基于Verilog HDL的axi-lite主机模块

前文对axi_lite接口协议的各个信号做了详细讲解，本文通过 Verilog　Hdl编写一个通用接口转axi_lite接口协议的模块。

1、生成xilinx官方提供axi源码

Xilinx其实给用户提供了axi相关模块，获取方式如下，首先打开vivado软件，然后点击Manage IP，之后选择New IP Location，如下图所示。

图1 创建IP

直接点击下一步。

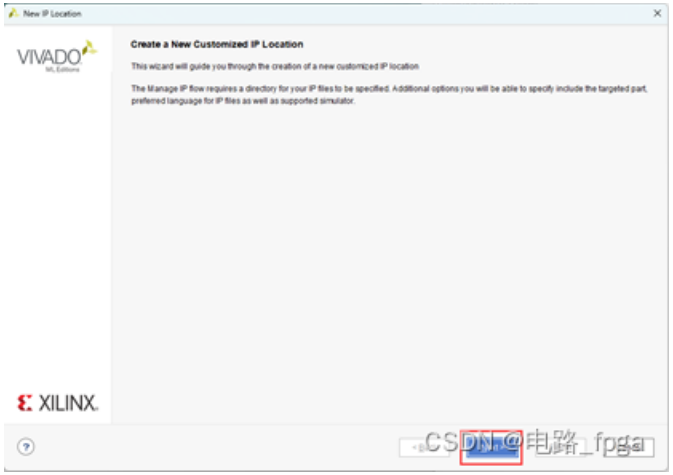

图2 向导界面

之后需要选择器件型号，然后确定工程路径，之后点击Finish。

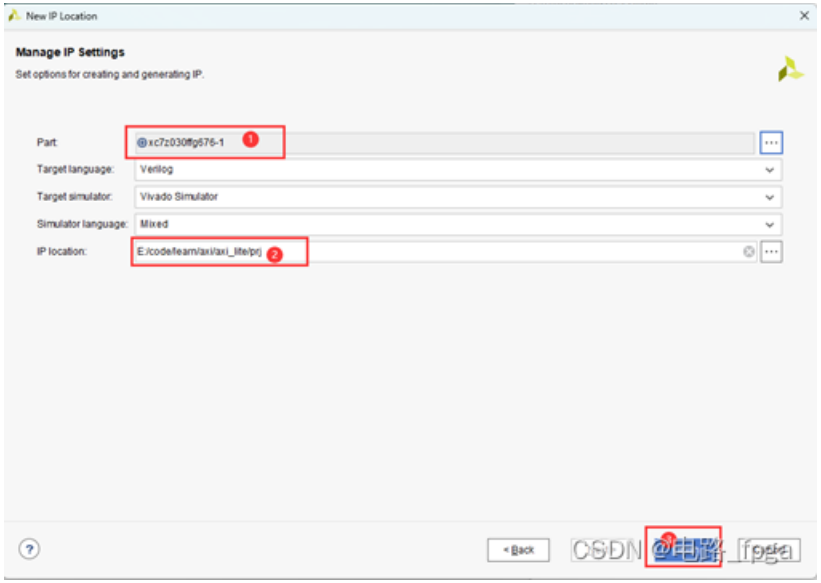

图3 Manage IP Settings

进入工程后，在Tools选项卡下点击Create and Package New IP，如下图所示。

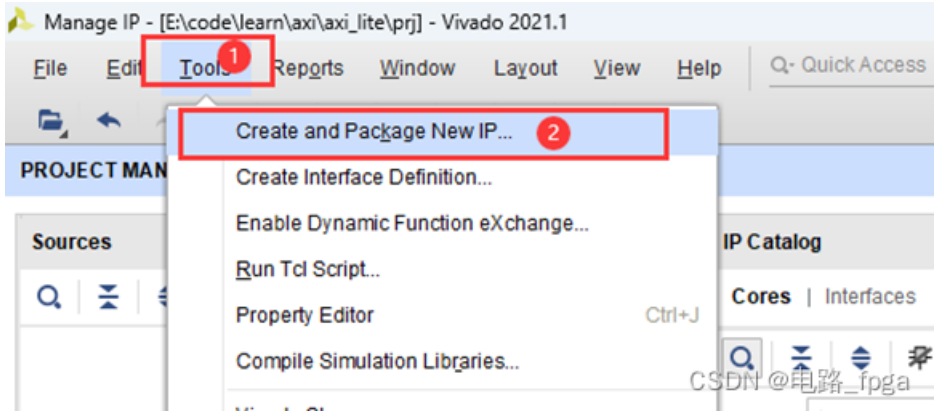

图4 生成IP

进入向导界面后直接点击下一步。

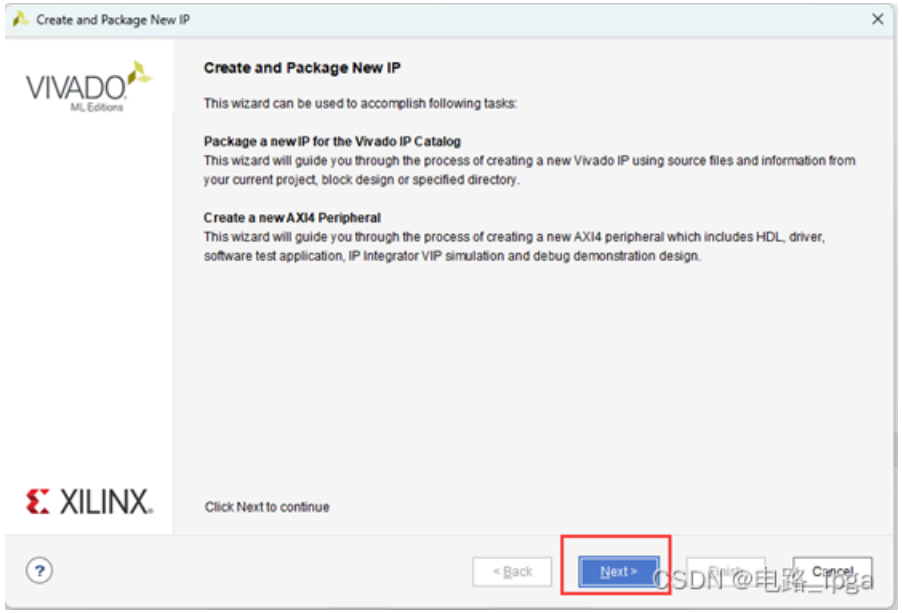

图5 Create and Package New IP

然后选择创建AXI外围设备，如下图所示。

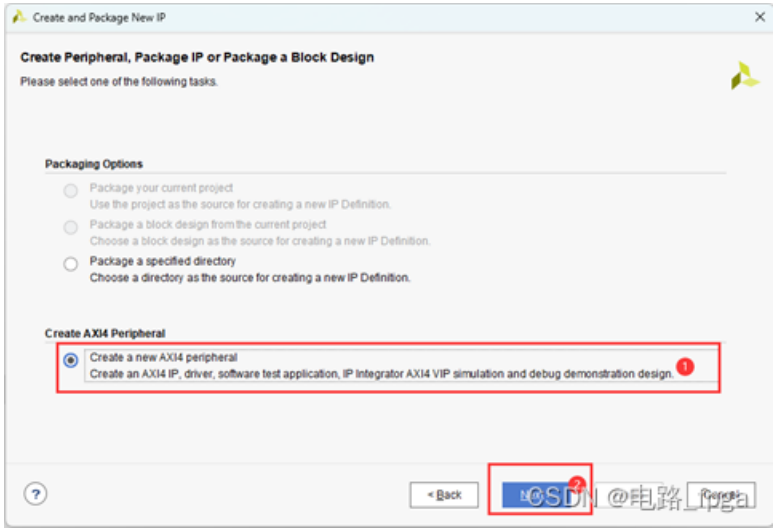

图6 创建AXI 外围设备

根据需求修改一下IP名，然后点击下一步，如下图所示。

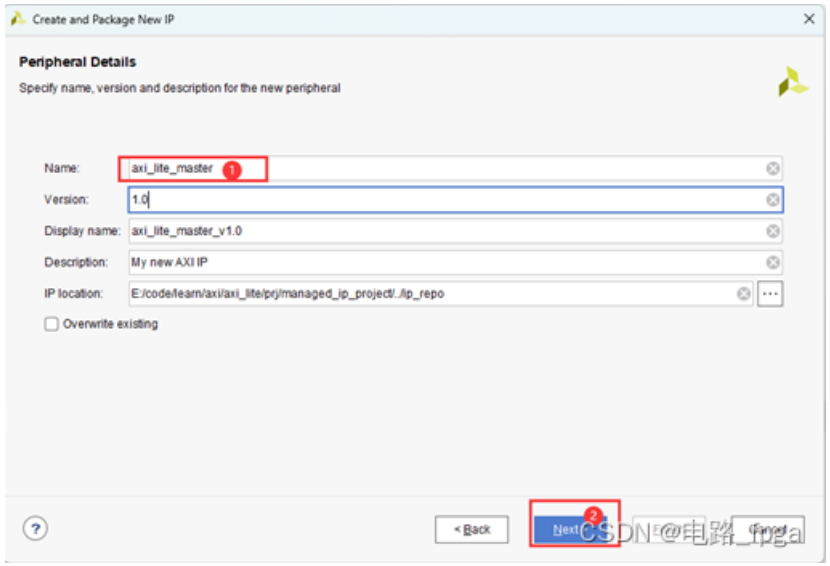

图7 修改IP名

然后选择生成IP的协议类型，此处选择lite协议，然后选择主机，数据位宽只能固定为32位。之后点击下一步，如下图所示。

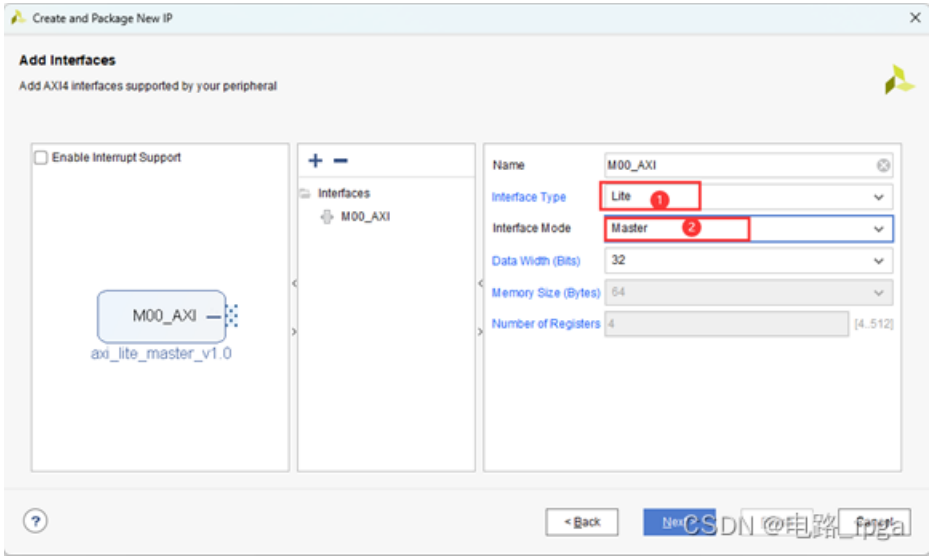

图8 生成axi_lite主机模块

通过修改上图设置，可以生成axi_lite、axi_full、axi_stream协议的主机和从机模块，后文会使用其余模块，就不再详细讲解。

最后需要选择Edit IP，才会生成IP的源文件，如下图所示。

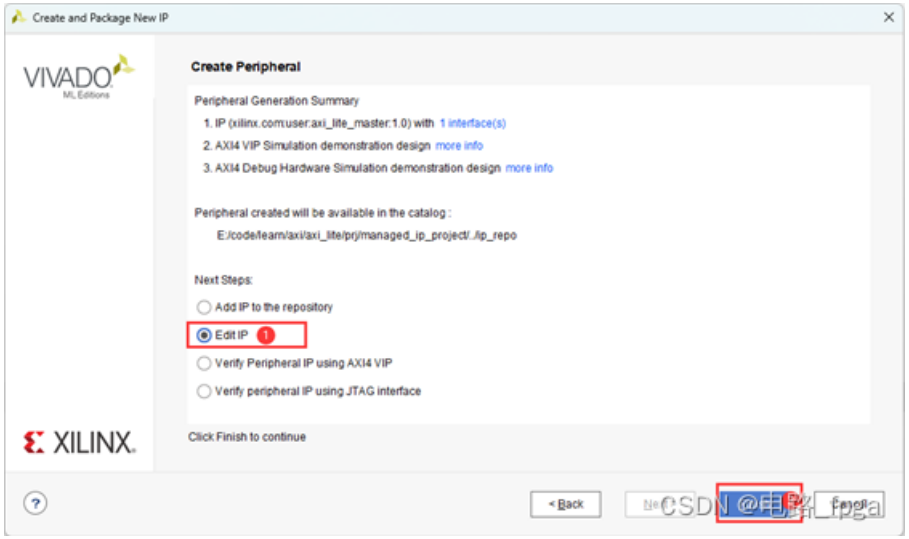

图9 生成IP

之后会生成一个工程，生成的工程会包含axi_lite_master_v1_0_M00_AXI的一个模块，这个模块就是生成的axi_lite_master模块。

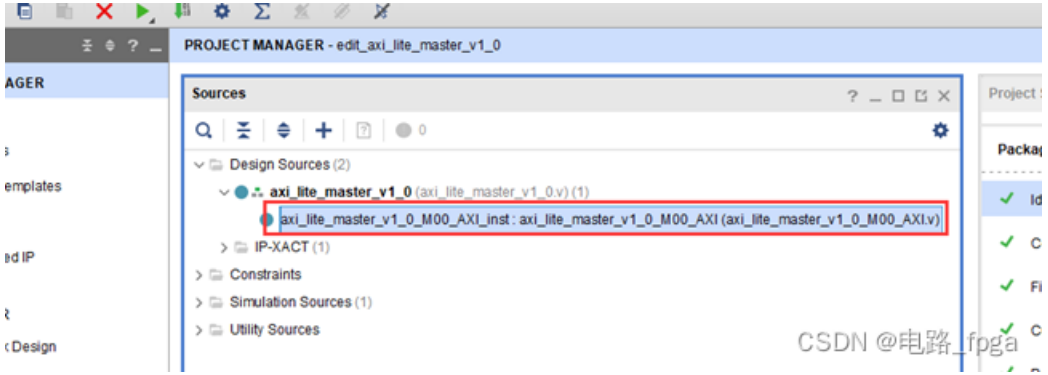

图10 生成文件

本文只是告诉读者vivado能够生成一个 axi协议 的相关模块，但是该模块的代码不够简洁，其实使用起来没那么方便。

后文不会对生成的模块进行解读和使用，后文会根据前文所讲协议内容自己编写一个axi_lite_master模块，而该模块的axi_lite端口信号与生成的模块保持一致。

2、axi_lite_master模块设计

用户端口肯定会设计的比较简单，只包括读写使能、地址、数据信号，由于axi_lite接口需要和从机进行应答，那么用户端口写入的数据和地址就不能立即写入，需要等待。

解决这个问题的方法有两种思路，第一种是在写和读部分都给用户输出一个模块忙闲指示信号，只有当模块对应功能空闲时，才能进行下次操作。

第二种就是在写数据、写地址、读地址端口处加一个FIFO，用来存储用户的数据和指令，当FIFO不为空，就读出FIFO中的数据通过axi_lite发送给从机。这种方式的优点是用户不需要等待模块空闲，直接将需要发送的指令、数据输入即可，简化逻辑，但是FIFO会消耗更多资源。

