



哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

(深圳)

实验作业

开课学期: 2021 春季

课程名称: 计算机组成原理 (实验)

实验名称: TileLink 总线协议设计

实验性质: 综合设计型

实验学时: 4 地点: T2612

学生班级: 1901105

学生学号: 190110509

学生姓名: 王铭

作业成绩: _____

实验与创新实践教育中心制

2021 年 4 月

一、 Master 模块添加注释

```

module master (
    input  wire      clk      ,
    input  wire      rst_n    ,
    input  wire      cpu_wr   ,//cpu 写使能
    input  wire      cpu_rd   ,//cpu 读使能
    input  wire [3:0]  cpu_byte ,//选择写入的数据的有效部分
    input  wire [3:0]  cpu_addr ,//写入的地址
    input  wire [31:0] cpu_wdata ,//写入的数据
    output wire        cpu_rdata_v, //读数据有效的标志
    output wire [31:0]  cpu_rdata ,//读出的数据
    input  wire        a_ready  ,//a 通道可以进行传输
    output reg         a_valid  , //a 通道传输信号有效
    output reg [3:0]   a_opcode ,//进行的操作
    output reg [3:0]   a_mask   ,//选择写入数据的有效部分
    output reg [3:0]   a_address ,//写入数据的地址
    output reg [31:0]  a_data   ,//写入的数据
    output reg         d_ready  ,//d 通道可以进行传输
    input  wire        d_valid  ,//d 通道传输的数据有效
    input  wire [3:0]  d_opcode  ,//d 信号给的反馈
    input  wire [31:0] d_data   ,//读出的数据
    output reg         trans_over //读/写是否结束
);
always @ (posedge clk or negedge rst_n) begin
    if (~rst_n)          d_ready <= 1'b0;
    else if (cpu_wr | cpu_rd) d_ready <= 1'b1; //读或写使能时，d 通道可以进行传输
end

always @ (posedge clk or negedge rst_n) begin
    if (~rst_n)          a_valid <= 1'b0;
    else if (cpu_wr | cpu_rd) a_valid <= 1'b1; //读或写使能有效时，a 通道信号有效
    else                  a_valid <= 1'b0;
end

always @ (posedge clk or negedge rst_n) begin
    if (~rst_n)          a_opcode <= 4'h0;
    else if (cpu_wr & (&cpu_byte)) a_opcode <= 4'h0; //部分写数据
    else if (cpu_wr)       a_opcode <= 4'h1; //全部写数据
    else if (cpu_rd)       a_opcode <= 4'h4; //读数据
    else                   a_opcode <= 4'h0;
end

always @ (posedge clk or negedge rst_n) begin

```

```

        if (~rst_n)            a_mask <= 4'h0;
        else if (cpu_wr | cpu_rd) a_mask <= cpu_byte; //选择数据有效字节位
        else                    a_mask <= 4'h0;
    end

    always @ (posedge clk or negedge rst_n) begin
        if (~rst_n)            a_address <= 4'h0;
        else if (cpu_wr | cpu_rd) a_address <= cpu_addr; //选择写或读数据的地址
        else                    a_address <= 4'h0;
    end

    always @ (posedge clk or negedge rst_n) begin
        if (~rst_n)            a_data <= 32'h0;
        else if (cpu_wr) a_data <= cpu_wdata; //写使能有效时，a 传输的数据为
cpu_wdata
        else                    a_data <= 32'h0;
    end

    reg rd_period; //表示是否正在读阶段，为 1 表示在读
    reg trans_over_ff; //便于采集读/写过程结束信号的上升沿

    always @ (posedge clk or negedge rst_n) begin
        if (~rst_n) trans_over_ff <= 1'b0;
        else        trans_over_ff <= trans_over; //较 trans_over 滞后一个时钟，使得
trans_over_pos 能采 trans_over 上升沿时也置 1.
    end
    wire trans_over_pos = trans_over & ~trans_over_ff; //当刚好结束读/写时的第一个时
钟周期内有效。

    always @ (posedge clk or negedge rst_n) begin
        if (~rst_n)            rd_period <= 1'b0;
        else if (trans_over_pos) rd_period <= 1'b0; //读取过程结束，置 0
        else if (cpu_rd)        rd_period <= 1'b1; //读取过程未结束，置 1
    end

    assign cpu_rdata_v = rd_period & d_valid; //正在读且读出数据有效表示读完，给出
读出信号有效的反馈
    assign cpu_rdata = d_data;
    always @ (posedge clk or negedge rst_n) begin
        if (~rst_n)            trans_over <= 1'b1;
        else if (a_ready & a_valid) trans_over <= 1'b0; //开始写/读操作
        else if (d_ready & d_valid) trans_over <= 1'b1; //读或写操作结束
    end
endmodule

