基于FPGA的AXI_FULL主机模块测试（含源码）

前文对axi_full总线的各个信号进行了简要讲解，本文使用 Verilog　HDL编写主机模块，对其时序进行讲解，最终进行仿真。

首先可以在vivado中获取官方提供的axi_full主机模块，参考其设计思路，但是官方设计过于繁琐，本文不对其模块代码进行讲解，而是自己实现这个模块的功能，从而熟悉该接口的设计。

该模块的功能主要是向从机的地址中写入固定突发长度的数据，然后读出这些地址的数据进行比较，如果读出数据与写入数据不一致，则停止读写。

1、设计思路

Axi_full协议是支持同时读写的，但是同时读写从机同一地址数据会导致读出错误数据。因为本文只是验证读、写各个通道时序，保证读写数据正确性，对于读写速度没有太高要求。因此可以通过一个状态来控制读写，写对一段地址写入数据，然后再读出该段地址数据，进行对比，如果没有错误，则再下一段地址中继续写入数据，由此反复执行。

状态机的状态转换图如下所示，包含空闲、写数据、读数据、错误四个状态，当跳转到错误状态时被锁死。

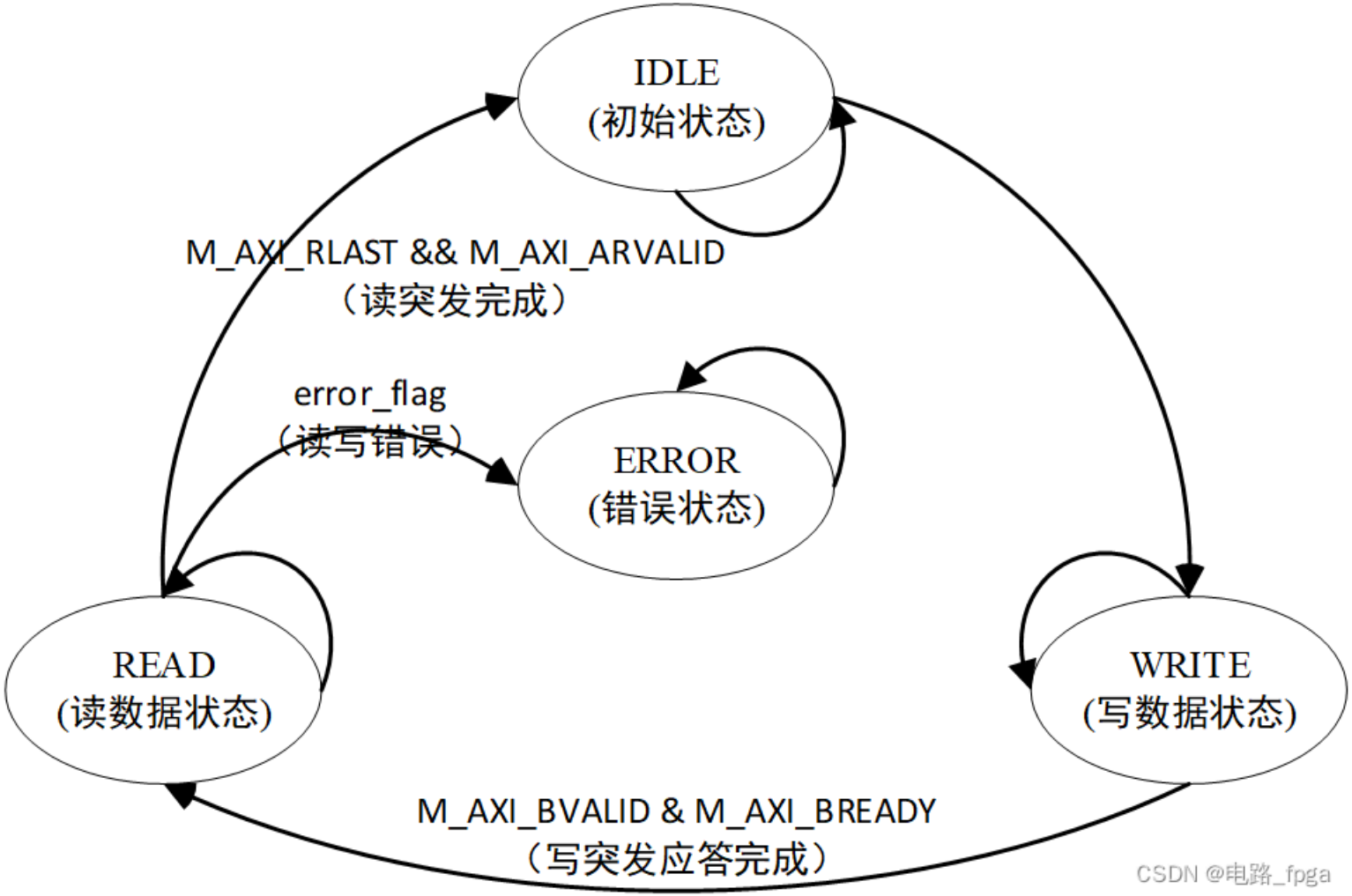

图1 状态转换图

状态机初始位于空闲状态，下个时钟跳转到写突发状态，当写应答通道握手成功时，表示写突发完成，跳转到读突发状态，当读出数据与写入数据错误时，跳转到错误状态。当读写数据一致且读突发传输最后一个数据，表示读突发完成，跳转到空闲状态开始下一次读写操作。

