本例子与第一个例子不同，本例子的testbench是使用verilog编写的，因此仿真无法使用cocotb进行，只能使用vivado仿真，testbench文件为：axi\_rw\_sim.v，此外，模块也比较复杂，由一个顶层模块和两个被调用的子模块组成，每个模块都写在单独的.v文件中，本例子还使用了.xci文件（赛灵思的ip）

对top文件的阅读：

在top文件中将各个小模块进行联合配置，顶层模块用于测试DDR（双倍数据速率存储器）的AXI接口。在顶层模块中首先定义了一系列参数，这些参数用于配置DDR AXI接口的各个参数（符合AXI协议的参数）。在参数定义之后，顶层模块声明了输入和输出端口，其中包含时钟信号clk和复位信号rst，以及和DDR相关的DDR接口时钟信号，这些输入信号是将这个由多个小模块组成的顶层模块封装之后所需要的输入信号。在顶层模块的输入输出端口定义完成之后，开始定义模块内部结构。首先定义4个通道：s0\_axi~s3\_axi，为每个通道定义一系列与AXI接口相关的信号比如：地址、数据、读写控制信号等。

之后定义了一组前缀为M00的信号：M00\_AXI\_AWID等。M指的是AXI协议中的主机，

在M00信号之后，为每个通道定义了控制信号和标志信号：s0\_test\_en和ddr\_init\_complete

在这之后，使用axi\_qos信号表示不同请求的优先级，在例子中，所有qos信号都被赋值为4’b0000，因此没有优先级设置。

之后通过实例化axi\_rw\_test\_noblock模块为4个通道实例分别配置了axi，用于执行AXI接口的读写测试。

axi\_rw\_test\_noblock #(

.AXI\_DATA\_WIDTH ( DDR\_DATA\_WIDTH ),

.AXI\_ADDR\_WIDTH ( DDR\_ADDR\_WIDTH ),

.AXI\_LEN\_WIDTH ( DDR\_LEN\_WIDTH ),

.AXI\_ID\_WIDTH ( DDR\_ID\_WIDTH ),

.AXI\_CACHE\_WIDTH ( DDR\_CACHE\_WIDTH ),

.AXI\_PROT\_WIDTH ( DDR\_PROT\_WIDTH ),

.AXI\_BRUST\_WIDTH ( DDR\_BURST\_WIDTH ),

.AXI\_STRB\_WIDTH ( DDR\_STRB\_WIDTH ),

.AXI\_RESP\_WIDTH ( DDR\_RESP\_WIDTH )

) axi\_rw\_test\_noblock\_s0\_inst (

// 连接信号

);

这种格式的实例化头一次见，其实也是很正常的，前面这带#的一大坨是在对模块进行参数实例化，这种实例化方式用于对带有参数化功能的模块进行实例化。

之后定义了一个axi\_interconnect\_diamond模块，该模块用于连接四个AXI接口，该交叉连接器是一个4\*4的矩阵，每个接口都与其他三个接口相连接，实现四个接口的全互联，这样子四个AXI接口之间可以相互通信，实现数据的交换和传输

在最后对axi\_ram模块进行实例化，这个模块同样是带有参数化功能的模块。通过对模块内参数进行实例化，以及对模块输入输出信号进行连线，实现对RAM存储器的读写操作。这个模块实例化过程使得ram存储器集成在top模块中。

总体的top模块结构是，外部有三个控制信号输入，内部通过一个interconnect模块连接四个通道，每个通道通过axi\_rw\_test\_noblock进行读写，同时模块内部配置了ram。其中interconnect是通过ip核来实现的，目前可以不用看。需要看的是top文件和读写模块文件

目前不了解的是M00前缀的信号是干嘛的

M00前缀的信号是给主机定义的axi接口，具体的结构是一个主机使用axi协议从ram中读取数据，一个主机和三个从机通过一个interconnect相连接。

具体的逻辑我目前是这么理解的：axi协议中主机和从机更倾向于是主机和设备，即主机和存储器这种方式，主机从存储器中读取数据。对于整个模块，M00主机从RAM中读取数据，然后往interconnect中写入(这一步不是很理解）

在查看了vivado中跑出来的原理图之后，对这个模块有进一步直观的理解：顶层模块定义的那么多信号，包括4个通道和一个m00通道信号，其实都是按照axi协议接到interconnect上的，这些信号都是为了interconnect模块定义的，然后m00接到ram上，四个通道分别接到4个非阻塞读写模块中，axi协议的读写和主从一定是按数据传输方向确定的，因为数据传输是单向的。主从的确定会帮助定传输两端模块的信号

对AXI4协议的学习：

几个关键点：

主机在复位之后的下一个时钟上升沿便对valid置高

在主机向从机写数据的过程中：主机不能够等待从机给出ready信号之后再给出valid信号（无论是AWREADY还是WREADY），在写数据的过程中，valid要先置高，等到ready也置高了之后便可以开启数据发送。

