基于 FPGA 的 PS2 鼠标接口设计方法及其应用

王小明 2007.4.10

电话：13510618289　邮箱：wang.x_m@163.com

【摘　要】利用现场可编程逻辑器件 FPGA 接收处理 PS/2 接口鼠标输入信息，并用 VGA 作为输出设备，显示当前鼠标状态及位置。

【关键词】现场可编程逻辑器件，FPGA，PS/2，状态机。

1. 引言

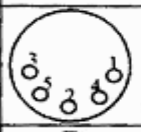
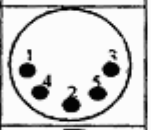
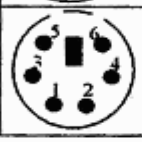
当前嵌入式系统技术已得到了广泛应用，但传统嵌入式系统的人机接口多采用小键盘操作的文本菜单方式，用户操作较为不便。本设计利用FPGA实现对PS/2 接口鼠标的控制，并在以VGA作为输出设备的单片机系统上初步实现图形化用户界面的方案。具有成本低、效果好等特点，具有很强的实用性。

FPGA 现场可编程门阵列（Field Programmable Gate Array）是 20 世纪 80 年代中期出现的 高密度 可编程逻辑器件。FPGA 及其软件系统是开发数字电路的最新技术。他利用 EDA 技术，以电路 原理图、硬件描述语言、状态机等形式输入设计逻辑；他提供功能模拟、时序 仿真等模拟手段，在功能模拟和时序仿真度满足要求后，经过一系列的变换，将输入逻辑转 换成 FPGA 器件的编程文件，以实现专用集成电路。本设计选用 Altera 公司推出的 CycloneII 系列 的 EP2C5T144C8 现场可编程门陈列来设计 PS/2 接口，体积减小，可靠性提高。

2. PS/2 接口和协议

2.1. 接口的物理特性

PS/2 接口用于许多现代的鼠标和键盘，由ＩＢＭ最初开发和使用。物理上的PS/2 接口有两种类型的连接器：５脚的DIN和６脚的MINI－DIN。图１就是两种连接器的引脚定义。使用中，主机提供＋５Ｖ电源给鼠标，鼠标的地连接到主机电源地上。

	插座(孔)	插头(针)
5脚的DIN		
6脚的mini-DIN		

	5脚的DIN	6脚的mini-DIN
1	时钟(CLOCK)	数据(DATA)
2	数据(DATA)	未实现、保留
3	未实现、保留	电源地(GND)
4	电源地(GND)	电源+5V(VCC)
5	电源+5V(VCC)	时钟(CLOCK)
6		未实现、保留

图 1 PS/2 接口连接器引脚定义

2.2. 接口协议原理

PS/2 鼠标接口采用一种双向同步串行协议。即每在时钟线上发一个脉冲，就在数据线上发送一位数据。在相互传输中 ，主机拥有总线控制权，即它可以在任何时候抑制鼠标的发送。方法是把时钟线一直拉低，鼠标就不能产生时钟信号和发送数据。在两个方向的传输中，时钟信号都是由鼠标产生，即主机不产生通信时钟信号。

如果主机要发送数据，它必须控制鼠标产生时钟信号。方法如下：主机首先下拉时钟线至少１００μs 抑制通信，然后再下拉数据线，最后释放时钟线。通过这一时序控制鼠标产生时钟信号。当鼠标检测到这个时序状态，会在１０ｍｓ内产生时钟信号。如图 3 中(Ａ)时序段。主机和鼠标之间，传输数据帧的时序如图 2 、图 3 所示。