当读、写首地址发送之后，突发地址需要增加突发长度乘以突发字节数，作为下一次突发传输的首地址。

其余信号的设计思路可以对照代码进行讲解，都比较简单。

2、代码讲解

首先该模块的端口与官方提供的模块保持一致，如下所示，包含axi_full接口协议的端口信号，前文已经讲解过这些信号且每个信号后面都有注释，本文就不再赘述每个信号的功能了。

```
module axi_full_master #(
    parameter       C_M_TARGET_SLAVE_BASE_ADDR      =   32'h40000000    ,//读写从机的基地址。
    parameter       C_M_AXI_BURST_LEN               =   16              ,//突发长度1、2、4、8、16、32、64、128、256。
    parameter       C_M_AXI_ID_WIDTH                =   1               ,//ID信号位宽。
    parameter       C_M_AXI_ADDR_WIDTH              =   32              ,//读、写地址位宽。
    parameter       C_M_AXI_DATA_WIDTH              =   32              ,//读、写数据位宽。
    parameter       C_M_AXI_AWUSER_WIDTH            =   0               ,//用户写地址总线的宽度
    parameter       C_M_AXI_ARUSER_WIDTH            =   0               ,//用户读地址总线的宽度。
    parameter       C_M_AXI_WUSER_WIDTH             =   0               ,//用户写数据总线宽度。
    parameter       C_M_AXI_RUSER_WIDTH             =   0               ,//用户读数据总线宽度。
    parameter       C_M_AXI_BUSER_WIDTH             =   0                //用户写响应总线的宽度。
)(
    input                                           M_AXI_ACLK      ,//AXI时钟信号。
    input                                           M_AXI_ARESETN   ,//AXI复位信号，默认低电平有效。
    //AXI写地址通道
    output          [C_M_AXI_ID_WIDTH-1 : 0]        M_AXI_AWID      ,//AXI写地址通道ID信号。
    output  reg     [C_M_AXI_ADDR_WIDTH-1 : 0]      M_AXI_AWADDR    ,//AXI写地址通道地址信号。
    output          [7 : 0]                         M_AXI_AWLEN     ,//AXI写地址通道突发长度信号。
    output          [2 : 0]                         M_AXI_AWSIZE    ,//AXI写地址通道突发大小信号，该信号指示突发中每次传输的数据大小。
    output          [1 : 0]                         M_AXI_AWBURST   ,//AXI写地址通道突发类型信号。
    output                                          M_AXI_AWLOCK    ,//AXI写地址通道锁信号，只是为了兼容AXI3总线。
    output          [3 : 0]                         M_AXI_AWCACHE   ,//AXI写地址通道内存类型信号。
    output          [2 : 0]                         M_AXI_AWPROT    ,//AXI写地址通道保护类型信号。
    output          [3 : 0]                         M_AXI_AWQOS     ,//AXI写地址通道服务质量信号。
    output          [C_M_AXI_AWUSER_WIDTH-1 : 0]    M_AXI_AWUSER    ,//AXI写地址通道用户自定义信号。
    output  reg                                     M_AXI_AWVALID   ,//AXI写地址通道有效指示信号。
    input                                           M_AXI_AWREADY   ,//AXI写地址通道地址应答信号。
    //AXI写数据通道。
    output  reg     [C_M_AXI_DATA_WIDTH-1 : 0]      M_AXI_WDATA     ,//AXI写数据通道写数据信号。
    output          [C_M_AXI_DATA_WIDTH/8-1 : 0]    M_AXI_WSTRB     ,//AXI写数据通道写数据掩码信号。
    output  reg                                     M_AXI_WLAST     ,//AXI写数据通道突发传输最后一个信号。
    output          [C_M_AXI_WUSER_WIDTH-1 : 0]     M_AXI_WUSER     ,//AXI写数据通道用户自定义信号。
    output  reg                                     M_AXI_WVALID    ,//AXI写数据通道有效指示信号。
    input                                           M_AXI_WREADY    ,//AXI写数据通道数据应答信号。
    //AXI写应答通道。
