内存控制与串口通信

林锦坤 杨国炜 朱佳豪小组

实验目的

1. 熟悉THINKPAD内存储器和串口的配置及与总线的了解方式。
2. 掌握教学机内存（RAM）和串口UART的访问时序和方法。
3. 理解总线数据传输基本原理。

实验环境

硬件环境：安装有win8系统的计算机，THINKPAD教学计算机。

软件环境：FPGA开发工具软件Xilinx ISE 14.7，串口调试精灵。

实验内容

设计状态机完成通过串口调试精灵与UART的通信，从串口输入数据对存储器RAM进行读写，最后将数据发送到串口显示。具体内容如下：

1. 串口输入数据：接受串口调试精灵发送的两个16为数据，分别是地址和数据。
2. 写RAM1：将接收的数据写入到RAM1对应地址的存储单元中，按CLK键后将地址和数据各加1后写入，共写10个数据。过程中在LED灯上分别显示地址和数据的后8位。
3. 读RAM1&写RAM2：按CLK键将RAM1中的数据减1后存到RAM2中的系统位置，共进行10次。过程中在LED灯上分别显示写入地址和数据的后8位。
4. 串口输出数据：按CLK将RAM2中的数据依次发送回串口调试精灵进行显示，每次发送8位数据，共发送20次。

实验原理

1. s_Init状态
   - 将数据初始化，LED灯清空。
   - 当CLK按下时，跳转到s1状态。

2. s1状态

   共需要进行4次数据的读取，分别是地址的高低8位和数据的高低8位。将s1状态划分为4个子状态

   - 0状态：当CLK被按下将读取计数器cnt_ygw加1，跳转到1状态。
   - 1状态：将i_UART_nRE置1，接收串口数据的变量i_UART_nRE置为高阻态。跳转到2状态。
   - 2状态：检查i_UART_Ready判断UART是否准备好数据。若为1，将i_UART_nRE置0，跳转到3状态。否则跳转回1状态准备接收数据。
   - 3状态：从数据总线i_UART_nRE读取数据到对应位置，并将数据输出到LED灯显示。判断cnt_ygw是否小于4，是则跳转到0状态等待CLK。否则将i_UART_nRE置1，总状态跳转到s2状态。

3. SRAM 读取操作

   读取 SRAM 主要分为两个状态：建立状态和读取状态。

   - 建立状态：将 nWE 端口置为 1，将 nOE 端口置为 0，使得 SRAM 进入数据读取模式；把数据总线 Data 置为高阻态，将地址总线 Addr 置为所要读取地址；
   - 读取状态：在建立时间过后，数据总线 Data 上的值，即为 Addr 对应 SRAM 中内存地址的值，直接读取即可。

4. SRAM 写入操作

   写入 SRAM 与 读取 SRAM 是类似的，不同之处在于使能端值的设置。同样分为两个状态：建立状态和写入状态。

   - 建立状态：将 nWE 端口置为 1，将 nOE 端口置为 1，使得 SRAM 进入数据写入建立模式；将数据总线 Data 置为所要写入的数据，将地址总线 Addr 置为所要写入的地址；
   - 写入状态：使 nWE 跳变为 0，之后重新置为 1，此时数据 Data 已写入 SRAM 中 Addr 对应地址。

5. 写串口

   始终要保证RAM1处于高阻态。

   - 状态0：wrn置1，转状态1。
   - 状态1：wrn置0，此时UART将数据送入发送器tbr并锁存，转状态2。
   - 状态2：wrn置1，转状态3。
   - 状态3：等待tbre为1，此时被发送数据将进入移位寄存器tsr。若tbre为1转状态4。
   - 状态4：tsre变为1后转状态0。