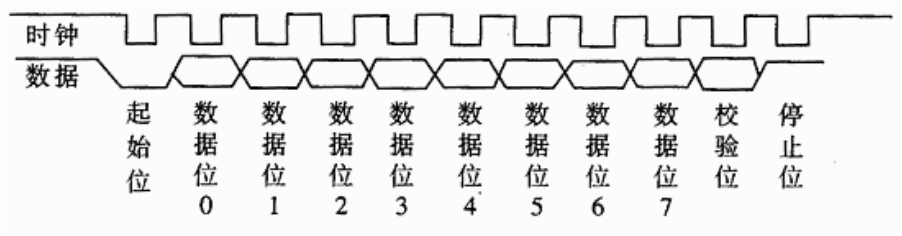


图 2 鼠标到主机传输时序

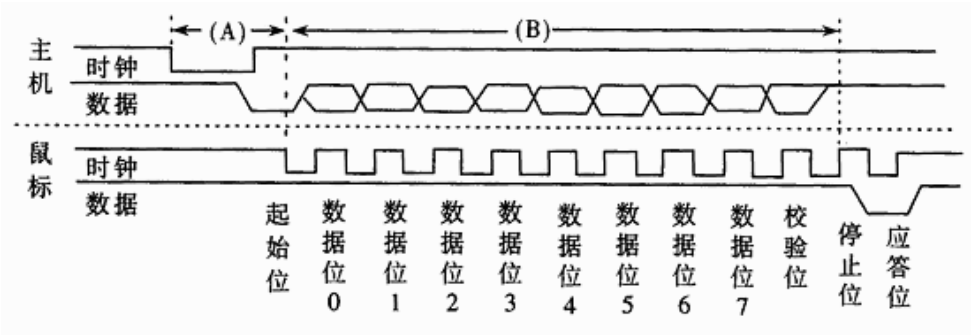


图 3 主机到鼠标的传输时序

2.3. PS/2 鼠标的工作模式和协议数据包格式

2.3.1. PS/2 鼠标的四种工作模式

PS/2 鼠标的四种工作模式是：Reset 模式，当鼠标上电或主机发复位命令(0xFF)给它时进入这种模式；Stream 模式　鼠标的默认模式，当鼠标上电或复位完成后，自动进入此模式，鼠标基本上以此模式工作；Remote 模式，只有在主机发送了模式设置命令(0xF0)后，鼠标才进入这种模式；Wrap 模式，这种模式只用于测试鼠标与主机连接是否正确。

2.3.2. 数据包结构

PS/2 鼠标在工作过程中，会及时把它的状态数据发送给主机。发送的数据包格式如表 1 所示。

表1 鼠标发送的数据包格式

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Byte1	Y overflow	X oberflow	X sipn bit	X sign bit	Always 1	Middle Btn	Right Btn	Left Btn
Byte2	X Movement							
Byte3	Y Movement							
Byte4	Z Movement							

Byte 1 中的 Bit0、Bit1、Bit2 分别表示左、右、中键的状态，状态值 0 表示释放　1 表示按下。Byte2 和 Byte3 分别表示 X 轴和 Y 轴方向的移动计量值，是二进制补码值。Byte4 的低四位表示滚轮的移动计量值，也是二进制补码值，高四位作为扩展符号位。这种数据包由带滚轮的三键三维鼠标产生。若是不带滚轮的三键鼠标，产生的数据包没有 Byte4 其余的相同。

3. VGA 信号时序

图 4 所示是计算机 VGA（640×480，60Hz）图像格式的信号时序图，其点时钟 DCLK 为 25．175MHz，场频为 59．94Hz。图中，Vsync 为场同步信号，场周期 Tvsync 为 16．683ms，每场有 525 行，其中 480 行为有效显示行，45 行为场消隐期。场同步信号 Vs 每场有一个脉冲，该脉冲的低电平宽度 twv 为 63μs（2 行）。场消隐期包括场同步时间 twv、场消隐前肩 tHV（13 行）、场消隐后肩 tVH（30 行），共 45 行。行周期 THSYNC 为 31．78μs，每显示行包括 800 点，其中 640 点为有效显示，160 点为行消隐期（非显示区）。行同步信号 Hs 每行有一个脉冲，该脉冲的低电平宽度 tWH 为 3．81μs（即 96 个 DCLK）；行消隐期包括行同步时间 tWH，行消隐前肩 tHC（19 个 DCLK）和行消隐后肩 tCH（45 个 DCLK），共 160 个点时钟。复合消隐信号是行消隐信号和场消隐信号的逻辑与，在有效显示期复合消隐信号为高电平，在非显示区域它是低电平。