    input           [C_M_AXI_ID_WIDTH-1 : 0]        M_AXI_BID       ,//AXI写响应通道响应ID信号。
    input           [1 : 0]                         M_AXI_BRESP     ,//AXI写响应通道写回复信号。
```

```
    input           [C_M_AXI_BUSER_WIDTH-1 : 0]         M_AXI_BUSER     ,//AXI写响应通道用户自定义信号。
    input                                               M_AXI_BVALID    ,//AXI写响应通道有效指示信号。
    output  reg                                         M_AXI_BREADY    ,//AXI写响应通道主机应答信号。
    //AXI读地址通道。
    output          [C_M_AXI_ID_WIDTH-1 : 0]            M_AXI_ARID      ,//AXI读地址通道ID信号。
    output  reg     [C_M_AXI_ADDR_WIDTH-1 : 0]          M_AXI_ARADDR    ,//AXI读地址通道读地址信号。
    output          [7 : 0]                             M_AXI_ARLEN     ,//AXI读地址通道数据突发长度信号。
    output          [2 : 0]                             M_AXI_ARSIZE    ,//AXI读地址通道突发大小信号，该信号指示突发中每次传输的数据大小。
    output          [1 : 0]                             M_AXI_ARBURST   ,//AXI读地址通道突发类型信号。
    output                                              M_AXI_ARLOCK    ,//AXI读地址通道锁信号，只是为了兼容AXI3总线。
    output          [3 : 0]                             M_AXI_ARCACHE   ,//AXI读地址通道内存类型信号。
    output          [2 : 0]                             M_AXI_ARPROT    ,//AXI读地址通道保护类型信号。
    output          [3 : 0]                             M_AXI_ARQOS     ,//AXI读地址通道服务质量信号。
    output          [C_M_AXI_ARUSER_WIDTH-1 : 0]        M_AXI_ARUSER    ,//AXI读地址通道用户自定义信号
    output  reg                                         M_AXI_ARVALID   ,//AXI读地址通道有效指示信号。。
    input                                               M_AXI_ARREADY   ,//AXI读地址通道地址应答信号。。
    //AXI读数据通道。
    input           [C_M_AXI_ID_WIDTH-1 : 0]            M_AXI_RID       ,//AXI读数据通道ID信号。
    input           [C_M_AXI_DATA_WIDTH-1 : 0]          M_AXI_RDATA     ,//AXI读数据通道读数据信号。
    input           [1 : 0]                             M_AXI_RRESP     ,//AXI读数据通道读回复信号。
    input                                               M_AXI_RLAST     ,//AXI读数据通道突发传输最后一个信号。
    input           [C_M_AXI_RUSER_WIDTH-1 : 0]         M_AXI_RUSER     ,//AXI读数据通道用户自定义信号。
    input                                               M_AXI_RVALID    ,//AXI读数据通道有效指示信号。
    output  reg                                         M_AXI_RREADY     //AXI读数据通道主机应答信号。
);
```

下面是状态机的编码以及通过函数去计算读写数据的大小和突发计数器的位宽。

```
    //Four-stage state machine;
    localparam      IDLE                        =       4'b0001         ;//状态机的空闲状态编码;
    localparam      WRITE                       =       4'b0010         ;//状态机的写数据状态编码;
    localparam      READ                        =       4'b0100         ;//状态机的读数据状态编码;
    localparam      ERROR                       =       4'b1000         ;//状态机的错误状态编码;
    localparam      SIZE                        =       clogb2(C_M_AXI_DATA_WIDTH/8-1   );//计算突发数据位宽的字节数。
    localparam      CNT_W                       =       clogb2(C_M_AXI_BURST_LEN - 1    );//通过突发长度计算计数器位宽.

