AHB-APB 数据传输设计

撰写人：杨登天-202028015926089

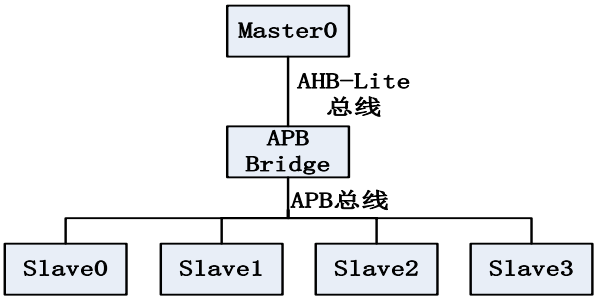
目录

AHB-APB 数据传输设计

杨登天-202028015926089-微电子研究所

一、任务设计要求

任务要求：利用 Verilog 实现如下图的系统功能并完成仿真。



如上图。主设备 Master0 利用 AHB-Lite 总线协议通过 AHB Bridge 访问四个 APB 从设备 Slave0、Slave、Slave2 和 Slave3。每个从设备的地址空间如下

Slave0: 0x0000_0000 ~ 0x0000_00ff;

Slave1: 0x0000_0100 ~ 0x0000_01ff;

Slave2: 0x0000_0200 ~ 0x0000_02ff;

Slave3: 0x0000_0300 ~ 0x0000_03ff;

令从设备地址空间的下边界为其地址的基址，假设每个从设备中有可访问 APB 寄存器 16 个，位宽均为 32 比特，16 个寄存器的访问地址计算方式为基址+寄存器编号左移 2 位（byte 偏移）。数据读取和写入分别遵照如下规范。