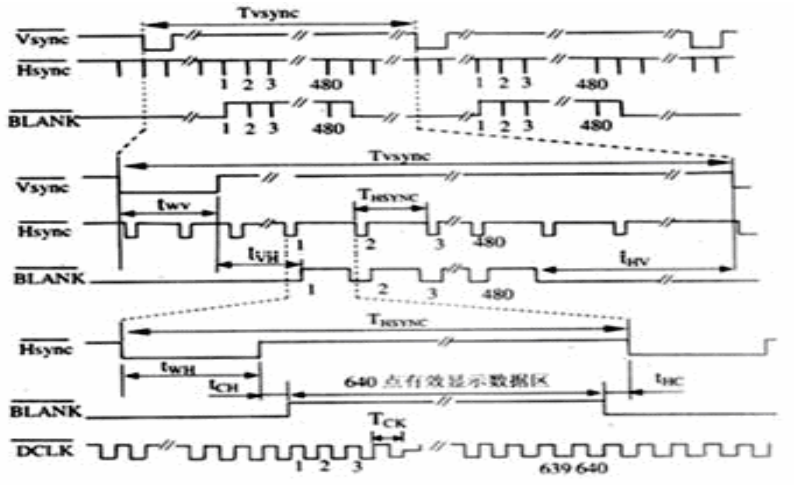


图 4 VGA 显示驱动时序

4. 设计实现

4.1. 实现功能

1、 用 FPGA 实现 PS/2 鼠标接口。

2、 鼠标左键按下时十字形鼠标图象的中间方块改变颜色，右按下时箭头改变颜色。

3、 Reset 按键：总复位。

4.2. 设计原理

主机复位后，首先向鼠标发送初始化命令（0xf4）。当鼠标收到命令字后会给出一个应答字节（0xfa）。主机根据应答字节来判断鼠标是否正确应答。应答正确则进行接收鼠标数据包，然后从接收到的数据包中获得鼠标位置及状态数据，并输出给显示模块。显示模块在 CRT 上显示出当前鼠标的状态和位置。否则，停止处理。如图 5

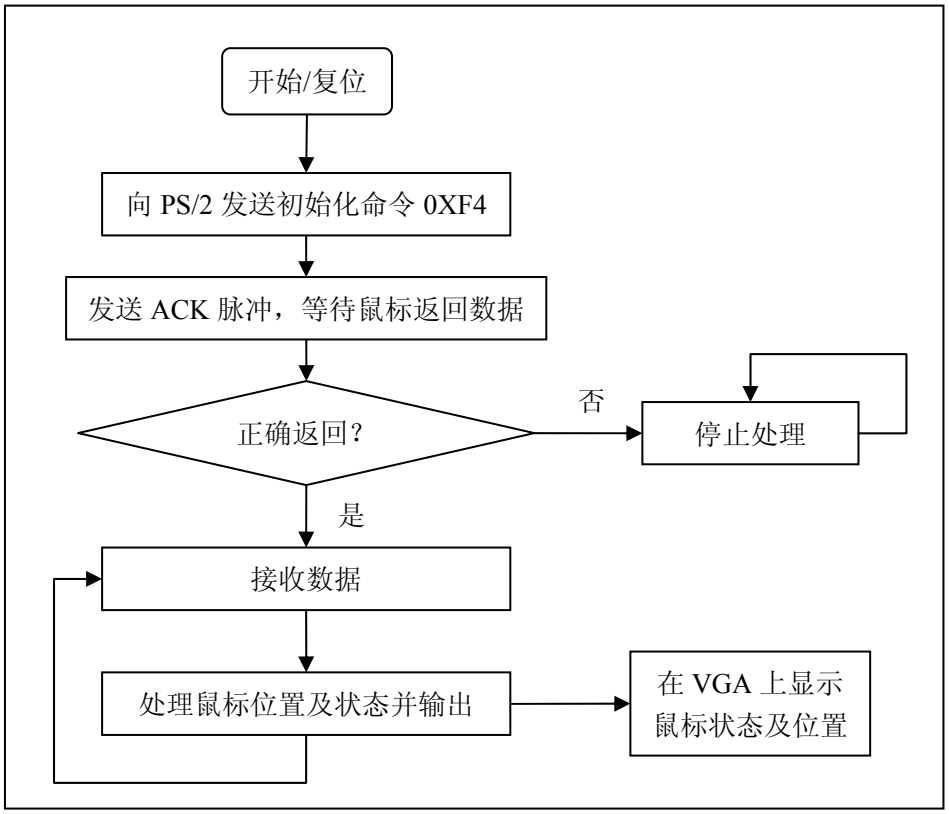


图 5 运行流程

如图 6，当状态机 m2_state 复位时，即进入 m2_reset 状态，并在一个 clk 周期后进入 m2_hold_clk_1 状态，当 ps2_clk_hi_z（时钟线）被拉低并保持 400us 后进入 m2_data_low_1 状态，此时向鼠标发送起始位和 d[0]、d[1]（d[0]=d[1]=0）。完成后进入 m2_data_high_1 状态，发送 d[2]（d[2]=1）并进入 m2_data_low_2 状态，此时向鼠标发送 d[3]位（d[3]=0），完成发送进入 m2_data_high_2 状态，向鼠标发送 d[4],d[5],d[6],d[7]（d[4]=d[5]=d[6]=d[7]=1），