```

```
    //自动计算位宽函数。
    function integer clogb2(input integer depth);begin
        if(depth == 0)
            clogb2 = 1;
        else if(depth != 0)
            for(clogb2=0 ; depth>0 ; clogb2=clogb2+1)
                depth=depth >> 1;
        end
    endfunction
```

下面是状态机现态与次态的跳转。

```
    //状态机次态到现态的跳转；
    always@(posedge M_AXI_ACLK)begin
        if(!M_AXI_ARESETN)begin
            state_c <= IDLE;
        end
        else begin
            state_c <= state_n;
        end
    end

    //状态机次态的跳转；
    always@(*)begin
        case(state_c)
            IDLE : begin//跳转到写数据状态；
                state_n = WRITE;
            end
            WRITE : begin
                if(M_AXI_BVALID & M_AXI_BREADY)begin//写数据完成，跳转到读数据状态；
                    state_n = READ;
                end
                else begin
                    state_n = state_c;
                end
            end
```

```
            READ : begin
                if(error_flag)begin//检测到接收的数据错误。
                    state_n = ERROR;
                end
                else if(M_AXI_RLAST && M_AXI_ARVALID)begin//读突发完成，跳转到空闲状态。
                    state_n = IDLE;
                end
                else begin
                    state_n = state_c;
                end
            end
            ERROR : begin//一直停留在错误状态；
                state_n = state_c;
            end
            default:begin
                state_n = IDLE;
            end
        endcase
    end
```

下面是写地址通道的一些固定数值的信号，写地址ID为0，由于突发长度是输入数据加1，因此输入时要将长度减1。

```
    assign M_AXI_AWID    = 0;//读写ID数值保持一致即可。
    assign M_AXI_AWLEN   = C_M_AXI_BURST_LEN - 1;//突发长度；
    assign M_AXI_AWSIZE  = SIZE;//突发数据的位宽字节数；
    assign M_AXI_AWBURST = 2'b01;//突发类型为递增类型，即地址逐渐累加。
    assign M_AXI_AWLOCK  = 1'b0;//AXI4不支持锁事务，只为了兼容AXI3而存在的信号；
    assign M_AXI_AWCACHE = 4'b0010;//不使用缓存。
    assign M_AXI_AWPROT  = 3'd0;//写地址端口为0；
    assign M_AXI_AWQOS   = 4'd0;//信号指令为0，不使用。
    assign M_AXI_AWUSER  = 0;//用户自定义数据为0；
```

写地址初始为基地址，每完成一次写入突发，需要将地址增加突发长度乘以突发数据字节数，作为下一次突发写入的首地址。

```
    //生成写突发的基地址信号,初始值为基地址;
    always@(posedge M_AXI_ACLK)begin
        if(M_AXI_ARESETN==1'b0)begin//初始值为基地址;
            M_AXI_AWADDR <= C_M_TARGET_SLAVE_BASE_ADDR;
        end
        else if(M_AXI_BREADY & M_AXI_BVALID)begin//每次写完后,跳过已经被写数据段的地址.
            M_AXI_AWADDR <= M_AXI_AWADDR + (C_M_AXI_BURST_LEN * C_M_AXI_DATA_WIDTH/8);
        end
    end
```

然后是写地址有效指示信号，当状态机从空闲状态跳转到写突发状态时拉高，当接收到从机应答信号时拉低，其余时间保持不变。

```
    //生成写地址有效指示信号，与应答信号握手。
    always@(posedge M_AXI_ACLK)begin
        if(M_AXI_ARESETN==1'b0)begin//初始值为0;
            M_AXI_AWVALID <= 1'b0;
        end
        else if(M_AXI_AWREADY)begin//当应答信号有效时拉低。
            M_AXI_AWVALID <= 1'b0;
        end
        else if(state_c == IDLE && state_n == WRITE)begin//当状态机从空闲状态跳转到写数据状态时拉高；
            M_AXI_AWVALID <= 1'b1;
        end
    end
```

需要对写入数据个数进行计数，因此需要一个写计数器，当写数据和写数据应答信号同时有效时加1，当状态机不处于写状态时清零。

写数据掩码信号所有位置为高电平，因为每次写数据都需要全部写入从机。

```
    //生成写数据计数器,用于计数写入数据个数.
    always@(posedge M_AXI_ACLK)begin
        if(M_AXI_ARESETN==1'b0)begin//初始值为0;
            wr_cnt <= {{CNT_W}{1'b0}};
        end
        else if(state_c != WRITE)begin//状态机不处于写数据状态时清零.
            wr_cnt <= {{CNT_W}{1'b0}};
        end
        else if(M_AXI_WVALID & M_AXI_WREADY)begin//当写入一个数据时加1.
            wr_cnt <= wr_cnt + 1;
```

```
        end
    end

    assign M_AXI_WUSER = 0;//用户自定义信号输出0;
    assign M_AXI_WSTRB = {{C_M_AXI_DATA_WIDTH/8}{1'b1}};//将掩码信号所有位拉高,不使用掩码功能.