实验代码

主工程 **main.vhd** 的代码如下：

```
1  ----------------------------------------------------------------------------
   ------------
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date:    23:42:52 11/07/2017
6  -- Design Name:
7  -- Module Name:    main - Behavioral
8  -- Project Name:
9  -- Target Devices:
```

```
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.std_logic_unsigned.all;
use work.constants.ALL;

entity main is
Port(   i_Click         : in        STD_LOGIC;
        i_Clock         : in        STD_LOGIC;
        i_nReset        : in        STD_LOGIC;
        o_Led           : out       STD_LOGIC_VECTOR (WORD_WIDTH-1 downto 0);

        o_SysBus_Addr   : out       STD_LOGIC_VECTOR (ADDR_WIDTH-1 downto 0);
        o_SysBus_Data   : inout     STD_LOGIC_VECTOR (WORD_WIDTH-1 downto 0);
        o_Ram1_nOE      : out       STD_LOGIC;
        o_Ram1_nWE      : out       STD_LOGIC;
        o_Ram1_nCE      : out       STD_LOGIC;

        o_ExtBus_Addr   : out       STD_LOGIC_VECTOR (ADDR_WIDTH-1 downto 0);
        o_ExtBus_Data   : inout     STD_LOGIC_VECTOR (WORD_WIDTH-1 downto 0);
        o_Ram2_nOE      : out       STD_LOGIC;
        o_Ram2_nWE      : out       STD_LOGIC;
        o_Ram2_nCE      : out       STD_LOGIC;

        o_Dig           : out       STD_LOGIC_VECTOR (6 downto 0);

        i_UART_Ready    : in        STD_LOGIC;
        i_UART_nRE      : out       STD_LOGIC;

        UART_nWE        : out       STD_LOGIC;
        UART_TBRE       : in        STD_LOGIC;
        UART_TSRE       : in        STD_LOGIC
        );
 end main;