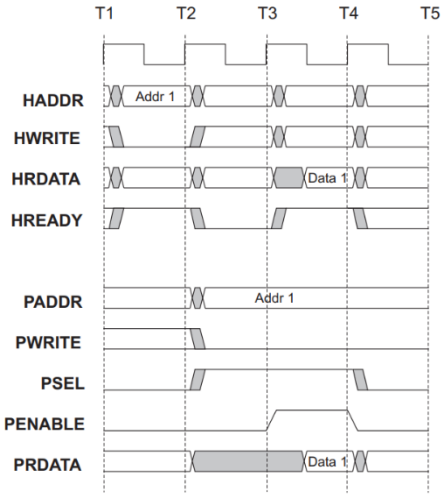


Figure 5-9 Read transfer to AHB

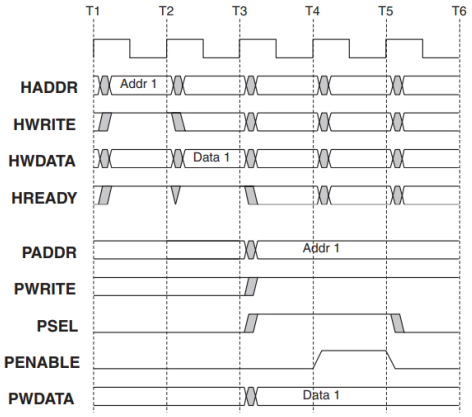


Figure 5-11 Write transfer from AHB

两图来源 ARM 官网的 AMBA 手册

注意：额外要求 PCLK 是 HCLK 的二分频时钟

二、思路分析

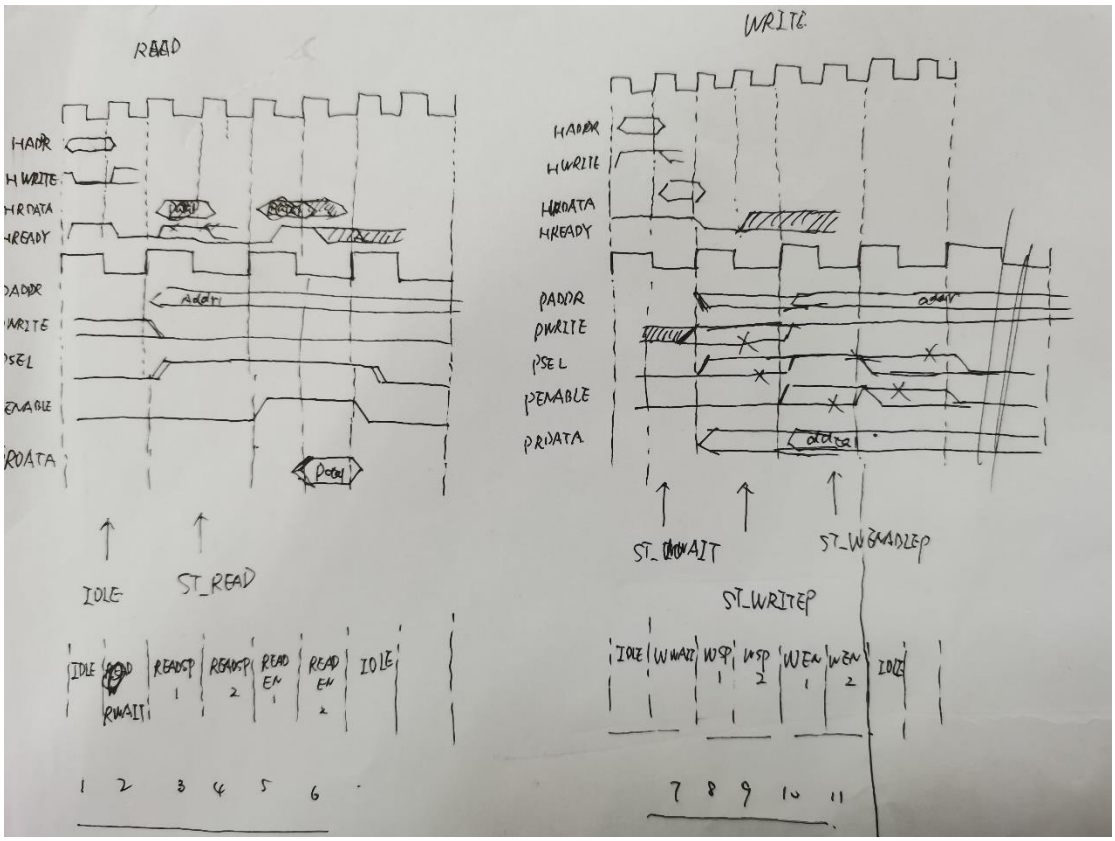
2.1、时钟生成

HCLK 采用模块时钟的二分频，PCLK 采用 HCLK 的二分频。

2.2、状态机的规定

由于前后尝试 8 次，之前 7 次均采用跨时钟域的状态机，即 APB 的三种状态机，但发

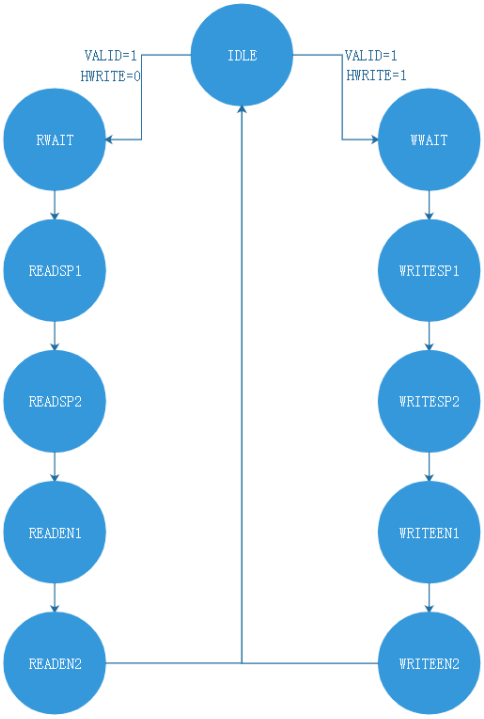
现实在无法解决问题。于是，采用 HCLK 时钟域下的状态机，并通过将读和写合并，共计 11 个状态。如下图所示。



该思路取巧，即忽略了 PCLK 的存在，将所有的状态都绑定在 HCLK 上。接下来依次规定各状态机的在程序中的编号和相关变量的取值。从读到写，上图从左至右，依次为

	Hready	Penable	Psel	Paddr	Pwrite	Hrdata	Pwdata	Prdata
IDLE(维持)	1 位 x	0	0	32 位 x	1 位 x	16 位 x	16 位 x	16 位 x
IDLE(跳转)	1	0	0	32 位 x	1 位 x	16 位 x	16 位 x	16 位 x
RWAIT	0	0	0	32 位 x	1 位 x	16 位 x		16 位 x
READSP1	0	0	1	Haddr_r2	Hwrite_r2	16 位 x		16 位 x
READSP2	0	0	1	Haddr_r3	Hwrite_r3	16 位 x		16 位 x
READEN1	0	1	1	Haddr_r4	Hwrite_r4	寄存器		寄存器
READEN2	1	1	1	Haddr_r4	Hwrite_r4			
WWAIT	1	0	0	32 位 x	1 位 x		16 位 x	
WRITESP1	0	0	1	Haddr_r2	Hwrite_r2		Hwdata_r1	
WRITESP2	1 位 x	0	1	Haddr_r3	Hwrite_r3		Hwdata_r2	
WRITEEN1	1 位 x	1	1	Haddr_r4	Hwrite_r4		Hwdata_r3	
WRITEEN2	1 位 x	1	1	Haddr_r5	Hwrite_r5		Hwdata_r4	

其状态机的变化如下图所示



其中 VALID 信号是地址有效信号和 HWRITE 有效信号传入的 wire 变量。

2.3、关于地址信号

四个 APB 从设备 Slave0、Slave、Slave2 和 Slave3。每个从设备的地址空间采用 4 个变量，即 reg [15:0] registers0 [3:0];reg [15:0] registers1 [3:0];reg [15:0] registers2 [3:0];reg [15:0] registers3 [3:0]。由地址信号的高 24 位作为判断选择何种存储器的依据，接着由地址信号的 4-7 位作为判断选择存储器内哪个寄存器单元的依据。