```

本文向地址中写入递增的数据，因此当数据写入时，写数据加1。经过axi_lite文章分析，写数据和写地址可以同时有效，因此在状态机跳转时就可以写入数据，将写数据指示信号拉高，直到写入最后一个数据为止。

```
    always@(posedge M_AXI_ACLK)begin
        if(M_AXI_ARESETN==1'b0)begin//初始值为0;
            M_AXI_WDATA <= 0;
        end
        else if(M_AXI_WVALID & M_AXI_WREADY)begin//当写入一个数据时加1.
            M_AXI_WDATA <= M_AXI_WDATA + 1;
        end
    end

    //生成写地址有效指示信号.
    always@(posedge M_AXI_ACLK)begin
        if(M_AXI_ARESETN==1'b0)begin//初始值为0;
            M_AXI_WVALID <= 1'b0;
        end
        else if((wr_cnt == C_M_AXI_BURST_LEN-1) && (M_AXI_WVALID & M_AXI_WREADY))begin//写完最后一个数据时拉低;
            M_AXI_WVALID <= 1'b0;
        end
        else if(state_c == IDLE && state_n == WRITE)begin//当状态机从空闲状态跳转到写数据状态时拉高;
            M_AXI_WVALID <= 1'b1;
        end
    end
```

在写入最后一个数据时，将WLAST信号拉高。

```
    //生成写突发结束信号,初始值为低电平.
    always@(posedge M_AXI_ACLK)begin
        if(M_AXI_ARESETN==1'b0)begin//初始值为0;
            M_AXI_WLAST <= 1'b0;
        end
        else if((M_AXI_WVALID & M_AXI_WREADY))begin
            if(wr_cnt == C_M_AXI_BURST_LEN-1)//当写入最后一次数据后拉低;
                M_AXI_WLAST <= 1'b0;
            else if(wr_cnt == C_M_AXI_BURST_LEN-2)//当要写入最后一次数据时拉高.
                M_AXI_WLAST <= 1'b1;
        end
        else begin
            M_AXI_WLAST <= 1'b0;
        end
    end
```

之后是写应答通道的应答信号，在数据全部写入后拉高，当从机输出应答信号有效时拉低，其余时间保持不变。

```
    //生成AXI写响应通道主机应答信号。
    always@(posedge M_AXI_ACLK)begin
        if(M_AXI_ARESETN==1'b0)begin//初始值为0;
            M_AXI_BREADY <= 1'b0;
        end
        else if(M_AXI_BVALID)begin//当从机输出有效的应答信号后拉低.
            M_AXI_BREADY <= 1'b0;
        end
        else if(M_AXI_WLAST & M_AXI_WVALID & M_AXI_WREADY)begin//当突发写传输完成时拉高;
            M_AXI_BREADY <= 1'b1;
        end
    end
```