本文采用第二种设计方式，整个模块通过检测三个FIFO是否为空，向从机发起读写时序。

端口信号如下所示，首先是axi_lite的时钟、复位和五个通道的信号，然后是用户的端口信号，用户的写使能信号与写数据、写地址信号对齐，读使能与读地址信号对齐。用户的地址信号位宽默认为8位，而axi_lite的地址和数据信号一般均为32位，可以根据实际情况进行拼接。

```
module axi_lite_master #(
    parameter           UADDR_W     =           8               ,//用户地址位宽；
    parameter           UDATA_W     =           32              ,//用户数据位宽；
    parameter           ADATA_W     =           32               //AXI_LITE的地址和数据位宽；
)(
    input                                       M_AXI_ACLK      ,//AXI接口时钟信号；
    input                                       M_AXI_ARESETN   ,//AXI接口复位信号，低电平有效；

```

```
    //AXI写地址通道信号;
    output  reg         [ADATA_W - 1 : 0]       M_AXI_AWADDR    ,//AXI地址信号;
    output              [2 : 0]                 M_AXI_AWPROT    ,//AXI地址端口信号;
    output  reg                                 M_AXI_AWVALID   ,//AIX地址数据有效指示信号，高电平有效;
    input                                       M_AXI_AWREADY   ,//AXI地址数据应答信号。
    //AXI写数据通道信号;
    output  reg         [ADATA_W - 1 : 0]       M_AXI_WDATA     ,//AXI写数据信号;
    output              [ADATA_W / 8 - 1 : 0]   M_AXI_WSTRB     ,//AXI写数据掩码信号，低电平有效;
    output  reg                                 M_AXI_WVALID    ,//AXI写入数据有效指示信号，高电平有效;
    input                                       M_AXI_WREADY    ,//AXI写入数据应答信号，高电平有效;
    //AXI写应答通道信号;
    input               [1 : 0]                 M_AXI_BRESP     ,//AXI写应答信号，为0时表示写入正确;
    input                                       M_AXI_BVALID    ,//AXI写应答有效指示信号，高电平有效;