三、结果分析

3.1 步骤一、通过 vivado 内置仿真工具得到仿真图

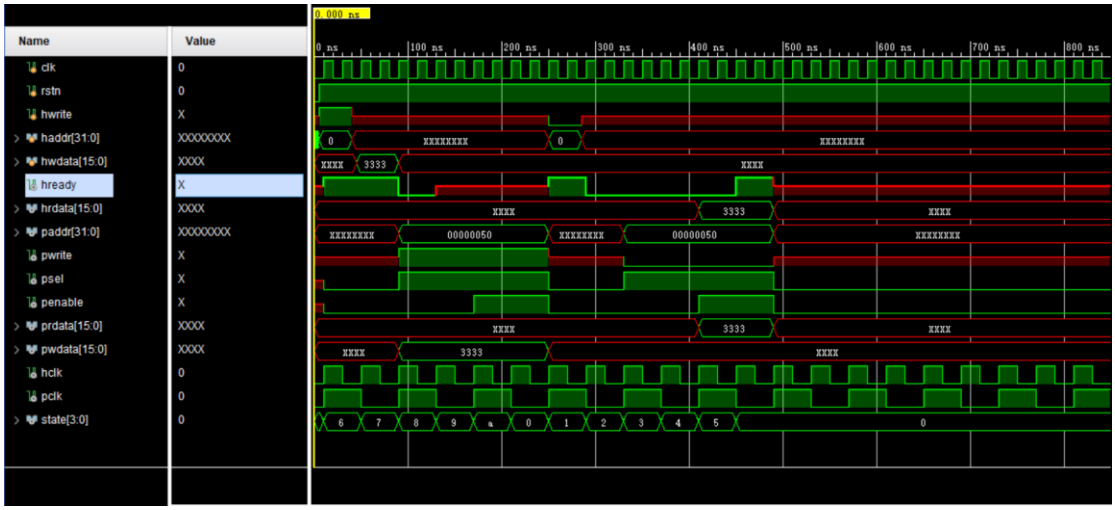


图 1 仿真图结果

能够顺利得到结果，表明程序运行无问题。

3.2 步骤二、分析仿真图内的写操作

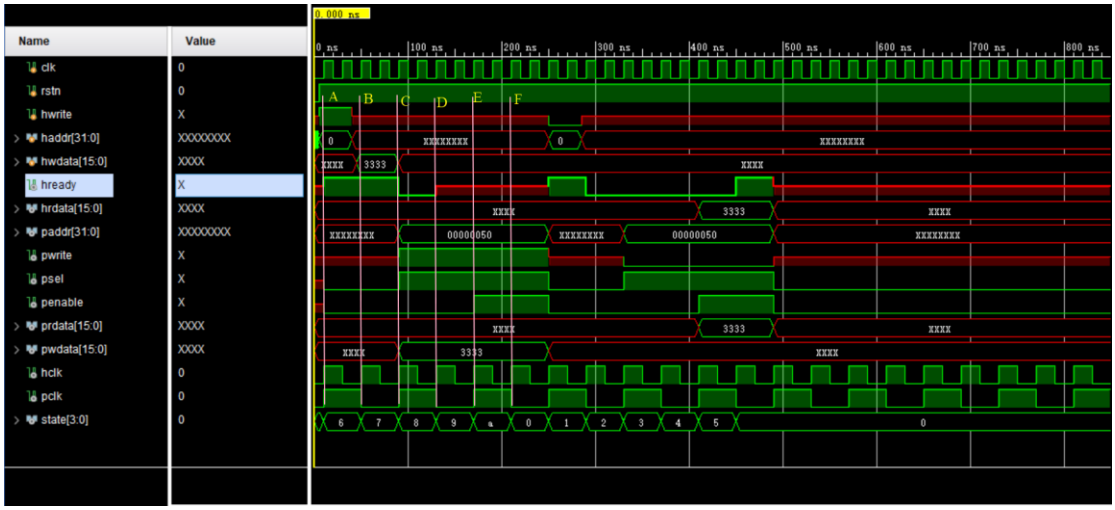


图 2 增加标志线和标志区后的仿真图

对图 2 进行分析，观察 ABCDEF 六组线的情况，可以恰好对应 ARM 写要求的 AHB 和 APB 的时序。

3.3 步骤三、分析仿真图内的读操作

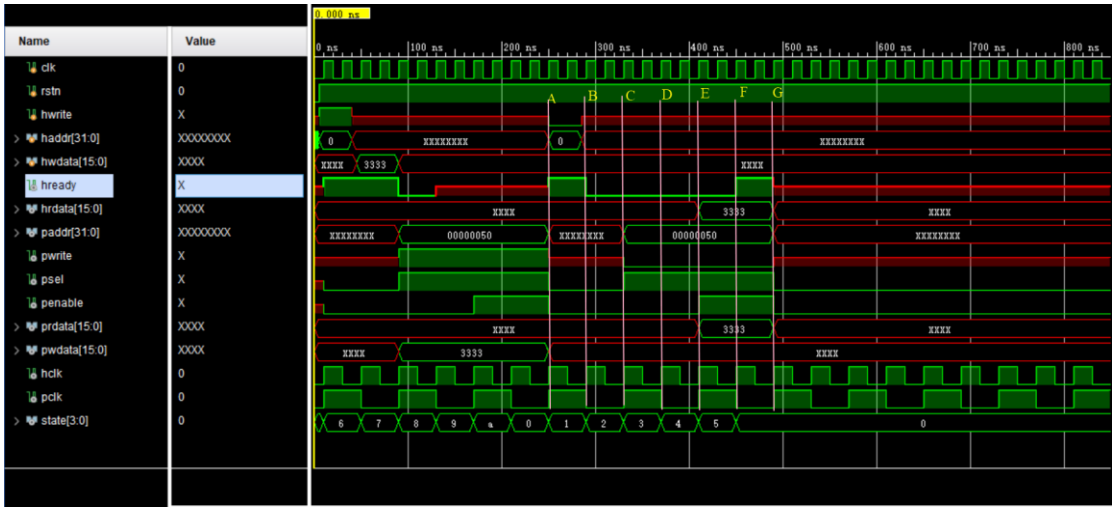


图 3 增加标志线和标志区后的仿真图

对图 3 进行分析，观察 ABCDEFG 七组线的情况，可以恰好对应 ARM 读要求的 AHB 和 APB 的时序。

3.4 步骤四、结论

在 testbench 中设定地址信号为 32'h0000_0050，并输入写数据为 16'h3333。随后继续测试写操作。见结论 4

3.4.1 结论 1——输入信号完全正确

无论是读操作还是写操作，都通过 testbench 编入相同的地址，但是 HWRITE 则视写操作和读操作要求。