```

二、 Slave 模块的实现

Slave 模块的实现:

在复位信号无效的条件下, reg_addr,reg_byte,reg_wdata,d_data 赋值成传入 slave 的 a_address,a_mask,a_data,reg_rdata 即可。

在复位信号无效的条件下, 根据 a_valid 信号以及 opcode 的值确定 reg_wr,reg_rd 是否置 1。

d_valid 信号在写操作时接受到 a_valid 和 a_opcode 的有效信号时置 1。

读操作时, 设置计数器 rd_cnt, 接受到读信号时置为 2'b01,此时钟周期向 cordic 模块传递地址, 第二周期置为 2'b10,此时在 cordic 模块中读取数据,根据 cnt 是否是 2'b10 确定 d_valid 是否置 1。

d_opcode 与 d_valid 同步变化, 若进行了读操作, d_opcode 应置 1, 否则置 0。

部分关键代码:

```
always@(posedge clk or negedge rst_n)

begin

    if(~rst_n) reg_wr <= 1'b0;

    else if(a_valid && (a_opcode == 4'b0000 || a_opcode == 4'b0001)) reg_wr <=
1'b1;//根据 opcode 为 0/1 置 reg_wr 为 1

    else reg_wr <= 1'b0;

end

always@(posedge clk or negedge rst_n)

begin

    if(~rst_n) reg_rd <= 1'b0;

    else if(a_valid && a_opcode == 4'b0100) reg_rd <= 1'b1;//根据 a_opcode 为 4 置
reg_rd 为 1

    else reg_rd <= 1'b0;

end
```

```
reg [1:0]rd_cnt;//控制读过程

always@(posedge clk or negedge rst_n)

begin

    if(~rst_n) rd_cnt <= 'b0;

    else if(a_valid && a_opcode == 'b0100) rd_cnt <= 'b01;

    else if(rd_cnt == 'b01) rd_cnt <= 'b10;

    else rd_cnt <= 'b0;

end

always@(posedge clk or negedge rst_n)

begin

    if(~rst_n) d_valid <= 1'b0;

    else if(a_valid && (a_opcode == 'b0001 || a_opcode == 'b0000)) d_valid <= 1'b1;

    else if(rd_cnt == 'b10) d_valid <= 1'b1;//读需要三个周期才能输出信号有效

    else d_valid <= 1'b0;

end

always@(posedge clk or negedge rst_n)

begin

    if(~rst_n) d_opcode <= 4'b0000;

    else if(a_valid && (a_opcode == 4'b0001 || a_opcode == 'b0000)) d_opcode <=
4'b0000;//写操作置 0

    else if(rd_cnt == 'b10) d_opcode <= 'b0001;//读操作置 1

    else d_opcode <= 4'b0000;

end
```

