖常见武器之一，较之于键盘，掌握本技能的难度稍有提高，一旦掌握将会极大提升。鼠标与键盘在数据的传输上

不完全相同，键盘与主机之间只是单向通信，即键盘向主机发送数据，主机只接收数据。而鼠标则是双向通信，主机既向鼠标发送数据，同时也接收鼠标发送的数据。因此，在数据传输的控制上相对于键盘要复杂一些。这里以三键鼠标为例，介绍一下鼠标的工作原理。

3.2.1　PS/2 鼠标数据帧格式

PS/2 鼠标数据帧格式见表 3-1。

表 3-1　PS/2 鼠标数据帧格式

	DATA7	DATA6	DATA5	DATA4	DATA3	DATA2	DATA1	DATA0
BYTE1	Y overflow	Y overflow	Y Sign bit	Y Sign bit	Always 1	Middle button	Right button	Left button
BYTE2	X movement							
BYTE3	Y movement							

鼠标向主机传输数据时序如图 3-1 所示。主机向鼠标传输数据时序如图 3-2 所示。

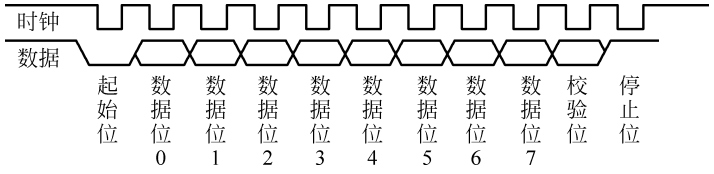


图 3-1　鼠标向主机传输数据时序

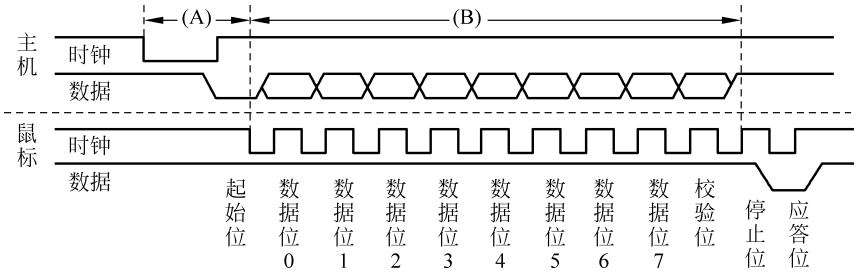


图 3-2　主机向鼠标传输数据时序

3.2.2　帧格式解析

标准的三键 PS/2 接口鼠标一般是由 3 字节构成，其中：

BYTE1 中的 DATA0、DATA1、DATA2 分别代表三键鼠标的左、右、中三个按键的状态，状态值为 0 表示抬起，为 1 表示按下；

BYTE2 表示鼠标 X 轴移动计量值，是二进制补码值；

BYTE3 表示鼠标 Y 轴移动计量值，是二进制补码值。

移动计量值保存在对应的位移计数器中，位移计数器用九位的二进制整数补码表示。其最高位作为符号位出现，而每一字节只有 8 位，因此这个符号位放在第一字节中。当鼠标位置发生变化时，位移计数器的值被更新。位移计算器可表示的范围是－255～＋255。如果鼠标的移动超出这个范围，则对应的溢出标志位就会被设置，一旦移位计数器将数据发送

给主机,并发送成功,则移位计数器会被复位。

3.3　牛刀小试

鼠标有多种操作模式,但一般都工作在一种模式下,即 Stream 模式。在此模式下,主机向鼠标发送 0XF4 命令,一旦鼠标接收到此命令,会用 0XFA 进行应答,并复位移位寄存器,完成鼠标的初始化工作。主机根据应答命令来判断鼠标是否应答正确,如果鼠标应答正确,则接受并解析鼠标发送的数据包,从中获取相应的数据,作出反馈。本实验结合数码管,将鼠标发送的数据包进行解析并用数码管显示出来。