3.4.2 结论 2——控制信号完全正确

无论是读操作还是写操作，根据 11 个状态分解得到的控制信号诸如 HREADY、PWRITE、PSEL、PENABLE 都符合要求。

3.4.3 结论 3——输出信号完全正确

无论是读操作还是写操作，根据 11 个状态分解得到的控制信号诸如 PWDATA、HRDATA 都符合要求。

3.4.4 结论 4——可以实现连续多写操作

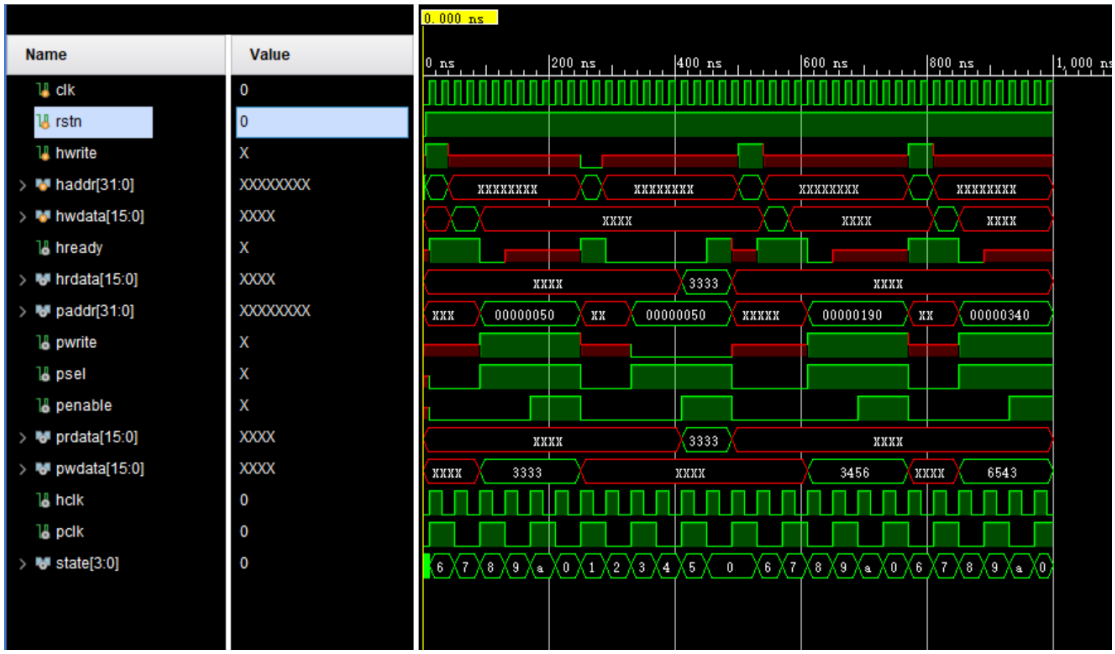


图 4 连续多写的仿真图

3.4.5 结论 5——额外测试不合法地址信号输入

设定 testbench 如下：

```
initial begin
        #500
        haddr = 32'h0000_0790; //  该地址为非法信号
        hwrite = 1;
        #40
        hwdata = 16'h3456;
        haddr = 32'hxxxx_xxxx;
        hwrite = 1'bx;
        #40
        hwdata = 16'hxxxx;
        end
```