    output  reg                                 M_AXI_BREADY    ,//AXI写应答响应信号，高电平有效;
    //AXI读地址通道信号;
    output  reg         [ADATA_W - 1 : 0]       M_AXI_ARADDR    ,//AXI读地址信号;
    output              [2 : 0]                 M_AXI_ARPROT    ,//AXI读地址端口信号;
    output  reg                                 M_AXI_ARVALID   ,//AXI读地址有效指示信号，高电平有效;
    input                                       M_AXI_ARREADY   ,//AXI读地址应答信号，高电平有效;
    //AXI读数据通道信号;
    input               [ADATA_W - 1 : 0]       M_AXI_RDATA     ,//AXI读数据信号;
    input               [1 : 0]                 M_AXI_RRESP     ,//AXI读数据状态应答信号，为0表示读数据正确;
    input                                       M_AXI_RVALID    ,//读数据有效指示信号，高电平有效;
    output  reg                                 M_AXI_RREADY    ,//主机接收从机读出数据的应答信号，高电平有效;
    //用户写数据信号;
    input                                       wr_en           ,//写使能信号，高电平有效;
    input               [UADDR_W - 1 : 0]       waddr           ,//写地址信号，与写使能对齐;
    input               [UDATA_W - 1 : 0]       wdata           ,//写数据信号，与写使能对齐;
    //用户读数据信号;
    input                                       rd_en           ,//读使能信号，高电平有效;
    input               [UADDR_W - 1 : 0]       raddr           ,//读地址信号，与读使能对齐;
    output  reg         [UDATA_W - 1 : 0]       rdata           ,//读数据信号;
    output  reg                                 rdata_vld       ,//读数据有效指示信号，高电平有效;
    output  reg                                 error            //错误指示信号，当写入错误或读出错误时为1，正常为0;
);
```