设计思路:使用 3 个数码管显示鼠标 X 坐标值,再用 3 个数码管显示 Y 坐标值,左中右三个按键可以通过余下的数码管和设计好的特定值来显示(例如:当按下左键时数码管就显示数字 1)。

1. 设计流程

设计流程如图 3-3 所示。

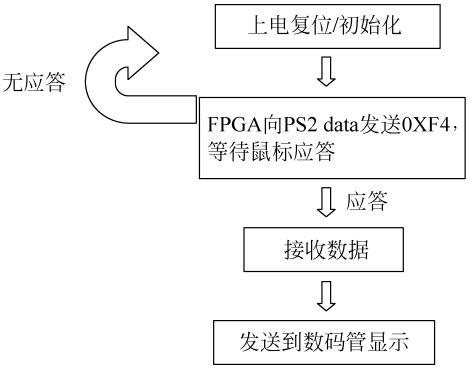


图 3-3　设计流程图

2. 接口定义

接口定义代码如下:

```
entity ps2_mouse is
  port( clk_in : in std_logic;
        reset_in : in std_logic;
        ps2_clk : inout std_logic;
        ps2_data : inout std_logic;
        left_button : out std_logic;
        right_button : out std_logic;
        middle_button : out std_logic;
        mousex: buffer std_logic_vector(9 downto 0);
        mousey: buffer std_logic_vector(9 downto 0);
        error_no_ack : out std_logic );
end ps2_mouse;
```

关键信号说明代码如下：

```
constant x_max:integer: = 800;
constant y_max:integer: = 600;
constant total_bits: integer: = 33;              -- number of bits in one full packet
constant watchdog: integer: = 100;               -- number of sys_clks for 400usec
constant debounce_timer : integer: = 2;          -- number of sys_clks for debounce:2
signal m1_state, m1_next_state : m1statetype;    -- the two states
signal m2_state, m2_next_state : m2statetype;
signal watchdog_timer_done,debounce_timer_done : std_logic; -- signals of command from host to
                                                            -- mouse
signal q : std_logic_vector(total_bits - 1 downto 0);         -- bit sequence
signal bitcount : std_logic_vector(5 downto 0);               -- bit count

signal watchdog_timer_count : std_logic_vector(8 downto 0);  -- wait time
signal debounce_timer_count : std_logic_vector(1 downto 0);  -- debounce time
signal ps2_clk_hi_z : std_logic;        -- without keyboard, high z equals 1 due to pullups
signal ps2_data_hi_z : std_logic;       -- without keyboard, high z equals 1 due to pullups

signal clean_clk : std_logic;           -- debounced output from m1, follows ps2_clk
signal rise,n_rise : std_logic;         -- output from m1 state machine
signal fall,n_fall : std_logic;         -- output from m1 state machine

signal output_strobe : std_logic;       -- latches data into the output registers(选通脉冲)
signal packet_good : std_logic;         -- check whether the data is valid
signal clk,reset : std_logic;
signal count : std_logic_vector(20 downto 0);
```

代码中把鼠标 X、Y 方向移动的坐标范围分别定义为 800 和 600。

由于鼠标的数据流长度为 3 字节，每字节加上起始位、校验位和停止位，一共是 33bit。

FPGA 与鼠标的通信分为两个状态：

- FPGA 向鼠标写入数据的状态；
- FPGA 从鼠标读取数据的状态(与键盘一致)。

3. 部分代码

(1) 当 FPGA 上电后，主机首先向鼠标发送数据并等待鼠标应答。代码如下：

```
case m2_state is
    when m2_reset  = >     -- after reset, send command to mouse.
      m2_next_state < =  m2_hold_clk_l;

    when m2_wait  = >
      if (fall = '1') then
        m2_next_state < =  m2_gather;
      else
        m2_next_state < =  m2_wait;
      end if;

    when m2_gather  = >
      if watchdog_timer_done = '1' and bitcount = total_bits then
        m2_next_state < =  m2_use;
      else
```