完成发送进入 m2_data_low_3 状态，向鼠标发送奇偶校验位，然后进入 m2_data_high_3 状态，将数据线拉高，等待鼠标返回应答信号。若 PS/2 时钟信号下降沿来临时，数据线仍未高电平，则进入 m2_error_no_ack 状态，此时握手失败，系统将保持 m2_error_no_ack 状态直道下一次复位。否则进入 m2_await_response 状态接收应答字，接收完成进入 m2_verify 数据校验，然后进入 m2_use 状态，锁存输出数据，并进入 m2_wait 状态，等待接收数据，当检测到时钟下降沿后进入 m2_gather 状态，进行接收鼠标数据包，接收完成进入 m2_verify 状态，此时便形成了数据接收循环。

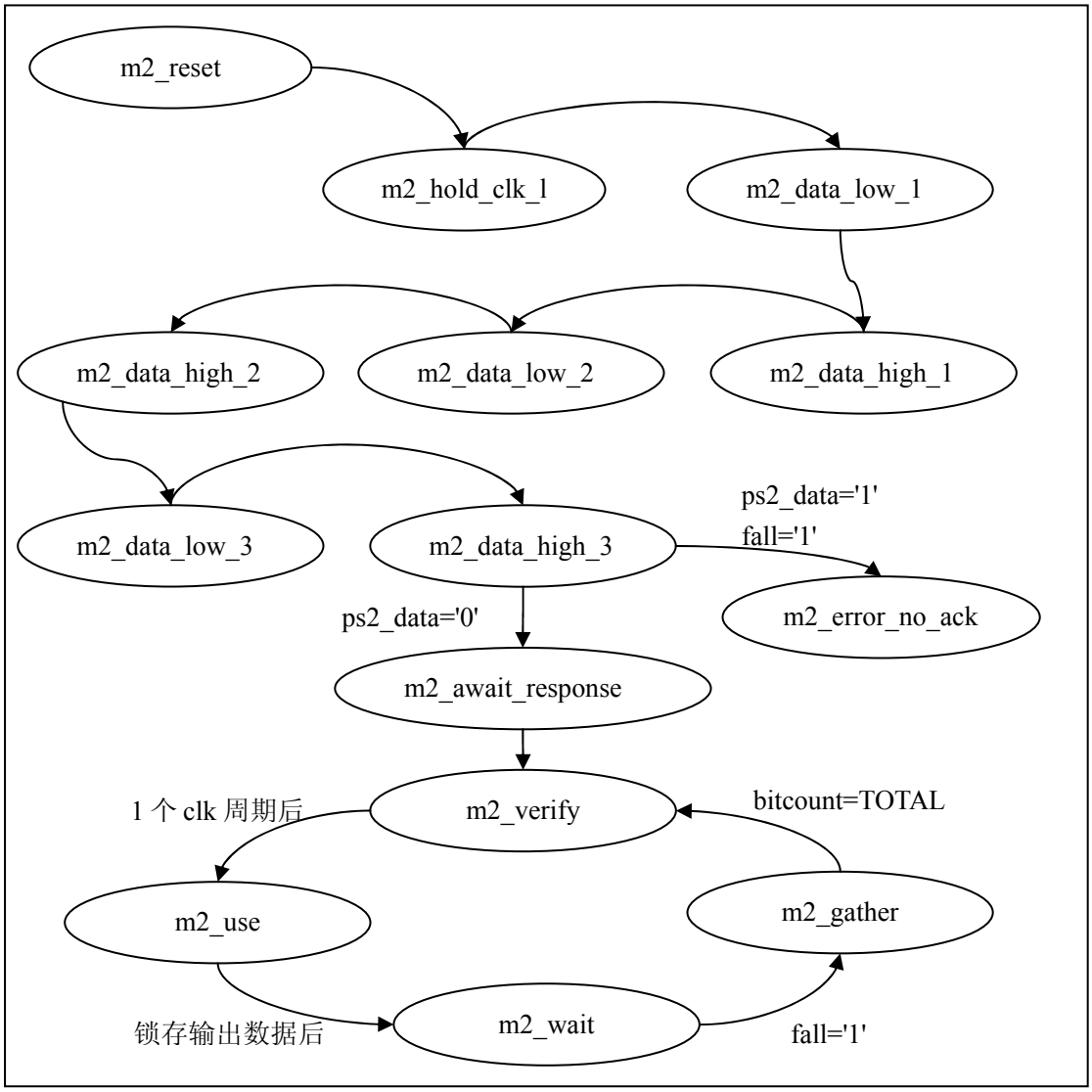


图 6 状态机

4.3. PS/2 程序源码

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mouse is
```

```
    Port (
           clk : in std_logic;
          reset : in std_logic;
          ps2_clk : inout std_logic;
          ps2_data : inout std_logic;
          left_button : out std_logic;
          right_button : out std_logic;
          mousex: buffer std_logic_vector(9 downto 0);
          mousey: buffer std_logic_vector(9 downto 0);
          data_ready : out std_logic;-- rx_read_o
          error_no_ack : out std_logic
              );
end mouse;

architecture Behavioral of mouse is

constant TOTAL_BITS : integer :=33;  -- 数据包位数

constant WATCHDOG : integer :=320;   -- 400usec 所需 sys_clk 脉冲数
--constant DEBOUNCE_TIMER : integer := 2;