读地址相关信号的取值和设计与写地址相关信号设计一致，如下所示。

```
    assign M_AXI_ARID     = 0;//读写ID数值保持一致即可。
    assign M_AXI_ARLEN    = C_M_AXI_BURST_LEN - 1;//突发长度；
    assign M_AXI_ARSIZE   = SIZE;//突发数据的位宽字节数；
    assign M_AXI_ARBURST  = 2'b01;//突发类型为递增类型，即地址逐渐累加。
```

```
    assign M_AXI_ARLOCK   = 1'b0;//AXI4不支持锁事务，只为了兼容AXI3而存在的信号；
    assign M_AXI_ARCACHE  = 4'b0010;//不使用缓存。
    assign M_AXI_ARPROT   = 3'd0;//写地址端口为0；
    assign M_AXI_ARQOS    = 4'd0;//信号指令为0，不使用。
    assign M_AXI_ARUSER   = 0;//用户自定义数据为0；

    //生成读突发的基地址信号,初始值为基地址;
    always@(posedge M_AXI_ACLK)begin
        if(M_AXI_ARESETN==1'b0)begin//初始值为基地址;
            M_AXI_ARADDR <= C_M_TARGET_SLAVE_BASE_ADDR;
        end
        else if(M_AXI_RLAST)begin//每次读完后,跳过已经被读数据段的地址.
            M_AXI_ARADDR <= M_AXI_ARADDR + (C_M_AXI_BURST_LEN * C_M_AXI_DATA_WIDTH/8);
        end
    end

    //生成读地址有效指示信号，与应答信号握手。
    always@(posedge M_AXI_ACLK)begin
        if(M_AXI_ARESETN==1'b0)begin//初始值为0;
            M_AXI_ARVALID <= 1'b0;
        end
        else if(M_AXI_ARREADY)begin//当应答信号有效时拉低。
            M_AXI_ARVALID <= 1'b0;
        end
        else if(state_c == WRITE && state_n == READ)begin//当状态机从写数据跳转到读数据状态时拉高；
            M_AXI_ARVALID <= 1'b1;
        end
    end

    //生成读数据响应信号,初始为低电平.
    always@(posedge M_AXI_ACLK)begin
        if(M_AXI_ARESETN==1'b0)begin//初始值为0;
            M_AXI_RREADY <= 1'b0;
        end
        else if(M_AXI_RLAST)begin//接收完一次突发数据时拉低.
            M_AXI_RREADY <= 1'b0;
        end
        else if(M_AXI_ARREADY & M_AXI_ARVALID)begin//写入读地址后拉高,准备接收数据.
            M_AXI_RREADY <= 1'b1;
        end
```

```
    end

    //读数据计数器,初始值为0,当读取一个数据时加1.
    always@(posedge M_AXI_ACLK)begin
        if(M_AXI_ARESETN==1'b0)begin//初始值为0;
            rd_cnt <= {{CNT_W}{1'b0}};
        end
        else if(state_c != READ)begin//状态机不处于读数据状态时清零.
            rd_cnt <= {{CNT_W}{1'b0}};
        end
        else if(M_AXI_RVALID & M_AXI_RREADY)begin//当读出一个数据时加1.
            rd_cnt <= rd_cnt + 1;
        end
    end
```

对读出数据进行计数，比较读出数据与写入数据是否相同，如果不同则把错误标志信号拉高，之后状态机锁死。

```
    //生成错误指示信号,当读出的数据与写入数据不相等时拉高,其余时间为低电平.
    always@(posedge M_AXI_ACLK)begin
        if(M_AXI_ARESETN==1'b0)begin//初始值为0;
            error_flag <= 0;
        end
        else if(M_AXI_WVALID && M_AXI_WREADY && (state_c == READ))begin
            error_flag <= ((M_AXI_ARADDR - C_M_TARGET_SLAVE_BASE_ADDR + rd_cnt) != M_AXI_RDATA);//当读出数据不等于读计数器时拉高,其余时间拉低.
        end
    end
```