axi_lite的读、写地址端口类型信号一般为0，且写入数据不会加掩码，因此这几个信号直接连接对应电平输出即可。

```
    assign M_AXI_AWPROT = 3'd0 ;//写地址端口信号始终为0；
    assign M_AXI_ARPROT = 3'd0 ;//读地址端口信号始终为0；
    assign M_AXI_WSTRB = {{ADATA_W/8}{1'b1}};//写数据掩码信号，本模块写入数据全部有效，因此全为高电平；
    assign wfifo_empty = wa_fifo_empty & wa_fifo_empty;//由于写地址和写数据可以同时进行，所以将两个写FIFO空指示信号合并；
```

由于用户端口的写地址信号、写数据信号都与写使能对齐，且前文分析axi_lite的写数据和写地址信号可以对齐（隐含意思：两个FIFO可以一起读出数据）。因此可以把两个FIFO的空标志信号整理为一个空标志信号。

下面是写地址FIFO、读地址FIFO、写数据FIFO的例化，地址FIFO的数据位宽为8，数据FIFO的数据位宽为32，FIFO深度均为32。FIFO输出数据会滞后读使能一个时钟。

```
    //用于存储写地址的FIFO；
    addr_fifo_8x32 u_waddr_fifo (
        .clk    ( M_AXI_ACLK        ),//input wire clk;
        .din    ( waddr             ),//input wire [7 : 0] din;
        .wr_en  ( wr_en             ),//input wire wr_en;
        .rd_en  ( wfifo_rd_en       ),//input wire rd_en;
        .dout   ( wa_fifo_dout      ),//output wire [7 : 0] dout;
        .full   ( wa_fifo_full      ),//output wire full;
        .empty  ( wa_fifo_empty     ) //output wire empty;
    );

    //用于存储读地址的FIFO；
    addr_fifo_8x32 u_raddr_fifo (
        .clk    ( M_AXI_ACLK        ),//input wire clk;
        .din    ( raddr             ),//input wire [7 : 0] din;
        .wr_en  ( rd_en             ),//input wire wr_en;
        .rd_en  ( ra_fifo_rd_en     ),//input wire rd_en;
        .dout   ( ra_fifo_dout      ),//output wire [7 : 0] dout;
        .full   ( ra_fifo_full      ),//output wire full;
        .empty  ( ra_fifo_empty     ) //output wire empty;
    );

    //用于存储写数据的FIFO；
    data_fifo_32x32 u_wdata_fifo (
        .clk    ( M_AXI_ACLK        ),//input wire clk;
        .din    ( wdata             ),//input wire [31 : 0] din;
        .wr_en  ( wr_en             ),//input wire wr_en;

```

```
        .rd_en  ( wfifo_rd_en       ),//input wire rd_en;
        .dout   ( wd_fifo_dout      ),//output wire [31 : 0] dout;
        .full   ( wd_fifo_full      ),//output wire full;
        .empty  ( wd_fifo_empty     ) //output wire empty;
    );
```

写地址FIFO和写数据FIFO的读使能采用一个信号，即同时输出信号。

然后下面是一个写标志信号wr_flag，该信号为高电平表示正在进行axi_lite写操作。当写地址、写数据FIFO均有数据且不处于写状态时拉高，当写应答通道响应时拉低。

```
    //写标志信号，初始值为0，当写入数据时拉高，一次写入完成后拉低；
    always@(posedge M_AXI_ACLK)begin
        if(M_AXI_ARESETN==1'b0)begin//初始值为0;
            wr_flag <= 1'b0;
        end//当写应答通道的有效信号和应答信号均为高时，表示一次写数据完成；
        else if(M_AXI_BVALID & M_AXI_BREADY)begin
            wr_flag <= 1'b0;
        end//当不处于写数据状态且写FIFO中有数据时拉高；
        else if((~wa_fifo_empty) && (~wr_flag))begin
            wr_flag <= 1'b1;
        end
    end
```

在wr_flag信号拉高时两个FIFO的读使能信号有效，将读使能信号打一拍，作为两个FIFO输出数据的有效指示信号。

```
    //写地址FIFO读使能信号；
    always@(posedge M_AXI_ACLK)begin
        if(M_AXI_ARESETN==1'b0)begin//初始值为0;
            wfifo_rd_en <= 1'b0;
        end
        else begin//当FIFO没有空且写标志信号开始时拉高；
            wfifo_rd_en <= ((~wfifo_empty) && (~wr_flag));
        end
    end


```

```
    //将写地址FIFO读使能信号延迟一个时钟作为读出数据有效指示信号；
    always@(posedge M_AXI_ACLK)begin
        wfifo_rdata_vld <= wfifo_rd_en;
    end
```