--type m1statetype is ( m1_clk_h, m1_falling_edge, m1_falling_wait,
--         m1_clk_l, m1_rising_edge, m1_rising_wait);
type m2statetype is (m2_reset, m2_wait, m2_gather, m2_verify, m2_use, m2_hold_clk_l,
         m2_data_low_1, m2_data_high_1, m2_data_low_2, m2_data_high_2, m2_data_low_3,
            m2_data_high_3, m2_error_no_ack, m2_await_response);
--signal m1_state,m1_next_state : m1statetype;
signal m2_state,m2_next_state : m2statetype;
--signal m3_state,m3_next_state : std_logic;

signal watchdog_timer_done : std_logic;--命令传输超时标志
signal q : std_logic_vector(TOTAL_BITS-1 downto 0);--位序列
signal bitcount : std_logic_vector(5 downto 0);--位计数器

signal watchdog_timer_count : std_logic_vector(8 downto 0); --等待时间
--signal debounce_timer_count : std_logic_vector(1 downto 0);
signal ps2_clk_hi_z : std_logic;
signal ps2_data_hi_z : std_logic;

signal fallsig,risesig : std_logic_vector(2 downto 0);
signal clean_clk : std_logic;        -- 从 m1 跟随 ps2_clk 反向输出
signal rise,n_rise : std_logic;      -- m1 状态机输出数据
signal fall,n_fall : std_logic;      -- m1 状态机输出数据
signal output_strobe : std_logic;    -- 锁存数据到输出寄存器
signal packet_good : std_logic;      -- 检查数据是否有效
--signal x_increment : std_logic_vector(8 downto 0);
--signal y_increment : std_logic_vector(7 downto 0);
signal mouseyy : std_logic_vector(9 downto 0);

begin

ps2_clk <= '0' when ps2_clk_hi_z='0' else 'Z';
ps2_data <= '0' when ps2_data_hi_z='0' else 'Z';

  -------------------------------------
  --检测 ps2clk 上升沿和下降沿
  -------------------------------------

  detect_ps2clkfall : process(clk,reset,ps2_clk)
  begin
    if reset='0' then
      fallsig <= "000";
    elsif clk'event and clk='1' then
```

```
      fallsig(0) <= ps2_clk;
       fallsig(1) <= fallsig(0);
       fallsig(2) <= fallsig(1);
    end if;
  end process;

  fall <= '1' when fallsig="110" else '0';

  detect_ps2clkrise : process(clk,reset,ps2_clk)
  begin
    if reset='0' then
      risesig <= "000";
    elsif clk'event and clk='1' then
      risesig(0) <= ps2_clk;
       risesig(1) <= risesig(0);
       risesig(2) <= risesig(1);
    end if;
  end process;

  rise <= '1' when risesig="001" else '0';

------------------m2 状态
m2statech: process (reset, clk)
begin
  if (reset='0') then
     m2_state <= m2_reset;
  elsif (clk'event and clk='1') then
     m2_state <= m2_next_state;
  end if;
end process;

--m2 状态传输逻辑
m2statetr: process (m2_state, q, fall,rise,watchdog_timer_done,bitcount,ps2_data,packet_good)
begin
  -- 输出信号的缺省值
  ps2_clk_hi_z <= '1';
  ps2_data_hi_z <= '1';
  error_no_ack <= '0';
  output_strobe <= '0';

  case m2_state is
    when m2_reset =>    -- 复位后向鼠标发送命令字
      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_verify;
      else
           m2_next_state <= m2_gather;
      end if;
    when m2_verify =>
      --if (bitcount < TOTAL_BITS) then  --替换 " packet_good='1' "
            --m2_next_state <= m2_wait;
      --else
            m2_next_state <= m2_use;
      --end if;
    when m2_use =>
      output_strobe <= '1';
```

```
  m2_next_state <= m2_wait;