该模块的设计到此结束，其实页比较简单，相比axi_lite只是增加了突发功能和几个信号而已。

3、模块仿真

之后对该模块进行仿真，仿真还需要一个从机模块，可以通过vivado生成一个官方提供的从机模块用于仿真。

然后编写TestBench，由于信号过多，代码过长，不贴代码，可以获取工程进行查看。

将突发长度设置为16，读写数据位宽设置为32，仿真结果如下所示：

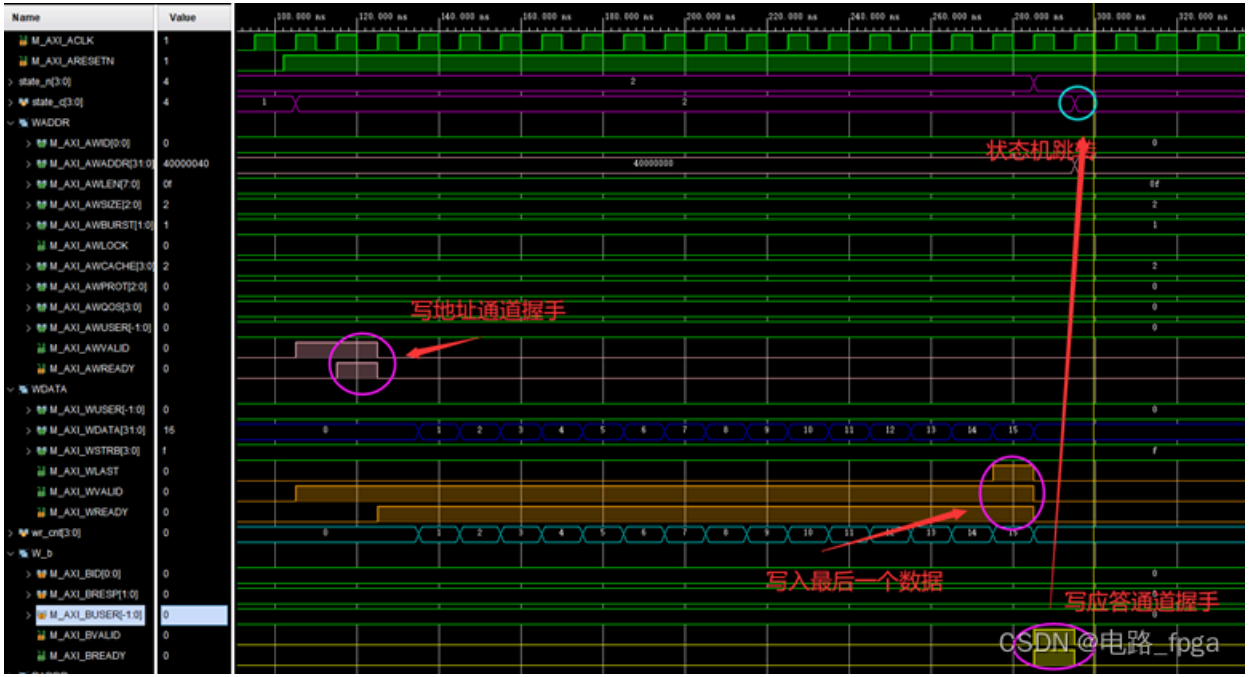

图2 写突发仿真

首先写地址通道先进行握手，之后写数据通道从机也开始握手，对写入数据个数（写数据握手次数）进行计数，写入最后一个数据时，需要把WLAST信号拉高。

之后写应答通道握手，返回的状态值WRESP为0，且BID与AWID一致，表示此次写入成功，之后状态机跳转到读状态。

读状态的仿真时序如下图所示。

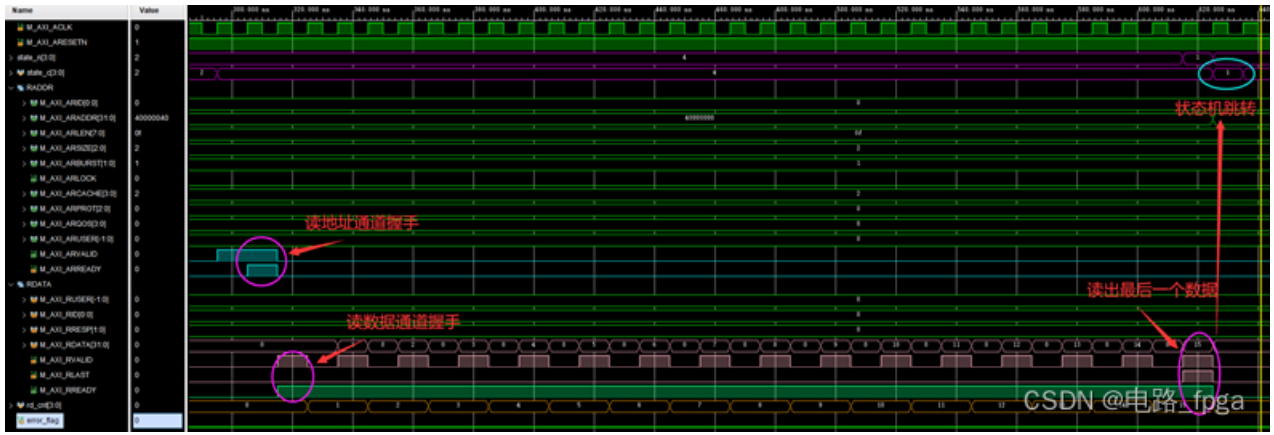

图3 读突发仿真

首先读地址通道应答信号，读地址与前文写地址一样，然后从机开始输出数据。当主机与从机握手时，从机输出数据才是有效的。从机输出数据与前文写入数据一致，证明读写时序没有问题，当主机接收完最后一个数据后，状态机跳转到空闲状态。