当写地址FIFO输出数据有效时，把写FIFO输出数据赋值给axi_lite写地址通道的写地址信号，由于位宽可能不同，需要进行拼接。

同时把写数据FIFO输出数据赋值给axi_lite写数据通道的写数据信号，用户读写数据位宽可能与axi_lite写数据位宽不同，也可能需要拼接。

```
    //生成AXI写地址信号，初始值为0，当写地址FIFO读数据有效时，输出FIFO读出的数据，其余时间保持不变；
    always@(posedge M_AXI_ACLK)begin
        if(M_AXI_ARESETN==1'b0)begin//初始值为0;
            M_AXI_AWADDR <= {{ADATA_W}{1'b0}};
            M_AXI_WDATA <= {{ADATA_W}{1'b0}};
        end
        else if(wfifo_rdata_vld)begin//将写地址FIFO输出数据赋值给AXI地址信号；
            M_AXI_AWADDR <= {{{ADATA_W - UADDR_W}{1'b0}},wa_fifo_dout[UADDR_W - 1 : 0]};
            M_AXI_WDATA <= {{{ADATA_W - UDATA_W}{1'b0}},wd_fifo_dout[UDATA_W - 1 : 0]};
        end
    end
```

之后需要生成axi_lite写地址、写数据通道的有效指示信号，当FIFO读出数据后拉高，各自应答信号为高电平时拉低。

```
    //生成AXI写地址有效指示信号，与写地址信号对齐；
    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//当写地址信号被应答后拉低；
            M_AXI_AWVALID <= 1'b0;
        end
        else if(wfifo_rdata_vld)begin//当写地址FIFO输出有效数据后拉高；
            M_AXI_AWVALID <= 1'b1;
        end
    end

    //生成AXI写数据有效指示信号，与写地址信号对齐；
    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(M_AXI_WREADY)begin//当写数据信号被应答后拉低；
            M_AXI_WVALID <= 1'b0;
        end
        else if(wfifo_rdata_vld)begin//当FIFO输出有效数据后拉高；
            M_AXI_WVALID <= 1'b1;
        end
    end
```

最后就是写应答通道的主机应答信号，该信号可以在写入数据完成时拉高，也可以在写入数据有效时就拉高，没有特别的要求。但必须在从机应答有效后拉低。

```
    //生成应答通道的应答数据，当写入数据后拉高，等待从机有效指示信号拉高后拉低。
    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//从机应答信号拉高；
            M_AXI_BREADY <= 1'b0;
        end
        else if(M_AXI_WREADY & M_AXI_WVALID)begin//写入数据完成；
            M_AXI_BREADY <= 1'b1;
        end
    end
```

下面是与读通道相关的信号，读地址通道与写地址通道类似。首先依旧需要一个读状态的标志信号rd_flag，但是为了避免同时读写同一地址数据，此处需要在写FIFO为空的时候，才会拉高读状态。

```
    //读标志信号；
    always@(posedge M_AXI_ACLK)begin
        if(M_AXI_ARESETN==1'b0)begin//初始值为0;
            rd_flag <= 1'b0;
        end
        else if(M_AXI_RREADY & M_AXI_RREADY)begin//当读数据通道的应答信号和输出数据有效信号同时为高时，表示读取成功；
            rd_flag <= 1'b0;
```

```
        end
        else if((~ra_fifo_empty) && (~rd_flag) && wfifo_empty && (~wr_flag))begin//当读地址FIFO不为空且不处于读数据状态时拉高；
            rd_flag <= 1'b1;
        end
    end

    //读地址FIFO读使能信号，当需要同时执行读写操作时，优先执行写操作，防止同时读写同一地址引发错误。
    always@(posedge M_AXI_ACLK)begin
        if(M_AXI_ARESETN==1'b0)begin//初始值为0;
            ra_fifo_rd_en <= 1'b0;
        end
        else begin//当FIFO没有空且读标志信号开始时拉高；
            ra_fifo_rd_en <= ((~ra_fifo_empty) && (~rd_flag) && wfifo_empty && (~wr_flag));
        end
    end

    //将读地址FIFO读使能信号延迟一个时钟作为读出数据有效指示信号；
    always@(posedge M_AXI_ACLK)begin
        ra_fifo_rdata_vld <= ra_fifo_rd_en;
    end
```

因此注意，用户写使能不能一直拉高，会造成无法进行读操作。

读地址FIFO输出数据使能、axi_lite读地址、读地址有效指示信号与前文写地址通道信号类似，对应代码如下，不再赘述。

```
    //生成AXI读地址信号，初始值为0，当读地址FIFO读数据有效时，输出FIFO读出的数据，其余时间保持不变；
    always@(posedge M_AXI_ACLK)begin
        if(M_AXI_ARESETN==1'b0)begin//初始值为0;
            M_AXI_ARADDR <= {{ADATA_W}{1'b0}};
        end
        else if(ra_fifo_rdata_vld)begin//将读地址FIFO输出数据赋值给AXI地址信号；
            M_AXI_ARADDR <= {{{ADATA_W - UADDR_W}{1'b0}},ra_fifo_dout[UADDR_W - 1 : 0]};
        end
    end

    //生成AXI读地址有效指示信号，与读地址信号对齐；
```

```
    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//当读地址信号被应答后拉低;
            M_AXI_ARVALID <= 1'b0;
        end
        else if(ra_fifo_rdata_vld)begin//当读地址FIFO输出有效数据后拉高;
            M_AXI_ARVALID <= 1'b1;
        end
    end

    //生成读数据通道的应答信号;
    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_RVALID)begin//当读数据有效时拉低;
            M_AXI_RREADY <= 1'b0;
        end
        else if(M_AXI_ARVALID & M_AXI_ARREADY)begin//当从机应答读地址信号时拉高;
            M_AXI_RREADY <= 1'b1;
        end
    end
```

然后将axi_lite读出的数据和数据有效指示信号输出，对应代码如下所示。

```
    //生成用户读数据及读数据有效信号;
    always@(posedge M_AXI_ACLK)begin
        if(M_AXI_ARESETN==1'b0)begin//初始值为0;
            rdata <= {{UDATA_W}{1'b0}};
            rdata_vld <= 1'b0;
        end
        else begin//将AXI读出数据输出，并且把数据有效信号拉高一个时钟周期;
            rdata <= M_AXI_RVALID ? M_AXI_RDATA[UDATA_W-1 : 0] : rdata;
            rdata_vld <= M_AXI_RVALID;
```

```
        end
    end
```