```

```
architecture Behavioral of main is
    type type_State is (s_Init, s1, s2, s3, s4);
    type type_SubState is (s_Init, s_Read_Setup, s_Read_Datain,
s_Write_Setup, s_Write_Datain, s_Idle);
    type type_SubState4 is (s_40, s_41, s_42, s_43, s_44);
    signal t_State : type_State := s_Init;
    signal t_SubState : type_SubState;
    signal is_high8_ljk: std_logic := '0';
    signal stx_ljk: type_SubState4 := s_40;
    signal st_ygw : integer range 0 to 31;
    signal clock : STD_LOGIC;
    signal stage_num : std_logic_vector(3 downto 0);

    component FreqDiv is
        generic
        (
            div : integer := 50;
            half : integer := 25
        );
        port
        (
            CLK: in std_logic;
            RST: in std_logic;
            O: out std_logic
        );
    end component FreqDiv;

    component seg7 is
        port(
            code: in std_logic_vector(3 downto 0);
            seg_out : out std_logic_vector(6 downto 0)
        );
    end component seg7;
begin
     FD_Inst : FreqDiv
    generic map
    (
        div => 100000,
        half => 50000
    )
    port map
    (
        CLK => i_Clock,
        RST => '0',
        O => clock
    );

     Seg7_Inst : Seg7
     port map
     (
            code => stage_num,
            seg_out => o_Dig
```

```
107       );
108
109     process (clock, i_nReset)
110         variable t_addr : STD_LOGIC_VECTOR (ADDR_WIDTH-1 downto 0);
111         variable t_data : STD_LOGIC_VECTOR (WORD_WIDTH-1 downto 0);
112         variable r_i    : integer range 0 to 31;
113         variable l_Click : STD_LOGIC := '1';
114         variable cnt_ljk : integer range 0 to 31 := 0;
115       variable cnt_ygw : integer range 0 to 31;
116     begin
117         if i_nReset = '0' then
118           -- reset
119           t_State <= s_Init;
120         elsif falling_edge(clock) then
121           case t_State is
122             when s_Init =>
123               stage_num <= (others => '0');
124               o_Led <= (others => '0');
125               if l_Click='0' and i_Click='1' then
126                 t_State <= s1;
127                 cnt_ygw := 0;
128                 st_ygw <= 0;
129                 stx_ljk <= s_40;
130                 cnt_ljk := 0;
131                 is_high8_ljk <= '0';
132                 stage_num <= stage_num + 1;
133               end if;
134
135             when s1 =>
136               case st_ygw is
137                 when 0 =>
138                   if l_Click='0' and i_Click='1' then
139                     st_ygw <= 1;
140                     cnt_ygw := cnt_ygw+1;
141                   end if;
142                 when 1 =>
143                   st_ygw <= 2;
144                   i_UART_nRE <= '1';
145                   o_SysBus_Data <= (others => 'Z');
146                 when 2 =>
147                   if i_UART_Ready = '1' then
148                     st_ygw <= 3;
149                     i_UART_nRE <= '0';
150                   else
151                     st_ygw <= 1;
152                   end if;
153                 when 3 =>
154                   st_ygw <= 0;
155                   case cnt_ygw is
156                     when 1 =>
157                       t_addr(15 downto 8) := o_SysBus_Data(7 downto
    0);
```

```
158                       o_Led(7 downto 0) <= o_SysBus_Data(7 downto 0);
159                     when 2 =>
160                       t_addr(7 downto 0) := o_SysBus_Data(7 downto 0);
161                       o_Led(7 downto 0) <= o_SysBus_Data(7 downto 0);
162                     when 3 =>
163                       t_data(15 downto 8) := o_SysBus_Data(7 downto
    0);
164                       o_Led(7 downto 0) <= o_SysBus_Data(7 downto 0);
165                     when 4 =>
166                       t_data(7 downto 0) := o_SysBus_Data(7 downto 0);
167                       o_Led(7 downto 0) <= o_SysBus_Data(7 downto 0);
168                       stage_num <= stage_num + 1;
169                       t_State <= s2;
170                       t_SubState <= s_Init;
171                       i_UART_nRE <= '1';
172                     when others =>
173                       st_ygw <= 0;
174                   end case;
175                 when others =>
176                   st_ygw <= 0;
177             end case;
178             when s2 =>
179               case t_SubState is
180                 when s_Init => -- Begin Stage II
181                   r_i := 0;
182                   o_Ram1_nCE <= '0';
183                   t_State <= s2;
184                   t_SubState <= s_Write_Setup;
185
186                 when s_Write_Setup => -- Write to RAM1: Setup data &
    addr
187                   o_SysBus_Data <= t_data;
188                   o_SysBus_Addr <= t_addr;
189                   t_State <= s2;
190                   t_SubState <= s_Write_Datain;
191
192                 when s_Write_Datain => -- Write to RAM1: Write
193                   o_Ram1_nWE <= '0';
194                   o_Ram1_nOE <= '1';
195                   t_State <= s2;
196                   t_SubState <= s_Idle;
197
198                 when s_Idle => -- Wait for click
199                   if l_Click = '0' and i_Click = '1' then
200                     if r_i < 10 then
201                       t_data := t_data + 1;
202                       t_addr := t_addr + 1;
203                       r_i := r_i + 1;
204                       t_State <= s2;
205                       t_SubState <= s_Write_Setup;
206                     else
207                       stage_num <= stage_num + 1;
```

```
208                       t_State <= s3;
209                       t_SubState <= s_Init;
210                     end if;
211                   else
212                     o_Led(15 downto 8) <= t_addr(7 downto 0);
213                     o_Led(7 downto 0) <= t_data(7 downto 0);
214                     t_State <= s2;
215                     t_SubState <= s_Idle;
216                   end if;
217
218                 when others =>
219                   t_State <= s2;
220                   t_SubState <= s_Idle;
221
222                 end case;
223                 when s3 =>
224                   case t_SubState is
225                     when s_Init => -- Begin Stage III
226                       o_Ram1_nCE <= '0';
227                       o_Ram2_nCE <= '0';
228                       r_i := 0;
229                       t_addr := t_addr - 10;
230                       t_State <= s3;
231                       t_SubState <= s_Read_Setup;
232
233                     when s_Read_Setup => -- Read from RAM1: Setup data
    & addr
234                       o_SysBus_Data <= (others => 'Z');
235                       o_SysBus_Addr <= t_addr;
236                       o_Ram1_nWE <= '1';
237                       o_Ram1_nOE <= '0';
238                       t_State <= s3;
239                       t_SubState <= s_Read_Datain;
240
241                     when s_Read_Datain => -- Read from RAM1: Read
242                       t_data := o_SysBus_Data;
243                       t_State <= s3;
244                       t_SubState <= s_Write_Setup;
245
246                     when s_Write_Setup => -- Write to RAM2: Setup data
    & addr
247                       o_ExtBus_Data <= t_data + 1;
248                       o_ExtBus_Addr <= t_addr;
249                       t_State <= s3;
250                       t_SubState <= s_Write_Datain;
251
252                     when s_Write_Datain => -- Write to RAM2: Write
253                       o_Ram2_nWE <= '0';
254                       o_Ram2_nOE <= '1';
255                       t_State <= s3;
256                       t_SubState <= s_Idle;
257
```

```
258                     when s_Idle => -- Wait for Click
259                       if l_Click = '0' and i_Click = '1' then
260                         if r_i < 10 then
261                           t_data := t_data + 1;
262                           t_addr := t_addr + 1;
263                           r_i := r_i + 1;
264                           t_State <= s3;
265                           t_SubState <= s_Read_Setup;
266                         else
267                           stage_num <= stage_num + 1;
268                           t_addr := t_addr - 10;
269                           t_State <= s4;
270                           t_SubState <= s_Idle;
271                         end if;
272                       else
273                         o_Led(15 downto 8) <= t_addr(7 downto 0);
274                         o_Led(7 downto 0)  <= t_data(7 downto 0);
275                         t_State <= s3;
276                         t_SubState <= s_Idle;
277                       end if;
278 
279                     when others =>
280                       t_State <= s3;
281                       t_SubState <= s_Idle;
282                 end case;
283 
284                 when s4 =>
285                   case stx_ljk is
286                     when s_40 =>
287                       o_Ram1_nCE <= '1';
288                       o_Ram1_nOE <= '1';
289                       o_Ram1_nWE <= '1';
290                       o_ExtBus_Addr <=
    CONV_STD_LOGIC_VECTOR(CONV_INTEGER(t_addr) + cnt_ljk, ADDR_WIDTH);
291                       o_ExtBus_Data <= (others => 'Z');
292                       o_Ram2_nCE <= '0';
293                       o_Ram2_nOE <= '0';
294                       o_Ram2_nWE <= '1';
295                       UART_nWE <= '1';
296                       if l_Click = '0' and i_Click = '1' then
297                         stx_ljk <= s_41;
298                       else
299                         stx_ljk <= s_40;
300                       end if;
301                     when s_41 => -- loading data
302                       UART_nWE <= '0';
303                       if is_high8_ljk = '0' then o_SysBus_Data(7
    downto 0) <= o_ExtBus_Data(7 downto 0);
304                       else                       o_SysBus_Data(7
    downto 0) <= o_ExtBus_Data(15 downto 8);
305                       end if;
306                       stx_ljk <= s_42;
```

```
307                     when s_42 =>
308                       UART_nWE <= '1';
309                       stx_ljk <= s_43;
310                     when s_43 =>
311                       if UART_TBRE = '1' then
312                         stx_ljk <= s_44;
313                       else stx_ljk <= s_43;
314                       end if;
315                     when s_44 => -- sending
316                       o_Led(15 downto 8) <=
    CONV_STD_LOGIC_VECTOR(CONV_INTEGER(t_addr) + cnt_ljk, WORD_WIDTH)(7
    downto 0);
317                                 o_Led(7 downto 0) <= o_SysBus_Data(7
    downto 0);
318                       if UART_TSRE = '1' then -- sent
319                         if is_high8_ljk = '1' then cnt_ljk := cnt_ljk
    + 1;
320                         end if;
321                         is_high8_ljk <= is_high8_ljk xor '1';
322                         stx_ljk <= s_40;
323                       else -- sending
324                         stx_ljk <= s_44;
325                       end if;
326                     when others =>
327                             stx_ljk <= s_40;
328                       end case;
329                 when others =>
330                     t_State <= s_Init;
331             end case;
332             l_Click := i_Click;
333         end if;
334     end process;
335 end Behavioral;
```