在主机读取从机数据的过程中，valid和ready的置高顺序没有固定的先后规定。

协议中规定的五个通道应该是指把各种信号合在一起的总线，而单向指的是数据单向传输，控制信号还是需要双向传输，方便理解的话也可以把控制信号单拎出来，把这些所谓的五条通道理解为五个分组就可以

另外，vivado要查看波形图的话，需要把实例化的模块加到波形图里面，实例化的模块要在scope窗口里面找

对波形图进行分析：

1. s0对应的读写模块的波形：

写地址通道信号及其握手过程

1. awid信号（用于区分不同写请求的标识符，每个写事件都有唯一的awid值，主机在每一个写事件开始时生成一个新的awid值，并在整个写事件中保持不变），在波形图中始终保持为b’0000，没有对不同次的写入事件起到区分作用，因为我们没有用到这个信号（比较高级的功能）
2. awaddr信号（用于传输写事件中的目标地址的首地址信息，由主机向从机发送，从机收到首地址之后根据其他信息确定地址范围）在波形图中一开始是h’0000

在awvalid与awready信号握手成功之后的下一个时钟上升沿要传输的首地址变为h’0400，在下一次握手成功后的下一个上升沿变为h’0800，在第三次握手成功之后的下一个上升沿变为0c00，每次首地址之间相隔1024，如果地址的粒度为字节，那么就每次相隔1024字节

1. awlen信号（表示突发传输事件中要进行多少个周期的数据传输）在开始时为h’00，在awlaid第一次置高时变为h’0f（表示每次进行awlen+1（15+1=16）个周期的传输）并在之后不再变化。
2. awsize信号（表示突发传输事件中每周期内传输的数据量，计算方式为：每周期传输数据大小为2^(awsize)字节）在波形图中一开始为0，在awvalid第一次置高是变为6（每周期传输2^6=64个字节）并不再变化，和之前的awlen以及之后的awburst信号配合，可以知道每次burst传输多少字节的内容

AWSIZE[2:0]：表示一个 transfer 中的 Bytes 个数。 3'b000=>1 ，3'b001=>2 ，3'b010=>4 ，3'b011=>8 ，3'b100=>16，3'b101=>32 ，3'b110=>64 ，3'b111=>128

1. awburst信号（细分的突发传输类型控制/标识信号，2bit信号定义了4种不同的细分的burst类型：00固定burst；01递增burst；10包级burst；11保留（目前不用））在波形图中是先0，再第一次awvalid置高时变为1，且不再变化，表示在整个burst事件中是递增burst类型（每次写事件的地址逐渐递增，每次写事件开始时在上一个写事件的地址基础上增加一个固定的偏移量，以便在连续的地址空间中写入数据。）
2. awlock信号（锁定传输标识符，高电平时表示写地址传输是锁定模式，低电平时非锁定，一般都是非锁定）波形图中一直是0
3. awcache信号（主机发送给从机，用于描述要写入的数据的缓存属性，是多比特信号，通过每一比特位上信号的高低电平确定不同的特点）波形图中刚开始先是0，在awvalid第一次置高时变为3（b’0011），并不再变化

注释：

awcache[3]：Bufferable（可缓存）。当为1时，表示该数据是可缓存的，从设备可以将数据存储在其内部缓存中，以便后续更快地读取。当为0时，表示数据不能被缓存。

awcache[2]：Cacheable（高速缓存）。当为1时，表示该数据可以被高速缓存，允许主设备将数据存储在高速缓存中以加快访问速度。当为0时，表示数据不能被高速缓存。

awcache[1]：Write-Allocate（写分配）。当为1时，表示写入操作需要先将数据读取到缓存中，然后再进行写入。当为0时，表示写入操作直接写入到主存中。

awcache[0]：Read-Allocate（读分配）。当为1时，表示读取操作需要将数据读取到缓存中，然后再传送给主设备。当为0时，表示读取操作直接从主存中读取。

1. awprot信号（多比特信号，每一比特的高低电平表示不同的保护属性）从始至终一直为b’000

注释：

awprot[2]：Privileged（特权）。当为1时，表示该写入操作是特权级的，拥有更高的访问权限。当为0时，表示写入操作是非特权级的。

awprot[1]：Secure（安全）。当为1时，表示该写入操作是安全的，即来自可信源。当为0时，表示写入操作是非安全的。

awprot[0]：Data Cacheable（数据可缓存）。当为1时，表示该数据可以被缓存。当为0时，表示数据不能被缓存

1. awvalid信号（写地址传输请求的有效信号，高电平表示主机在当前时钟周期内发送了有效的写地址信息）在波形图中：刚开始为0，在某一个时钟上升沿变为1且不再变化
2. awready信号（）波形图中刚开始为0，之后三次置高（有三次写地址事件握手成功）

写数据通道信号及其握手过程

1. wdata信号（512位写地址信号）在波形图中：在一开始以及没有握手成功的时间段内，512位信号线均为0（初始化的数值）在wready信号置高即写数据通道握手成功之后的下一个时钟上升沿开始，发送了16个周期的数据，每个时钟周期发送512比特的递增数据（从0开始，第16个周期发送数据为15），总共有三次。在最后一个写事件时，写地址握手正常，但是写数据的ready信号频繁跳变，但是依然最终完成了16个周期内的数据递增传输，和之前正常的情况一样。