最后会给用户生成一个错误指示信号，根据读数据和写应答通道从机反馈的状态信号是否为0，判断写数据和读数据是否正确，对应代码如下。

```
    //生成错误指示信号，初始值为0；
    always@(posedge M_AXI_ACLK)begin
        if(M_AXI_ARESETN==1'b0)begin//初始值为0;
            error <= 1'b0;
        end//每次读写使能信号有效时清零；
        else if(wfifo_rd_en | ra_fifo_rd_en)begin
            error <= 1'b0;
        end//当写入数据失败或者读出数据错误时拉高；
        else if((M_AXI_ARVALID & M_AXI_ARREADY & (M_AXI_RRESP!=2'd0)) | (M_AXI_BREADY & M_AXI_BREADY & (M_AXI_BRESP != 2'd0)))begin
            error <= 1'b1;
        end
    end
```

整个模块的设计就完成了，将用户端口时序转换为axi_lite接口协议。

3、axi_lite_master仿真

仿真有两种方式，第一种就是使用一个axi_lite从机模块与该模块一起仿真。第二种是使用一个axi_lite的IP作为从机进行仿真，使用IP需要阅读该IP手册，了解其内部寄存器，相对比较麻烦。

因此可以使用本文前半部分的方式，生成一个axi_lite的从机模块，作为仿真模型，生成从机的步骤省略，参考图9生成主机模块步骤即可。

编写的TestBench如下所示：

```
`timescale 1 ns/1 ns
module test();
    parameter   CYCLE       =   10              ;//系统时钟周期，单位ns，默认10ns；
    parameter   RST_TIME    =   10              ;//系统复位持续时间，默认10个系统时钟周期；
    parameter   UADDR_W     =   8               ;//用户地址位宽；
    parameter   UDATA_W     =   32              ;//用户数据位宽；
    parameter   ADATA_W     =   32              ;//AXI_LITE的地址和数据位宽；
    //时钟复位信号；
    reg                         clk             ;//系统时钟，默认100MHz；
    reg                         rst_n           ;//系统复位，默认低电平有效；
```

```
    //用户写数据信号；
    reg                         wr_en           ;
    reg   [UADDR_W - 1 : 0]     waddr           ;
    reg   [UDATA_W - 1 : 0]     wdata           ;
    //用户读数据信号；
    reg                         rd_en           ;
    reg   [UADDR_W - 1 : 0]     raddr           ;
    wire  [UDATA_W - 1 : 0]     rdata           ;
    wire                        rdata_vld       ;
    wire                        error           ;
    //AXI写地址通道信号；
    wire  [ADATA_W - 1 : 0]     M_AXI_AWADDR    ;
    wire  [2 : 0]               M_AXI_AWPROT    ;
    wire                        M_AXI_AWVALID   ;
    wire                        M_AXI_AWREADY   ;
    //AXI写数据通道信号；
    wire  [ADATA_W - 1 : 0]     M_AXI_WDATA     ;
    wire  [ADATA_W / 8 - 1 : 0] M_AXI_WSTRB     ;
    wire                        M_AXI_WVALID    ;
    wire                        M_AXI_WREADY    ;
    //AXI写应答通道信号；
    wire  [1 : 0]               M_AXI_BRESP     ;
    wire                        M_AXI_BVALID    ;
    wire                        M_AXI_BREADY    ;
    //AXI读地址通道信号；
    wire  [ADATA_W - 1 : 0]     M_AXI_ARADDR    ;
    wire  [2 : 0]               M_AXI_ARPROT    ;
    wire                        M_AXI_ARVALID   ;
    wire                        M_AXI_ARREADY   ;
    //AXI读数据通道信号；
    wire  [ADATA_W - 1 : 0]     M_AXI_RDATA     ;
    wire  [1 : 0]               M_AXI_RRESP     ;
    wire                        M_AXI_RVALID    ;
    wire                        M_AXI_RREADY    ;

    //例化主机模块
    axi_lite_master #(
        .UADDR_W        ( UADDR_W       ),
        .UDATA_W        ( UDATA_W       ),
        .ADATA_W        ( ADATA_W       )
```

```
    )
    u_axi_lite_master (
        .M_AXI_ACLK     ( clk           ),
        .M_AXI_ARESETN  ( rst_n         ),
        //AXI写地址通道信号;
        .M_AXI_AWREADY  ( M_AXI_AWREADY ),
        .M_AXI_AWADDR   ( M_AXI_AWADDR  ),
        .M_AXI_AWPROT   ( M_AXI_AWPROT  ),
        .M_AXI_AWVALID  ( M_AXI_AWVALID ),
        //AXI写数据通道信号;