三、 调试报告

仿真核心代码:

```
always #5 clk = ~clk; //时钟周期为 10

initial

begin

    #25 rst_n = 1;

    //全写 0x00ab_00ab 并读

    #5 begin byte = 4'b1111;  wdata = 32'h00ab_00ab;  addr=4'b0001; wr = 1; end

    #10 wr = 0;

    #50 begin byte = 4'b1111; addr = 4'b0001;rd = 1;end

    #10 begin rd = 0;end

    //写 0x0000_0303 后两个字节并读

    #50 begin  wdata = 32'h0000_0303; addr = 4'b0001; byte = 4'b0011;  wr = 1;

end

    #10 wr = 0;

    #50 begin addr = 4'b0001;byte = 4'b1111; rd = 1; end

    #10 rd = 0;

    #50 rst_n = 0;

    #10 rst_n = 1;

    //sin 是 0 cos 是 1 phase 是 0x1 result 为 0x2

    //sin,0x1000a,0xfc1b

    # 50 begin wdata = 32'h0001_000a;addr = 4'b0001;byte = 4'b1111;wr = 1; end

    # 10 wr = 0;

    # 50 begin wdata = (1 | (0 << 8));addr = 4'b0000;byte = 4'b1111;wr = 1;end

    # 10 wr = 0;

    # 200 begin addr = 4'b0010;byte = 4'b1111;rd = 1;end
```

```

# 10 rd = 0;

//cos,0x10014,0xffffa872

# 50 begin wdata = 32'h0001_0014;addr = 4'b0001;byte = 4'b1111;wr = 1; end

# 10 wr = 0;

# 50 begin wdata = (1 | (1 << 8));addr = 4'b0000;byte = 4'b1111;wr = 1;end

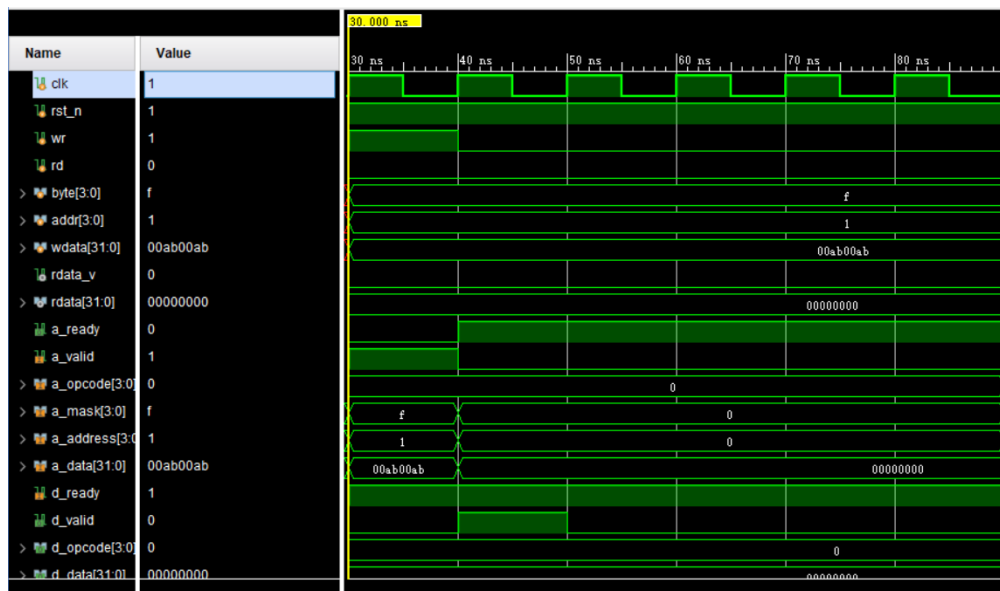
# 10 wr = 0;

# 200 begin addr = 4'b0010;byte = 4'b1111;rd = 1;end

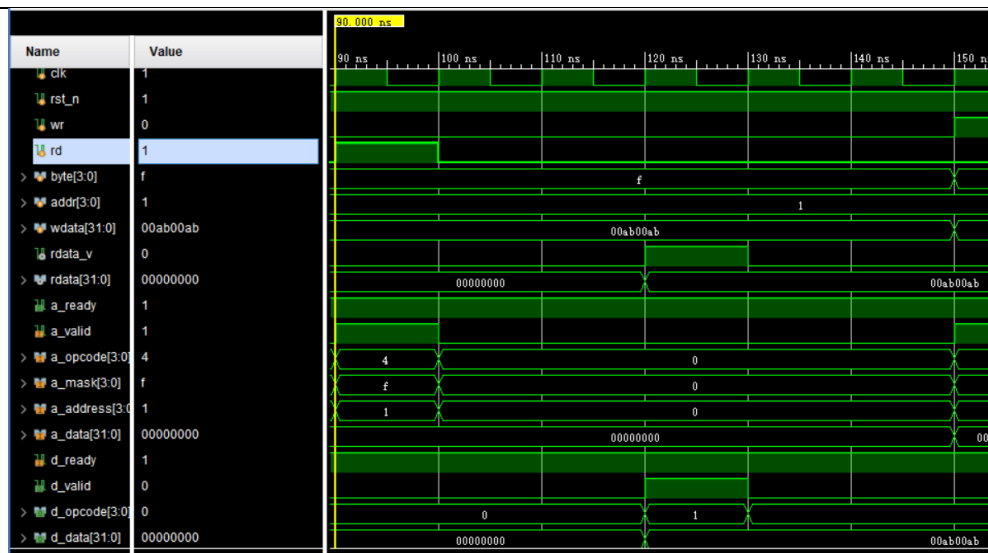
# 10 rd = 0;