 -- 用状态机的 9 个状态实现命令字传输，使鼠标进入"streaming"模式，
 -- 并等待鼠标正确应答
when m2_hold_clk_1 =>
  ps2_clk_hi_z <= '0';   -- 启动看门狗!
  if (watchdog_timer_done='1') then
        m2_next_state <= m2_data_low_1;
  else
        m2_next_state <= m2_hold_clk_1;
  end if;
 when m2_data_low_1 =>
  ps2_data_hi_z <= '0';  -- 数据位 开始位, d[0] and d[1]
  if (fall='1' and (bitcount = 2)) then
      m2_next_state <= m2_data_high_1;
  else
        m2_next_state <= m2_data_low_1;
  end if;
 when m2_data_high_1 =>
  ps2_data_hi_z <= '1';  -- 数据位 d[2]
  if (fall='1' and (bitcount = 3)) then
      m2_next_state <= m2_data_low_2;
  else
        m2_next_state <= m2_data_high_1;
  end if;
 when m2_data_low_2 =>
  ps2_data_hi_z <= '0';  -- 数据位 d[3]
  if (fall='1' and (bitcount = 4)) then
      m2_next_state <= m2_data_high_2;
  else
         m2_next_state <= m2_data_low_2;
  end if;
 when m2_data_high_2 =>
  ps2_data_hi_z <= '1';  -- 数据位 d[4],d[5],d[6],d[7]
  if (fall='1' and (bitcount = 8)) then
      m2_next_state <= m2_data_low_3;
  else
        m2_next_state <= m2_data_high_2;
  end if;
 when m2_data_low_3 =>
  ps2_data_hi_z <= '0';  -- 奇偶校验位
  if (fall='1') then
        m2_next_state <= m2_data_high_3;
  else
        m2_next_state <= m2_data_low_3;
  end if;
 when m2_data_high_3 =>
    ps2_data_hi_z <= '1';  -- 允许鼠标拉成低电平(ACK 脉冲)
  if (fall='1' and (ps2_data='1')) then
       m2_next_state <= m2_error_no_ack;
  elsif (fall='1' and (ps2_data='0')) then
      m2_next_state <= m2_await_response;
  else
       m2_next_state <= m2_data_high_3;
  end if;
 when m2_error_no_ack =>
  error_no_ack <= '1';
  m2_next_state <= m2_error_no_ack;

 -- 为了鼠标正确进入"streaming"模式，状态极必须等待足够长的时间，
 -- 确保鼠标正确应答 0xFA。
 when m2_await_response =>
  --if (bitcount = 22) then
```

```
              m2_next_state <= m2_verify;
      --else
             -- m2_next_state <= m2_await_response;
      --end if;
    when others => m2_next_state <= m2_wait;
  end case;
end process;----------------------------m2 状态结束


--位计数器
bitcoun: process (reset, clk)
begin
  if (reset='0') then
     bitcount <= (others=>'0');  -- normal reset
  elsif (clk'event and clk='1') then
    if (fall='1') then
      bitcount <= bitcount + 1;
    elsif (watchdog_timer_done='1') then
      bitcount <= (others=>'0');  -- rx watchdog timer reset
     end if;
  end if;
end process;

-- 数据移位寄存器
dataseq: process (reset, clk)
begin
  if (reset='0') then
     q <= (others=>'0');
  elsif (clk'event and clk='1') then
    if (fall='1') then
       q <= ps2_data & q(TOTAL_BITS-1 downto 1);
     end if;
  end if;
end process;

-- 看门狗时间计数器
watchcount: process (reset,rise,fall, clk)
begin
  if ((reset='0') or (rise='1') or (fall='1')) then
    watchdog_timer_count <= (others=>'0');
  elsif (clk'event and clk='1') then
    if (watchdog_timer_done='0') then
      watchdog_timer_count <= watchdog_timer_count + 1;
     end if;
  end if;
end process;

watchdog_timer_done <= '1' when (watchdog_timer_count=WATCHDOG-1) else '0';


-- 接收数据包有效标志
packet_good <= '1';

-- 输出数据
outdata: process (reset, clk)
begin
  if (reset='0') then
    left_button <= '0';
    right_button <= '0';
    --x_increment <= (others=>'0');
    --y_increment <= (others=>'0');
  elsif (clk'event and clk='1') then
    if (output_strobe='1') then
```