得到如下结果——蓝框内并未出现 Psel、Penable 的变化，也不存在 Pwdata 的赋值。因此可以断定该程序对非法地址信号具有阻断作用。

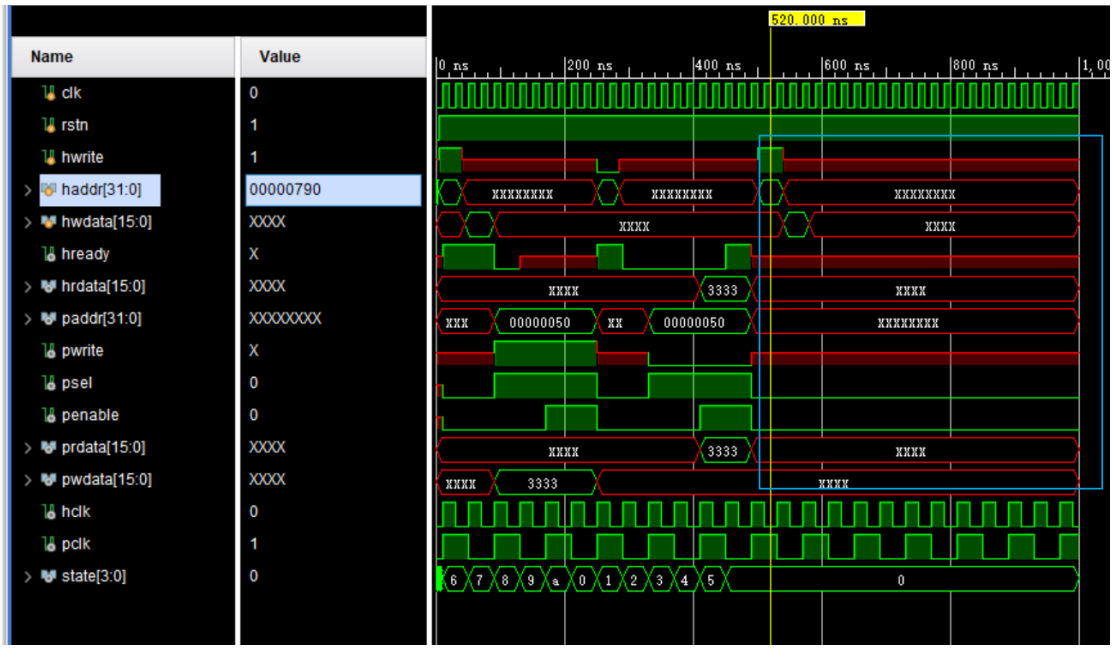


图 5 非法地址输入信号的仿真图(仅注意蓝框即可)

四、心得

这次作业难度很大，但是总的来看是完成任务的，实现了以下要求

1、 单读和单写操作;
2、 写后读操作;
3、 连续写操作;
4、 控制信号正确输出;
5、 输出信号正确;

但没有做到的地方有如下:

1、 代码未精简;
2、 读操作程序还有待提高;
3、 PCLK 未被利用，状态机仅和 HCLK 有关，这算是取巧(见第 6 点说明);
4、 关于跨异步时钟域，当前程序仅适用于时钟域无相位差，但是该程序对于时钟域有相位差则无能为力。
5、 总的来说，没有上次 IEEE754 标准的 64 位乘法器完美，后续会继续思考。
6、 关于这次作业，程序几乎重写了好几遍，主要问题还是在于跨时钟域的问题没有彻底解决，后来某次做到很绝望的时候，大晚上出去吹了吹冷风，突然想到干脆不要跨时钟域的手段串联状态机，直接将所有状态归到 HCLK 上，即将 PCLK 的一个状态分解为 HCLK 的两个状态。但跨时钟域的状态机串联从本质上讲还是没有解决。

五、附录

附录一、design code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
```

```
// Engineer:
//
// Create Date: 2020/12/25 22:09:53
// Design Name:
// Module Name: AHB2APBB
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module AHB2APBB(
  input clk,
  input rstn,
  input hwrite,
  input [31:0] haddr,
  input [15:0] hwdata,
  output reg hready,
  output reg [15:0] hrdata,

  output reg [31:0] paddr,
  output reg pwrite,
  output reg psel,
  output reg penable,
  output reg [15:0] prdata,
  output reg [15:0] pwdata,
  output reg hclk,
  output reg pclk,
  output reg [3:0] state
      );

// Pslave 相关的参数
parameter slave0 = 24'h00_0000;
parameter slave1 = 24'h00_0001;
parameter slave2 = 24'h00_0002;
parameter slave3 = 24'h00_0003;
reg [15:0] registers0 [3:0];
reg [15:0] registers1 [3:0];
```

```
reg [15:0] registers2 [3:0];
reg [15:0] registers3 [3:0];
//reg [15:0] registers0_0,registers0_1,registers0_2,registers0_3,registers0_4,registers0_5,registers0_6,registers0_7,registers0_8,registers0_9,registers0_10,registers0_11,registers0_12,registers0_13,registers0_14,registers0_15;
//reg [15:0] registers1_0,registers1_1,registers1_2,registers1_3,registers1_4,registers1_5,registers1_6,registers1_7,registers1_8,registers1_9,registers1_10,registers1_11,registers1_12,registers1_13,registers1_14,registers1_15;
//reg [15:0] registers2_0,registers2_1,registers2_2,registers2_3,registers2_4,registers2_5,registers2_6,registers2_7,registers2_8,registers2_9,registers2_10,registers2_11,registers2_12,registers2_13,registers2_14,registers2_15;
//reg [15:0] registers3_0,registers3_1,registers3_2,registers3_3,registers3_4,registers3_5,registers3_6,registers3_7,registers3_8,registers3_9,registers3_10,registers3_11,registers3_12,registers3_13,registers3_14,registers3_15;
// 计数器生成时钟
always @(posedge clk or negedge rstn) begin
   if (!rstn) hclk <= 0;
   else hclk <=    ~hclk;
   end
always @(posedge hclk or negedge rstn) begin
   if (!rstn) pclk <= 0;
   else pclk <= ~pclk;
   end