时钟分频器 **FreqDiv.vhd** 的代码如下：

```vhdl

library IEEE;
use IEEE.std_logic_1164.all;

entity FreqDiv is
    generic
    (
        div : integer := 50;
        half : integer := 25
    );
    port
    (
        CLK: in std_logic;
        RST: in std_logic;
        O: out std_logic
    );
end FreqDiv;

architecture FreqDiv_bhv of FreqDiv is
    signal counter: integer range 0 to half - 1;
    signal output: std_logic;
begin
    O <= output;

    process(CLK, RST)
    begin
        if RST = '1' then
            output <= '0';
            counter <= 0;
        else
            if rising_edge(CLK) then
                if counter = half - 1 then
                    output <= not output;
                    counter <= 0;
                else
                    counter <= counter + 1;
                end if;
            end if;
        end if;
    end process;
end;
```

七段数码管显示转换器 **seg7.vhd** 的代码如下：

```vhdl
library ieee;
use ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
entity seg7 is
port(
    code: in std_logic_vector(3 downto 0);
```

```
    seg_out : out std_logic_vector(6 downto 0)
);
end seg7;

architecture behave of seg7 is

begin
process(code)
begin
    case code is
        when  "0000" =>
        seg_out<="0111111";
        when "0001" =>
        seg_out<="0000110";
        when "0010" =>
        seg_out<="1011011";
        when "0011"=>
        seg_out<="1001111";
        when "0100" =>
        seg_out<="1100110";
        when "0101" =>
        seg_out<="1101101";
        when "0110" =>
        seg_out <="1111110";
        when "0111" =>
        seg_out <="0000111";
        when "1000" =>
        seg_out <="1111111";
        when "1001" =>
        seg_out <="1011111";
        --here
        when "1010" =>
        seg_out <="1110111";
        when "1011" =>
        seg_out <="1111010";
        when "1100" =>
        seg_out <="0111100";
        when "1101" =>
        seg_out <="1101011";
        when "1110" =>
        seg_out <="1111100";
        when "1111" =>
        seg_out <="1110100";
        when others =>seg_out<="0000000";

        end case;
    end process;


 end behave;