```
      left_button <= q(1);
      right_button <= q(2);
      --x_increment <= '0' & q(19 downto 12);
      mouseyy <= not (q(6) & q(6) & q(30 downto 23)) + "1";
      end if;
  end if;
end process;


cordinatex: process (reset, clk)
begin
  if (reset='0') then
      mousex <= "0110010000";  -- 400
  elsif (clk'event and clk='1') then
    if (output_strobe='1') then
        if ((mousex >= 797 and q(5)='0') or (mousex <= 2 and q(5)='1')) then
          mousex <= mousex;
          else
            mousex <= mousex + (q(5) & q(5) & q(19 downto 12));--q(5):xsign q(6):ysign
          end if;
      end if;
  end if;
end process;

cordinatey: process (reset, clk)
begin
  if (reset='0') then
      mousey <= "0100101100";  -- 300
  elsif (clk'event and clk='1') then
    if (output_strobe='1') then
        if ((mousey >= 597 and q(6)='1') or (mousey <= 2 and q(6)='0')) then
          mousey <= mousey;
          else
       mousey <= mousey + mouseyy; --(q(6) & q(6) & q(30 downto 23));
          end if;
      end if;
  end if;
end process;
--mousey <= "1001010101"-mouseyy;

data_ready <= output_strobe;
end Behavioral;
```

4.4. VGA 源码

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity vgacore is
     port
     (
          reset : in std_logic;  -- reset
          clock : in std_logic;  -- VGA dot clock
          hsyncb : buffer std_logic;  -- horizontal (line) sync
          vsyncb : buffer std_logic;  -- vertical (frame) sync
       model : in std_logic_vector(2 downto 0); --background color select.
          rgb : buffer std_logic_vector(2 downto 0);-- red,green,blue colors
          read_data : in std_logic; -----the flag show data is ready to be read
          lbutton : in std_logic;
          rbutton : in std_logic;
          mousex : in std_logic_vector(9 downto 0);
          mousey : in std_logic_vector(9 downto 0)  ------mouse data
);
```

```
end vgacore;

architecture vgacore_arch of vgacore is
signal hcnt: std_logic_vector(10 downto 0);    -- horizontal pixel counter
signal vcnt: std_logic_vector(9 downto 0);     -- vertical line counter
--signal pixrg: std_logic_vector(2 downto 0);  -- byte-wide register for 4 pixels
--signal blank: std_logic;                     -- video blanking signal
signal pblank: std_logic;                      -- pipelined video blanking signal
--signal div_clk: std_logic_vector(4 downto 0);
--signal hrgb: std_logic_vector(2 downto 0); -- red,green,blue h colors
signal vrgb,hrgb,frame: std_logic_vector(2 downto 0);-- red,green,blue v colors
signal mousehcnt: std_logic_vector(9 downto 0);     -- horizontal pixel counter
signal mousevcnt: std_logic_vector(9 downto 0);     -- vertical line counter
signal bangcolor: std_logic_vector(2 downto 0);
--signal bang: std_logic;
--signal mouse_clk: std_logic;
--signal flash_clk: std_logic;

begin

A: process(clock,reset)
begin
     -- reset asynchronously clears pixel counter
     if reset='0' then
          hcnt <= "00000000000";
     -- horiz. pixel counter increments on rising edge of dot clock
     elsif (clock'event and clock='1') then
          -- horiz. pixel counter rolls-over after 381 pixels
          if hcnt<1040 then
               hcnt <= hcnt + 1;
          else
               hcnt <= "00000000000";
          end if;
     end if;
end process;

B: process(hsyncb,reset)
begin
     -- reset asynchronously clears line counter
     if reset='0' then
          vcnt <= "0000000000";
     -- vert. line counter increments after every horiz. line
     elsif (hsyncb'event and hsyncb='1') then
          -- vert. line counter rolls-over after 528 lines
          if vcnt<666 then
               vcnt <= vcnt + 1;
          else
               vcnt <= "0000000000";
          end if;
     end if;
end process;

C: process(clock,reset)
begin
     -- reset asynchronously sets horizontal sync to inactive
     if reset='0' then
          hsyncb <= '1';
     -- horizontal sync is recomputed on the rising edge of every dot clock
     elsif (clock'event and clock='1') then
          -- horiz. sync is low in this interval to signal start of a new line
          if (hcnt>=856 and hcnt<=976) then -- and hcnt<=800) then
               hsyncb <= '1';
          else
```

```
            hsyncb <= '0';
        end if;
    end if;
end process;

D: process(hsyncb,reset)
begin
    -- reset asynchronously sets vertical sync to inactive