        .M_AXI_WDATA    ( M_AXI_WDATA   ),
        .M_AXI_WSTRB    ( M_AXI_WSTRB   ),
        .M_AXI_WVALID   ( M_AXI_WVALID  ),
        .M_AXI_WREADY   ( M_AXI_WREADY  ),
        //AXI写应答通道信号;
        .M_AXI_BRESP    ( M_AXI_BRESP   ),
        .M_AXI_BVALID   ( M_AXI_BVALID  ),
        .M_AXI_BREADY   ( M_AXI_BREADY  ),
        //AXI读地址通道信号;
        .M_AXI_ARREADY  ( M_AXI_ARREADY ),
        .M_AXI_ARADDR   ( M_AXI_ARADDR  ),
        .M_AXI_ARPROT   ( M_AXI_ARPROT  ),
        .M_AXI_ARVALID  ( M_AXI_ARVALID ),
        //AXI读数据通道信号;
        .M_AXI_RDATA    ( M_AXI_RDATA   ),
        .M_AXI_RRESP    ( M_AXI_RRESP   ),
        .M_AXI_RVALID   ( M_AXI_RVALID  ),
        .M_AXI_RREADY   ( M_AXI_RREADY  ),
        //用户读写信号;
        .wr_en          ( wr_en         ),
        .waddr          ( waddr         ),
        .wdata          ( wdata         ),
        .rd_en          ( rd_en         ),
        .raddr          ( raddr         ),
        .rdata          ( rdata         ),
        .rdata_vld      ( rdata_vld     ),
        .error          ( error         )
    );

    //例化AXI从机模块
```

```
    axi_lite_slave #(
        .C_S_AXI_DATA_WIDTH ( ADATA_W   ),
        .C_S_AXI_ADDR_WIDTH ( ADATA_W   )
    )
    u_axi_lite_slave (
        .S_AXI_ACLK     ( clk           ),
        .S_AXI_ARESETN  ( rst_n         ),
        //AXI写地址通道信号;
        .S_AXI_AWREADY  ( M_AXI_AWREADY ),
        .S_AXI_AWADDR   ( M_AXI_AWADDR  ),
        .S_AXI_AWPROT   ( M_AXI_AWPROT  ),
        .S_AXI_AWVALID  ( M_AXI_AWVALID ),
        //AXI写数据通道信号;
        .S_AXI_WDATA    ( M_AXI_WDATA   ),
        .S_AXI_WSTRB    ( M_AXI_WSTRB   ),
        .S_AXI_WVALID   ( M_AXI_WVALID  ),
        .S_AXI_WREADY   ( M_AXI_WREADY  ),
        //AXI写应答通道信号;
        .S_AXI_BRESP    ( M_AXI_BRESP   ),
        .S_AXI_BVALID   ( M_AXI_BVALID  ),
        .S_AXI_BREADY   ( M_AXI_BREADY  ),
        //AXI读地址通道信号;
        .S_AXI_ARREADY  ( M_AXI_ARREADY ),
        .S_AXI_ARADDR   ( M_AXI_ARADDR  ),
        .S_AXI_ARPROT   ( M_AXI_ARPROT  ),
        .S_AXI_ARVALID  ( M_AXI_ARVALID ),
        //AXI读数据通道信号;
        .S_AXI_RDATA    ( M_AXI_RDATA   ),
        .S_AXI_RRESP    ( M_AXI_RRESP   ),
        .S_AXI_RVALID   ( M_AXI_RVALID  ),
        .S_AXI_RREADY   ( M_AXI_RREADY  )
    );

    //生成周期为CYCLE数值的系统时钟;
    initial begin
        clk = 0;
        forever #(CYCLE/2) clk = ~clk;
    end

    //生成复位信号;
```

```
    initial begin
        rst_n = 1;
        #2;
        rst_n = 0;//开始时复位10个时钟；
        #(RST_TIME*CYCLE);
        rst_n = 1;
    end

    initial begin
        wr_en <= 1'b0;
        rd_en <= 1'b0;
        waddr <= 0;
        raddr <= 0;
        wdata <= 0;
        wait(rst_n);//等待复位信号为高电平后；
        repeat(10)@(posedge clk);
        wr_data(13,133);
        repeat(10)@(posedge clk);
        rd_data(13);
        repeat(10)@(posedge clk);
        fork//并行运行写数据和读数据两个函数；
            wr_data(20,311);
            rd_data(20);
        join
        repeat(15)@(posedge clk);
        rd_data(20);
        repeat(20)@(posedge clk);
        $stop;
    end

    //用户写数据任务；
    task wr_data(
        input   [UADDR_W - 1 : 0]       addr    ,
        input   [UDATA_W - 1 : 0]       data
    );
        begin
            wr_en <= 1'b0;
            @(posedge clk);
            wr_en <= 1'b1;
            waddr <= addr;