```

实验分工

本次实验我们采取了分工合作的形式。

实验中的四个步骤分别涉及到了：

1. 读串口
2. 写ram
3. 读写ram
4. 读ram，写串口

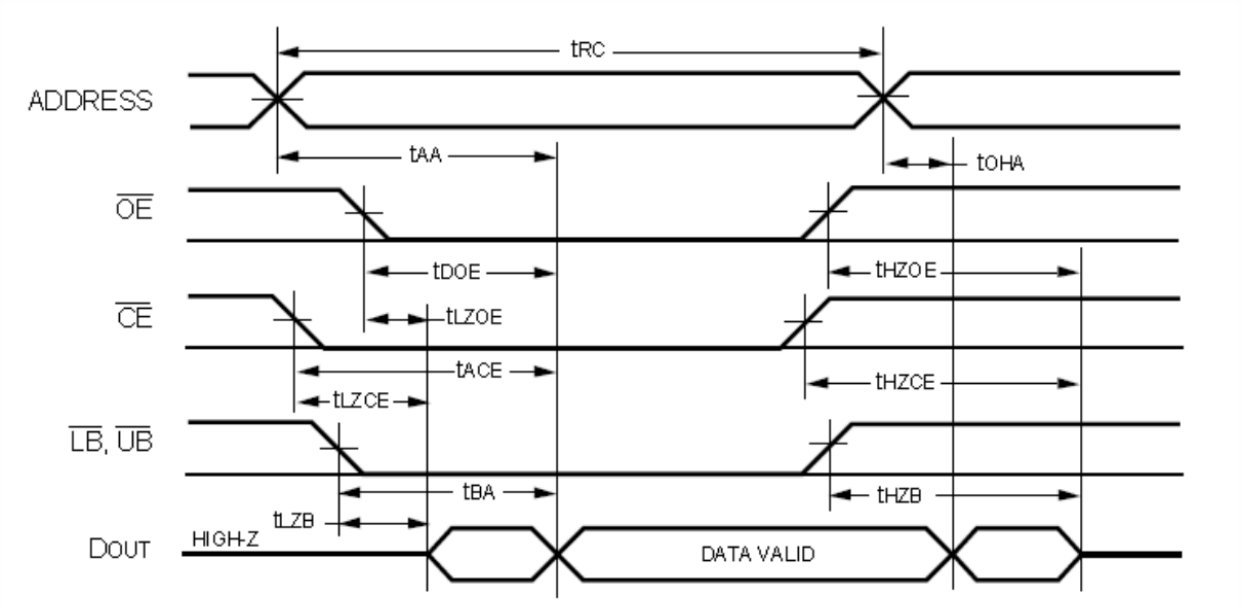
互相之间没有较大的耦合，而且设计知识点重复的也不多，因而我们将代码分为对应的4个大的状态，在一开始先写好框架，包含了全局变量和引脚，然后再细分每个状态：杨国炜负责写第一个状态的代码，朱佳豪负责第二第三，林锦坤负责最后一个状态。并且我们约定每个人的变量命名规则，保证没有冲突，方便合并。实际操作中，合并还是很顺利的。