// FSM 设定,合并所有的状态
reg [3:0] next_state;
//reg [3:0] state;
parameter ST_IDLE = 4'b0000;
parameter ST_RWAIT = 4'b0001;
parameter ST_READSP1 = 4'b0010;
parameter ST_READSP2 = 4'b0011;
parameter ST_READEN1 = 4'b0100;
parameter ST_READEN2 = 4'b0101;
parameter ST_WWAIT = 4'b0110;
parameter ST_WRITESP1 = 4'b0111;
parameter ST_WRITESP2 = 4'b1000;
parameter ST_WRITEEN1 = 4'b1001;
parameter ST_WRITEEN2 = 4'b1010;
always @(posedge hclk or negedge rstn) begin
   if (!rstn) state <= ST_IDLE;
```

```
   else state <= next_state;
end
wire VALID;
wire [23:0] hap1;
wire [7:0] hap2;
assign hap1 = haddr[31:8];
assign hap2 = haddr[7:0];
assign VALID =
((hap1==24'h0000_0000||hap1==24'h0000_0001||hap1==24'h0000_0002||hap1==24'h0000
_0003)&&

(hap2==8'h00||hap2==8'h10||hap2==8'h20||hap2==8'h30||hap2==8'h40||hap2==8'h50||ha
p2==8'h60||hap2==8'h70||hap2==8'h80||

hap2==8'h90||hap2==8'ha0||hap2==8'hb0||hap2==8'hc0||hap2==8'hd0||hap2==8'he0||hap
2==8'hf0)) ? 1 : 0;
always @(state,VALID,hwrite) begin
next_state=ST_IDLE;
case(state)
ST_IDLE: begin
                if(VALID==1 & hwrite==1)
                    next_state = ST_WWAIT;                  // 进入写操作的 IDLE 阶段
                else if(VALID==1 & hwrite==0)
                    next_state = ST_RWAIT;                  // 进入读操作的 SETUP 阶段
                else
                    next_state = ST_IDLE;
               end
ST_WWAIT: next_state = ST_WRITESP1;
ST_WRITESP1: next_state = ST_WRITESP2;
ST_WRITESP2: next_state = ST_WRITEEN1;
ST_WRITEEN1: next_state = ST_WRITEEN2;
ST_WRITEEN2: next_state = ST_IDLE;
ST_RWAIT: next_state = ST_READSP1;
ST_READSP1: next_state = ST_READSP2;
ST_READSP2: next_state = ST_READEN1;
ST_READEN1: next_state = ST_READEN2;
ST_READEN2: next_state = ST_IDLE;
endcase
end

reg hreadyout_temp,penable_temp,psel_temp;
reg [31:0] paddr_temp;
reg pwrite_temp;
reg [15:0] hrdata_temp,pwdata_temp,prdata_temp;
```

```
reg [15:0] registers0_temp [3:0];
reg [15:0] registers1_temp [3:0];
reg [15:0] registers2_temp [3:0];
reg [15:0] registers3_temp [3:0];

//wire VALID2;
wire [23:0] hap1_2;
wire [7:0] hap2_2;
assign hap1_2 = haddr_reg2[31:8];
assign hap2_2 = haddr_reg2[7:0];
assign VALID2 =
((hap1_2==24'h0000_0000||hap1_2==24'h0000_0001||hap1_2==24'h0000_0002||hap1_2==
24'h0000_0003)&&

(hap2_2==8'h00||hap2_2==8'h10||hap2_2==8'h20||hap2_2==8'h30||hap2_2==8'h40||hap2_
2==8'h50||hap2_2==8'h60||hap2_2==8'h70||hap2_2==8'h80||

hap2_2==8'h90||hap2_2==8'ha0||hap2_2==8'hb0||hap2_2==8'hc0||hap2_2==8'hd0||hap2_2
==8'he0||hap2_2==8'hf0)) ? 1 : 0;

//wire VALID3;
wire [23:0] hap1_4;
wire [7:0] hap2_4;
assign hap1_4 = haddr_reg4[31:8];
assign hap2_4 = haddr_reg4[7:0];
assign VALID3 =
((hap1_4==24'h0000_0000||hap1_4==24'h0000_0001||hap1_4==24'h0000_0002||hap1_4==
24'h0000_0003)&&

(hap2_4==8'h00||hap2_4==8'h10||hap2_4==8'h20||hap2_4==8'h30||hap2_4==8'h40||hap2_
4==8'h50||hap2_4==8'h60||hap2_4==8'h70||hap2_4==8'h80||

hap2_4==8'h90||hap2_4==8'ha0||hap2_4==8'hb0||hap2_4==8'hc0||hap2_4==8'hd0||hap2_4
==8'he0||hap2_4==8'hf0)) ? 1 : 0;