```

```
            wdata <= data;
            @(posedge clk);
            wr_en <= 1'b0;
            @(posedge clk);
        end
    endtask

    //用户读数据任务：
    task rd_data(
        input   [UADDR_W - 1 : 0]       addr
    );
        begin
            rd_en <= 1'b0;
            @(posedge clk);
            rd_en <= 1'b1;
            raddr <= addr;
            @(posedge clk);
            rd_en <= 1'b0;
            @(posedge clk);
        end
    endtask

endmodule
```

运行仿真结果如下，当用户向地址13中写入133时，橙色信号是主机写地址通道输出的三个信号，天蓝色是主机写数据使出的三个信号，粉红色是从机输出该主机的写应答通道信号。

写地址有效信号和写数据有效指示信号同时拉高，在各自应答有效后拉低。而黄色信号是主机的应答从机的信号，此处在写入数据后拉高。

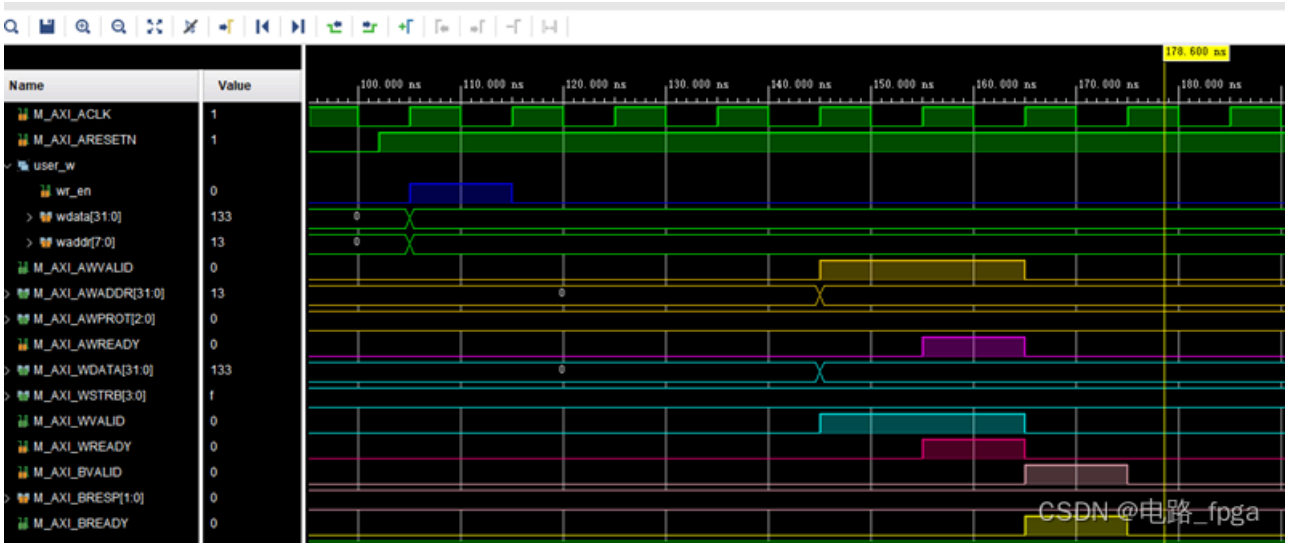

图11 写时序仿真

写时序仿真正确，下图是用户端口从地址13中读出数据。

白色信号表示读地址通道主机输出的三个信号，橙黄色表示读出从机的数据信号，而紫红色表示输出给用户的读数据。

最终从地址13读出的数据为133，与前文写入数据一致，证明读写时序均没有问题。

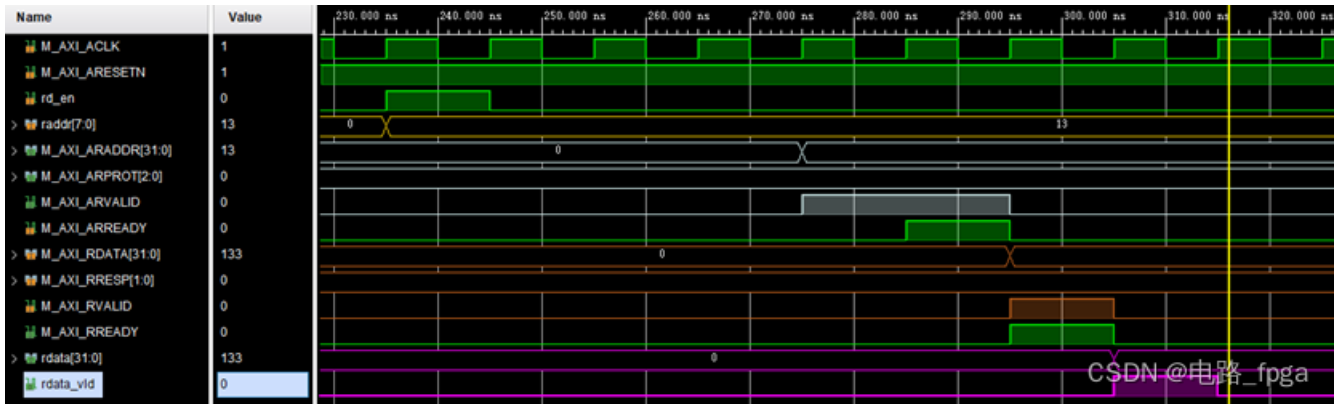

图12 读时序仿真

下图中用户端口同时读写从机的同一地址数据，根据模块的仲裁，首先会向地址为20的寄存器中写入数据311，然后在从地址20中读出数据，最后读出数据也为311。

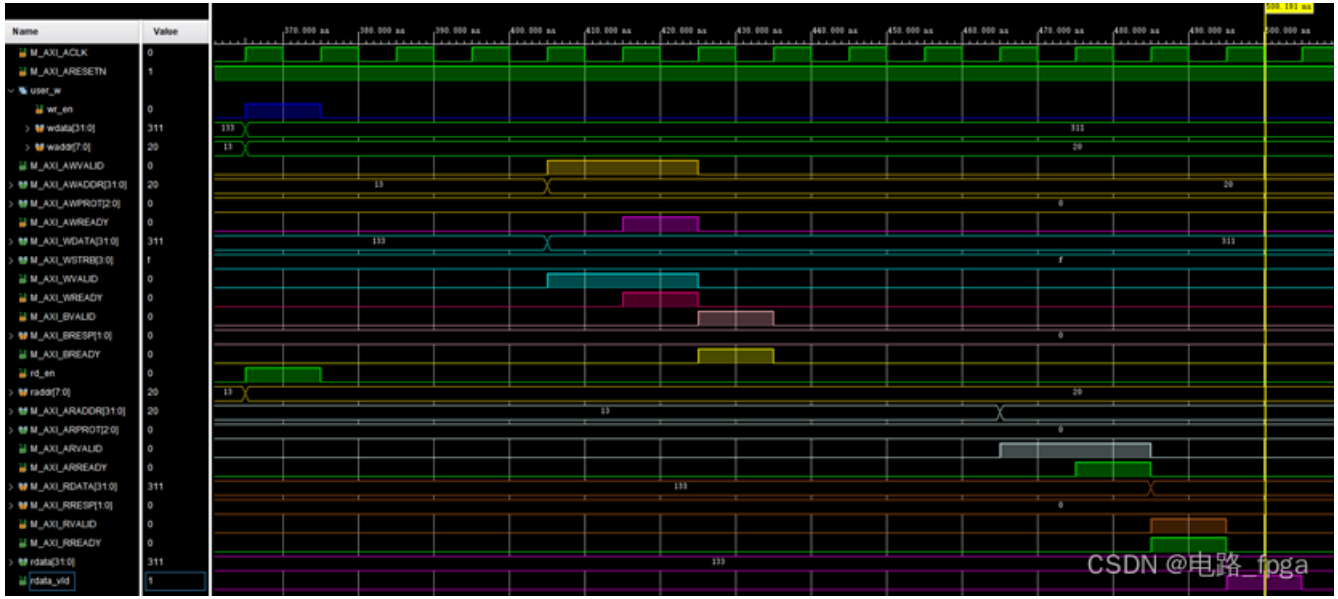

图13 用户端口同时读写同一地址

上图中axi_lite的读写时序中间有段间隔，这是由于中间判断信号造成的。

关于该模块的仿真到此结束，通过前文的讲解，实现axi_lite协议其实难度不大，相对SDRAM这些接口，axi_lite算是很简单的协议。

由于FPGA内部一般不会使用从机模块，因此先跳过从机的实现，下文将讲解axi_full接口协议，会发现axi_full其实也比较简单，很多信号和模式FPGA并不会使用。

需要本文工程的可以在公众号后台回复“**axi_lite_master工程**”（不包括引号）。

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！