其实最好的分工方式是元件例化，但考虑到本次实验代码量并不大，元件例化反而会增加许多不必要的代码量。

思考题

SRAM读写特点

读时序：

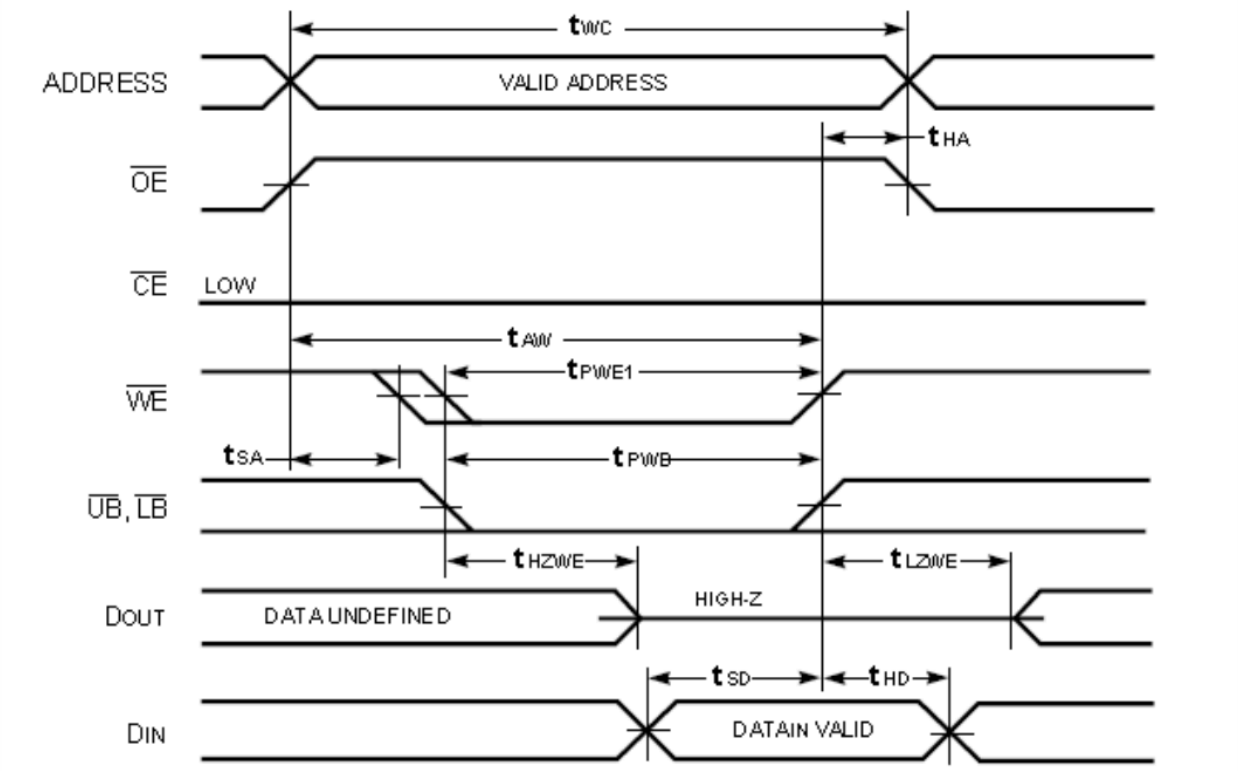


读操作需要FPGA首先同时：

1. 关闭总线上其他器件的使能端
2. 将数据总线赋为高阻态
3. 往地址总线写入地址
4. 拉低读信号

然后等待SRAM稳定后再把数据从总线读出。等待时间为tRC。查表得10ns，故读取SRAM最高频率可达100Mhz。

写 时 序 :



写操作类似，需要FPGA同时向数据和地址总线写入数据和地址，拉低需要的信号，等待SRAM稳定即可。等待时间为twc，查表得10ns，故写SRAM最高频率可达100Mhz。

什么是高阻态?

高阻态表示电路中的某个节点具有相对电路中其他点相对更高的阻抗。

作用是使RAM不驱动总线，让出总线给其他器件使用。

如何将SRAM1和SRAM2作为统一的32位的存储器使用?

将二者的控制信号CE,OE,WE,地址分别接在一起，两条数据总线并置即可。

实验心得

本次实验相比上次实验有一定难度。

首先是团队合作方面，本次实验代码较长，我们由组内三名成员合作完成。我们吸取了别组架构疏忽从而导致事倍功半的失败经验，在分工之前，先将整个任务分割成四个部分，最外层由一个主状态机控制整个任务四部分的进程，进程内由子状态机完成每部分的任务；统一了以一个闲置状态作为子状态机任务完成的标志；规定了每部分代码的变量命名规则，使得局部变量之间不会互相冲突。这使得我们的代码在整合时没有遇到困难，单元测试通过后，整个代码一遍跑通，这是值得保持的良好习惯。

其次是建立时间，由于之后的大实验中选用的时钟频率是一个重要的评分指标，助教建议我们尽可能用更快的时钟，来测试所写代码是否足够高效。我组认为这点也是十分重要的，在两个小实验中我们主要关心了代码的正确性，并没有放太多心思在提升效率上。之后的代码编写中，要注意自己所写的子进程的建立时间，找出代码瓶颈并尽力优化，才能保证结果的正确高效。