综合分析一下写入过程：写地址通道握手成功，传入目的地址首地址，并通过突发控制信号：awlen和awsize调整突发传输的规则即每次突发传输事件持续16周期（传输16次），每周期传输64字节（512位）数据，在首地址传好，写数据通道握手成功之后，按照这个规则传输数据，传输数据的值递增，每次突发传输的传输数据量是16\*64=1024字节，两次突发传输之间的首地址偏移量也为1024字节（要注意，看握手信号成功握手那一拍或者说那个上升沿对应的是什么数据，传输的就是什么数据）

1. wstrb信号（写入数据字节有效信号64位）波形图中最开始时是0，在awvalid第一次置高之后变为全1，并保持不变（表示输入信号的每一拍输入的64字节都有效）
2. wlast信号（单比特信号，用于指示当前传输的数据是否是写数据事件中的最后一个数据（一般指最后一拍数据））在波形图中，每一次写入事件都对应一个wlast
3. wvalid信号（写数据握手信号）在波形图中，先是0，然后在某一个时间点直接拉高，从此不再往下拉
4. wready信号（写数据握手信号2）在波形图中，周期性置高，在两个写数据事件之后ready信号变得不稳定

写响应通道：

1. bid信号（多比特信号，用于关联写响应与具体的写数据事件，按理来说bid应该与awid匹配）波形图中bid一直是高阻态，感觉有问题
2. bresp信号（写响应信号，从设备对写入传输信息分析之后传回的信号，2比特，表示四种不同情况。00写入成功；01写入成功，且对于某些字节进行了额外处理；10写入失败，从机出现错误；11写入失败，从机无法解析主机提供的目标地址）在波形图中始终为00，说明一直都是写入成功的正常状态。
3. bvalid信号（从设备向主设备发送的表示写响应有效的信号）波形图中在每次写数据事件之后，下一次写数据事件之前，bvalid信号拉高，写响应通道握手成功，主机读入bresp信号。
4. bready信号（主设备向从设备发送的写响应通道握手信号）在波形图中和上面的主机控制信号一样，都是刚开始时是0，然后在某一拍置高。

读地址通道（主机向从设备读数据的过程）

1. arid信号（读地址id）波形图中从头到尾都是0(与之前说的一样，目前不用这个)
2. araddr信号（读地址信号）波形图中先是0，然后从当时往里面写的地址逐个往外读。
3. arlen信号（读事件中突发传输的传输次数）波形图中刚开始时是0，后来始终是0f，也就是说在所有的读事件中每次突发都写15+1=16次。
4. arsize信号（读事件中burst中每次（每周期）传输的字节数，这个和之前一样，都是取决于布线上数据位宽来定义的，512位，64字节）
5. 后面有几个控制信号和前面说的是一样的，因此就不说了
6. arvalid信号（读地址事件中的握手信号）在写响应结束之后的下一个上升沿置高
7. arready信号（读地址事件中的另一个握手信号）波形图中该信号每隔一段时间置高，读地址通道握手成功，开始从araddr信号线上读取首地址

读数据通道

1. rid和之前一样
2. rdata信号（读数据信号线），在每次读事件中（读地址+读数据）首地址读完之后会在这个地址上读16\*64=1024个字节。
3. rvalid信号和rready信号和上面一样
4. rlast信号，用rlast信号在读数据信号线连续给出数据时去区分每次读事件，因为每次读事件的地址是不一样的
5. rresp信号是读数据通道中给出的一个读取响应信号（标识读事物的状态：00表示成功）波形图中全程00

需要注意的一点是，我们在工作中需要掌握的是，我们清楚掌握了axi协议的信号作用和连接方法，在使用中我们需要做的是将信号线（如握手信号等）按照协议规定的方式进行连接，并且通过状态机对状态进行转换，协议的工作原理通过定义好的协议ip核进行细节操作的定义和实现。

另外对有一点有了一些进一步的理解：就是为什么声明的信号线wire，在模块中还要使用相同名字的reg，还要再把它们接起来，这样是不是有点多此一举。

从verilog语法角度理解：在always模块中，被赋值的只能是reg类型，因此在模块中进行处理时，需要使用reg当作媒介，最后再接到wire上输出模块。

但是从电路角度还是不很理解：wire需要电平变化来传递信息等功能的实现，而我们是通过给寄存器写值的方式去驱动wire上电平的变化。

在ddr读写中，还差一个检测完毕点灯的逻辑，可以从最外层的模块的输出线led一层一层往里面接线，接到我的读写模块中，在读写模块中接出调试信号，写一个逻辑然后把led接到调试信号输出上，这样子就可以根据调试结果点灯。之后再重新生成一个bit文件，再往板子里烧写

点到点的高可靠传输

高可靠传输的调研报告，看论文，看综述

归纳方法，进行分类，将各种方法对比一下，分析优劣势，然后引用文献。

对着原理图配置mig，

Strb id 信号的验证工作