//延迟信号
reg [31:0] haddr_reg1,haddr_reg2,haddr_reg3,haddr_reg4,haddr_reg5;
reg hwrite_reg1,hwrite_reg2,hwrite_reg3,hwrite_reg4,hwrite_reg5;
//reg [15:0] hwdata_reg2,hwdata_reg3,hwdata_reg4,hwdata_reg5;
reg [15:0] hwdata_reg1,hwdata_reg2,hwdata_reg3,hwdata_reg4,hwdata_reg5;
wire [3:0] Pregister1,Pregister2,Pregister3,Pregister4;
//wire [3:0] Pregister2,Pregister4;
wire [23:0] Pslave1,Pslave2,Pslave3,Pslave4;
```

```
//wire [23:0] Pslave2,Pslave4;
assign Pregister1 = haddr_reg2[7:4];
assign Pregister2 = haddr_reg3[7:4];
assign Pregister3 = haddr_reg4[7:4];
assign Pregister4 = haddr_reg5[7:4];
assign Pslave1 = haddr_reg2[31:8];
assign Pslave2 = haddr_reg3[31:8];
assign Pslave3 = haddr_reg4[31:8];
assign Pslave4 = haddr_reg5[31:8];

always @(posedge hclk or negedge rstn) begin
    if (!rstn) begin
        haddr_reg1 <= 32'hxxxx_xxxx;haddr_reg2 <= 32'hxxxx_xxxx;haddr_reg3 <= 32'hxxxx_xxxx;haddr_reg4 <= 32'hxxxx_xxxx;haddr_reg5 <= 32'hxxxx_xxxx;
        hwrite_reg1 <= 1'bx;hwrite_reg2 <= 1'bx;hwrite_reg3 <= 1'bx;hwrite_reg4 <= 1'bx;hwrite_reg5 <= 1'bx;
        hwdata_reg1 <= 16'hxxxx;hwdata_reg2 <= 16'hxxxx;hwdata_reg3 <= 16'hxxxx;hwdata_reg4 <= 16'hxxxx;hwdata_reg5 <= 16'hxxxx;
        end
    else begin
        haddr_reg1 <= haddr;haddr_reg2 <= haddr_reg1;haddr_reg3 <= haddr_reg2;haddr_reg4 <= haddr_reg3;haddr_reg5 <= haddr_reg4;
        hwrite_reg1 <= hwrite;hwrite_reg2 <= hwrite_reg1;hwrite_reg3 <= hwrite_reg2;hwrite_reg4 <= hwrite_reg3;hwrite_reg5 <= hwrite_reg4;
        hwdata_reg1 <= hwdata;hwdata_reg2 <= hwdata_reg1;hwdata_reg3 <= hwdata_reg2;hwdata_reg4 <= hwdata_reg3;hwdata_reg5 <= hwdata_reg4;
        end
end

always@(*)
begin
//{hreadyout_temp, penable_temp, pwrite_temp,  pselx_temp, paddr_temp, pwdata_temp, hrdata_temp} = 0;
 case(state)
ST_IDLE: begin
            if(VALID==1) begin
                hreadyout_temp = 1;
                penable_temp = 0;
                psel_temp = 0;
                paddr_temp = 32'hxxxx_xxxx;
                pwrite_temp = 1'bx;
                hrdata_temp = 16'hxxxx;
                pwdata_temp = 16'hxxxx;
                prdata_temp = 16'hxxxx;
```

```
                end
            else begin
                hreadyout_temp = 1'bx;
                penable_temp = 0;
                psel_temp = 0;
                paddr_temp = 32'hxxxx_xxxx;
                pwrite_temp = 1'bx;
                hrdata_temp = 16'hxxxx;
                pwdata_temp = 16'hxxxx;
                prdata_temp = 16'hxxxx;
                end
            end
ST_WWAIT: begin
                hreadyout_temp = 1;
                penable_temp = 0;
                psel_temp = 0;
                paddr_temp = 32'hxxxx_xxxx;
                pwrite_temp = 1'bx;
                pwdata_temp = 16'hxxxx;
            end
ST_WRITESP1: begin
                hreadyout_temp = 0;
                penable_temp = 0;
                psel_temp = 1;
                paddr_temp = haddr_reg2;
                pwrite_temp = hwrite_reg2;
                pwdata_temp = hwdata_reg1;
                // 存储到寄存器内
                case({Pslave1,VALID2})
                  {slave0,1}: registers0_temp[Pregister1] = hwdata_reg1;
                  {slave1,1}: registers1_temp[Pregister1] = hwdata_reg1;
                  {slave2,1}: registers2_temp[Pregister1] = hwdata_reg1;
                  {slave3,1}: registers3_temp[Pregister1] = hwdata_reg1;
                endcase
              end
ST_WRITESP2: begin
                hreadyout_temp = 1'bx;
                penable_temp = 0;
                psel_temp = 1;
                paddr_temp = haddr_reg3;
                pwrite_temp = hwrite_reg3;
                pwdata_temp = hwdata_reg2;
              end
ST_WRITEEN1: begin
```