```

end



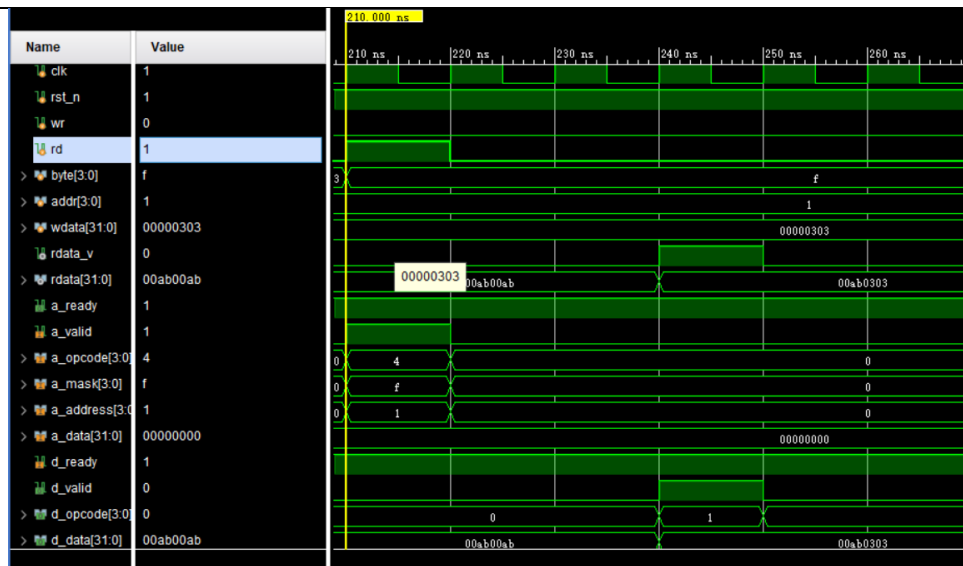
时序分析：30ns 进行第一次写操作，向 addr=0x1 全写数据 0x00ab_00ab，第一个时钟周期 a_valid 和 d_ready 置 1，第二个时钟周期 a_ready 和 d_valid 置 1，写操作结束,d_opcode 为 0。



时序分析：90ns 时进行对 0x1 进行读操作，在第二个周期向 cordic 模块传输地址和读使能，第三个周期读出数据，0x00ab_00ab，并置 d_valid 和 rdata_v 为 1，表示读出的数据有效，d_opcode 置 1。



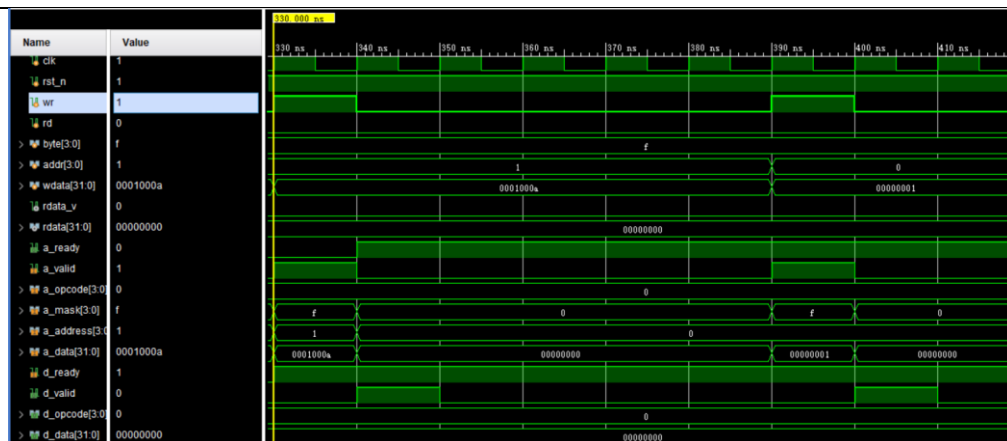
时序分析：150ns 时对 0x1 进行部分写操作，byte 为 4'b0011 表示写入 0x0000_0303 的后两个字节即 0303。第二个周期 d_valid 置 1，d_opcode 为 0



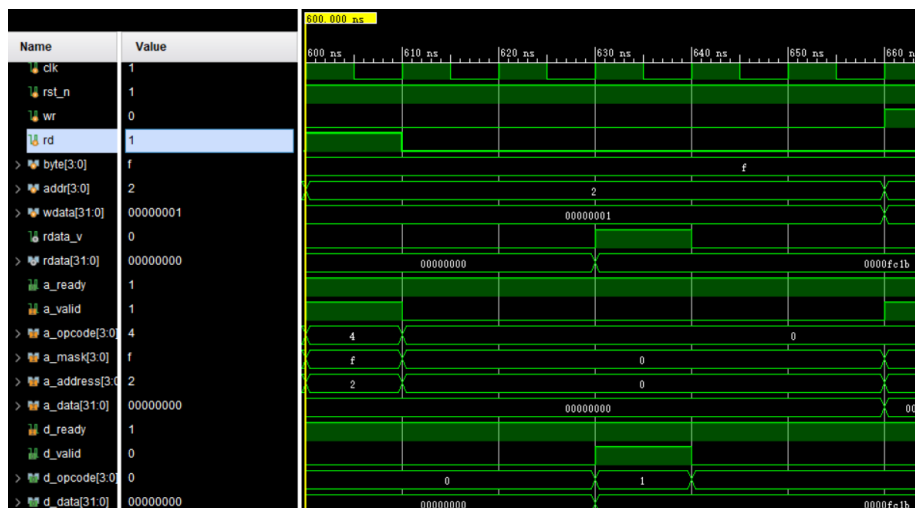
时序分析：210ns 对 0x1 读取数据，第二个周期时向 cordic 传递地址，第三个周期读取数据为 0x00ab_0303，置 d_valid 和 rdata_v 为 1 表示读出数据有效，d_opcode 给出高电平反馈信号。