    if reset='0' then
        vsyncb <= '1';
    -- vertical sync is recomputed at the end of every line of pixels
    elsif (hsyncb'event and hsyncb='1') then
        -- vert. sync is low in this interval to signal start of a new frame
        if (vcnt>=637 and vcnt<=643) then
            vsyncb <= '1';
        else
            vsyncb <= '0';
        end if;
    end if;
end process;

--E: blank <= '1' when ((hcnt<=16 and hcnt>=640) or (vcnt<=5 and vcnt>=480)) else '0';
--E: blank <= (hsyncb or vsyncb);

F: process(clock,reset)
begin
    if reset='0' then
        pblank <= '0';
    elsif (clock'event and clock='1') then
      if (hcnt>799 or vcnt>599) then
        pblank<='1';
      else
         pblank <= '0';
      end if;
    end if;
end process;

K: process(pblank,clock,hcnt,vcnt)
begin
 if (clock'event and clock='1') then
    if pblank='0' then
            case hcnt(7 downto 5) is
                when "000" =>   hrgb <= "100";   -- red
                when "001" =>   hrgb <= "010";   -- green
                when "010" =>   hrgb <= "001";   -- blue
                when "011" =>   hrgb <= "110";   -- red
                when "100" =>   hrgb <= "011";   -- red
                when "101" =>   hrgb <= "101";   -- green
                when "110" =>   hrgb <= "000";   -- blue
                when "111" =>   hrgb <= "111";   -- red
                when others =>   hrgb <= "000";  -- white
            end case;
            case vcnt(7 downto 5) is
                when "000" =>   vrgb <= "100";   -- red
                when "001" =>   vrgb <= "010";   -- green
                when "010" =>   vrgb <= "001";   -- blue
                when "011" =>   vrgb <= "110";   -- red
                when "100" =>   vrgb <= "011";   -- red
                when "101" =>   vrgb <= "101";   -- green
                when "110" =>   vrgb <= "000";   -- blue
                when "111" =>   vrgb <= "111";   -- red
                when others =>   vrgb <= "000";  -- white
            end case;
```

```
         else
            vrgb <= "000";
              hrgb <= "000";
         end if;
 end if;
end process;

XX: process(pblank,clock,hcnt,vcnt)
begin
 if (clock'event and clock='1') then
    if pblank='0' then
         if ((hcnt>=0 and hcnt<=9) or (hcnt>=790 and hcnt<=799) or
            (vcnt>=0 and vcnt<=9) or (vcnt>=590 and vcnt<=599)) then
           frame <= "100";
         else
            frame <= "000";
         end if;
      end if;
 end if;
end process;

L: process(model,clock)
begin
  if (clock'event and clock='1') then
   if
((hcnt>=(mousehcnt-12))and(hcnt<=mousehcnt+12)and(vcnt>=mousevcnt-2)and(vcnt<=mousevcnt+2))
or

((vcnt>=(mousevcnt-12))and(vcnt<=mousevcnt+12)and(hcnt>=mousehcnt-2)and(hcnt<=mousehcnt+2))
then
       rgb <= "111" xor bangcolor;
     else
       case model(2 downto 0) is
                  when "000" =>   rgb <= "100" xor frame;   -- red
                  when "001" =>   rgb <= "010" xor frame;   -- green
                  when "010" =>   rgb <= "001" xor frame; --blue
                  when "011" =>   rgb <= (hrgb or vrgb) xor frame; --yellow
                  when "100" =>   rgb <= (hrgb and vrgb) xor frame;  -- teal
                  when "101" =>   rgb <= (hrgb xor vrgb) xor frame; --grid
                  when "110" =>   rgb <= hrgb xor frame; --h bar
                  when "111" =>   rgb <= vrgb xor frame; --v bar
                  when others =>  rgb <="000";
     end case;
     end if;
  end if;
 end process;


MM: process(clock,reset)  -- press flash button then background color flashing.
begin
    if (clock'event and clock='1') then
    if reset='0' then
          mousehcnt <= "0110010000";  -- 400
          mousevcnt <= "0100101100";  -- 300
      else
         mousehcnt <= mousex;
          mousevcnt <= mousey;
      end if;
     end if;
end process;

P: process(clock,reset) -- chang ball color;
begin
```

```
  if (reset='0') then
      bangcolor <= "000";
  elsif (clock'event and clock='1') then
      --if (read_data='1') then
            bangcolor <= lbutton & rbutton & '1';
         --else
        --bangcolor <= "000";
         --end if;
  end if;
end process;
end vgacore_arch;
```

5. 结束语

在该设计中，采用了清华大学 THCII-1 创新 SOPC 实验套件进行综合、仿真和下载，测试得到了满意的效果，完整地实现了对 PS2 和 VGA 的时序驱动。

该设计可以被应用到各种用需要用鼠标操作、以 VGA 作为显示的嵌入式系统中，从而大大提高人机交互能力。降低了开发成本、提高了开发效率，系统的稳定性也得到了可靠的保障。