```
                    hreadyout_temp = 1'bx;
                    penable_temp = 1;
                    psel_temp = 1;
                    paddr_temp = haddr_reg4;
                    pwrite_temp = hwrite_reg4;
                    pwdata_temp = hwdata_reg3;
                  end
ST_WRITEEN2: begin
                    hreadyout_temp = 1'bx;
                    penable_temp = 1;
                    psel_temp = 1;
                    paddr_temp = haddr_reg5;
                    pwrite_temp = hwrite_reg5;
                    pwdata_temp = hwdata_reg4;
                  end
ST_RWAIT: begin
                    hreadyout_temp = 0;
                    penable_temp = 0;
                    psel_temp = 0;
                    paddr_temp = 32'hxxxx_xxxx;
                    pwrite_temp = 1'bx;
                    hrdata_temp = 16'hxxxx;
                    prdata_temp = 16'hxxxx;
                  end
ST_READSP1:begin
                    hreadyout_temp = 0;
                    penable_temp = 0;
                    psel_temp = 1;
                    paddr_temp = haddr_reg2;
                    pwrite_temp = hwrite_reg2;
                    hrdata_temp = 16'hxxxx;
                    prdata_temp = 16'hxxxx;
                 end
ST_READSP2:begin
                    hreadyout_temp = 0;
                    penable_temp = 0;
                    psel_temp = 1;
                    paddr_temp = haddr_reg3;
                    pwrite_temp = hwrite_reg3;
                    hrdata_temp = 16'hxxxx;
                    prdata_temp = 16'hxxxx;
                 end
ST_READEN1:begin
                    hreadyout_temp = 0;
```

```
                penable_temp = 1;
                psel_temp = 1;
                paddr_temp = haddr_reg4;
                pwrite_temp = hwrite_reg4;
               case({Pslave3,VALID3})
                 {slave0,1}: begin hrdata_temp = registers0_temp[Pregister3];prdata_temp =
registers0_temp[Pregister3]; end
                 {slave1,1}: begin hrdata_temp = registers1_temp[Pregister3];prdata_temp =
registers1_temp[Pregister3]; end
                 {slave2,1}: begin hrdata_temp = registers2_temp[Pregister3];prdata_temp =
registers2_temp[Pregister3]; end
                 {slave3,1}: begin hrdata_temp = registers3_temp[Pregister3];prdata_temp =
registers3_temp[Pregister3]; end
                endcase
              end
ST_READEN2:begin
                hreadyout_temp = 1;
                penable_temp = 1;
                psel_temp = 1;
                paddr_temp = haddr_reg5;
                pwrite_temp = hwrite_reg5;
              end
endcase
end

always@(posedge hclk or negedge rstn)
begin
if (!rstn) begin
penable <= 1'bx;
pwrite <= 1'bx;
psel <= 1'bx;
paddr <= 32'hxxxx_xxxx;
pwdata <= 16'hxxxx;
hrdata <= 16'hxxxx;
hready <= 1'bx;
prdata <= 16'hxxxx;
end
else begin
penable <= penable_temp;
pwrite <= pwrite_temp;
psel <= psel_temp;
paddr <= paddr_temp;
pwdata <= pwdata_temp;
hrdata <= hrdata_temp;
```

```
hready <= hreadyout_temp;
prdata <= prdata_temp;
end
end

endmodule
```

附录二、test code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2020/12/28 19:32:37
// Design Name:
// Module Name: APBAHBtb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//缺一个锁


module APBAHBtb();
  reg clk;
  reg rstn;
  reg hwrite;
  reg [31:0] haddr;
  reg [15:0] hwdata;
  wire hready;
  wire [15:0] hrdata;

  wire [31:0] paddr;
  wire pwrite;
  wire psel;
  wire penable;
  wire [15:0] prdata;
```

```
  wire [15:0] pwdata;
  wire hclk;
  wire pclk;
  wire [3:0] state;
  //wire [3:0] next_state;

  AHB2APBB
myAHB2APB(clk,rstn,hwrite,haddr,hwdata,hready,hrdata,paddr,pwrite,psel,penable,prdata,p
wdata,hclk,pclk,state);

  always #10 clk = ~clk;
  initial begin
          clk = 0;
          rstn = 0;
          #5
          rstn = 1;
          haddr = 32'h0000_0050;     //32'b0000_0000_0000_0000_ _0000_0000_0101_1111
          hwrite = 1;   //写高位
          #40
          hwdata = 16'b0011_0011_0011_0011;
          end
  initial begin
         #40
         haddr = 32'hxxxx_xxxx;
         hwrite = 1'bx;
         #50
         hwdata = 16'hxxxx;
         end
  initial begin
         #250
         haddr = 32'h0000_0050;
         hwrite = 0;
         #35
         haddr = 32'hxxxx_xxxx;
         hwrite = 1'bx;
         end
  initial begin
         #500
         haddr = 32'h0000_0790;
         hwrite = 1;
         #40
         hwdata = 16'h3456;
         haddr = 32'hxxxx_xxxx;
         hwrite = 1'bx;
```

```
        #40
        hwdata = 16'hxxxx;
        end
endmodule
```