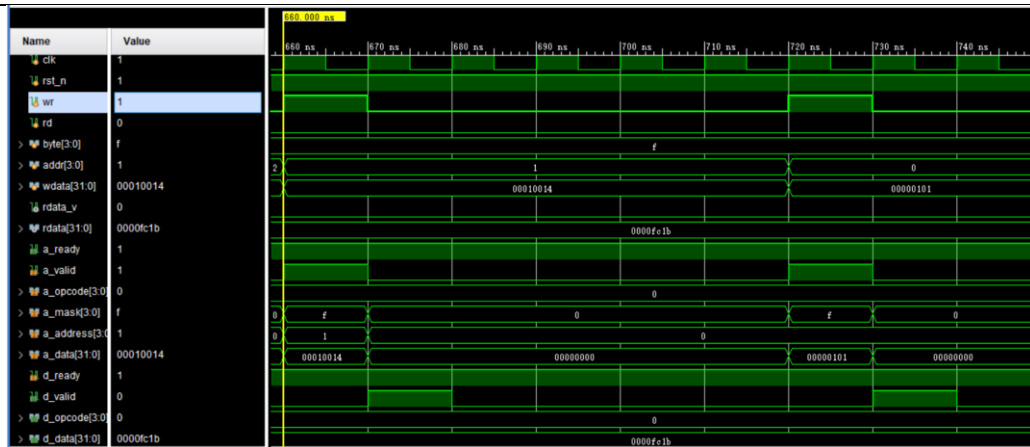
测试复位功能，d_data 恢复 0



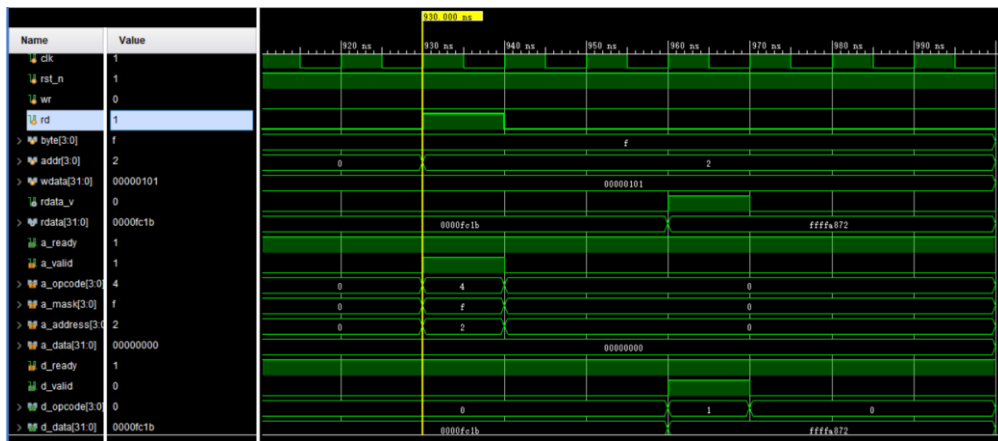
时序分析: 330ns 对 phase 寄存器进行写操作, 写入 0x0001_000a, 390ns 时向 config 寄存器写入 1 表示计算 sin, 均在第二个周期置 d_valid 信号为 1, d_opcode 为 0



时序分析: 600ns 时对 result 寄存器进行读操作, 第二个周期向 cordic 模块传地址, 第三个周期读数据, 置 d_valid 和 rdata_v 为 1, d_opcode 为 1, 读出计算的 sin 函数结果为 0x0000_fc1b。



时序分析: 660ns 对 phase 寄存器进行写操作, 写入 0x0001_0014, 720ns 时向 config 寄存器写入 0x101 表示计算 cos, 均在第二个周期置 d_valid 信号为 1, d_opcode 为 0



时序分析: 930ns 对 result 寄存器进行读操作, 第二个周期向 cordic 模块传地址, 第三个周期读数据, 置 d_valid 和 rdata_v 为 1, d_opcode 为 1, 读出计算的 cos 函数结果为 0xffff_a872。