上面完成了一次读写操作，后续读、写数据都是没有问题的，下图是随意截取的一次读写时序（部分不关注信号删除，屏幕放不下），读、写数据无误。

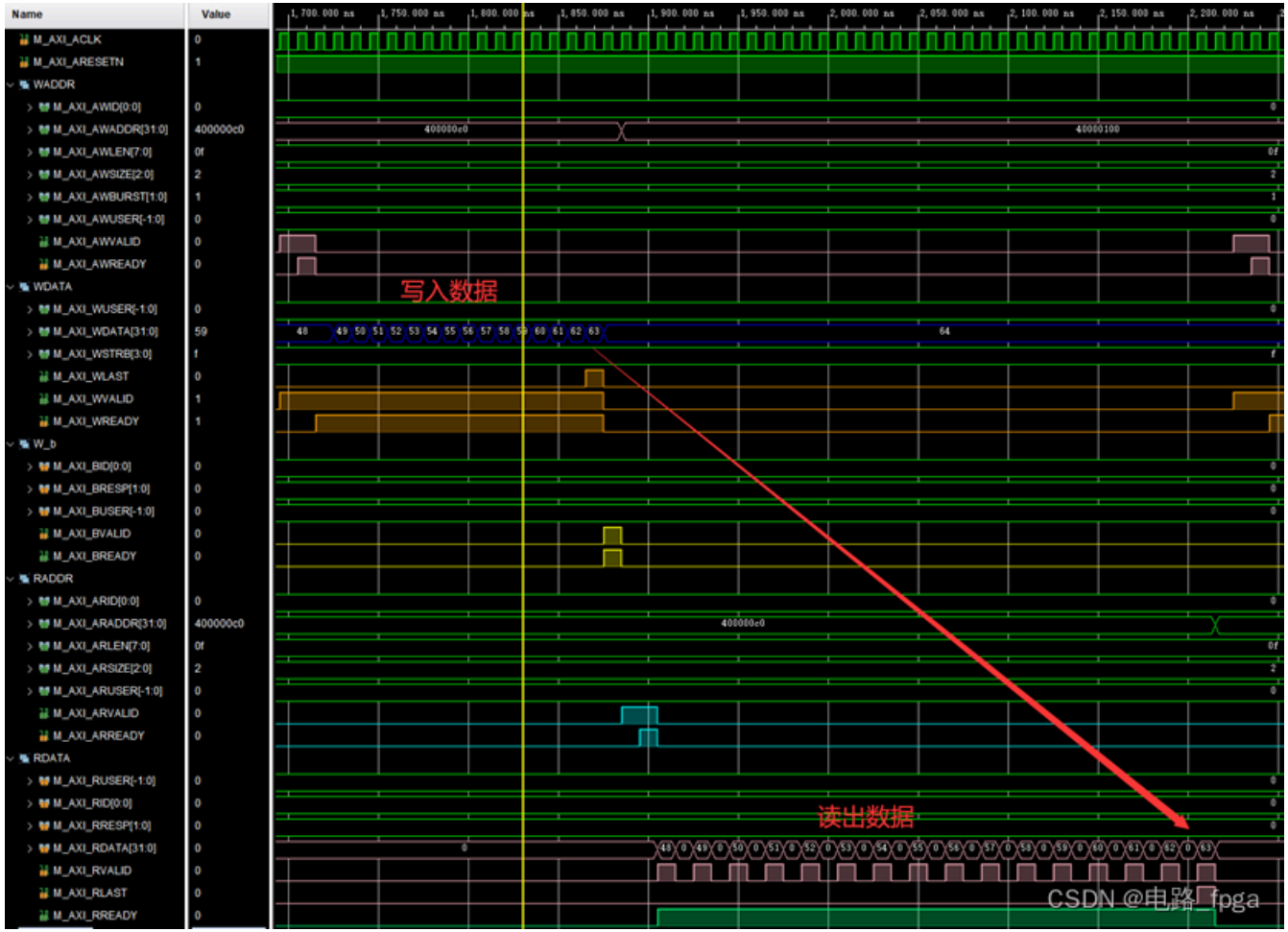

图4 读、写突发仿真

由此证明该模块的设计没有问题，该协议在FPGA中的使用其实也比较简单，下文将通过该接口驱动一个比较常用的IP，来实现具体功能。

需要本工程在公众号后台回复“**AXI_FULL主机测试**”（不包括引号）即可。

如果对文章内容理解有疑惑或者对代码不理解，可以在评论区或者后台留言，看到后均会回复！

如果本文对您有帮助，还请多多点赞👍、评论💬和收藏⭐！您的支持是我更新的最大动力！将持续更新工程！