**AXIStream协议学习报告**

1. AXIStream协议概述：arm公司提出的一种用于高速数据流传输的串行通信协议，我们主要使用这种协议进行数据包的传输。

A．数据通路和信号：

1.1数据总线tdata（多位宽）：该协议使用一个单一的数据总线来传输数据，数据总线可以是不同数据位宽的位向量，在工程所给示例中数据总线位宽由parameter DATA\_WIDTH = 128定义为128bit，在示例中的数据总线通过。。。定义，一般定义为tdata

1.2控制信号线：

1.2.1 tvalid信号（1bit）：握手信号之一：master主机发送给从机，表示数据传输有效，主机已经做好数据发送准备

1.2.2tready信号（1bit）：另一个握手信号：slave从机发送给主机，表示从机做好数据接收准备

握手过程：当数据准备完毕之后，主机将tvalid信号置为高电平，时钟信号到达上升沿时会对控制信号进行采样，当采到tvalid为高电平时，说明主机已经准备好发送数据，时钟的上升沿同样会对从机的tready信号进行采样，当采到高电平说明从机准备好接受数据，这时主从机握手成功，数据传输开始。

1.2.3tlast信号（1bit）：数据包边界信号，通过tlast信号来标识数据包的结束，当tlast信号为高电平时，表示当前数据包传输结束

1.2.4tkeep信号（多位宽）：用来标识tdata有效字节的信号，宽度为tdata/8(因为tdata时bit单位，tstrb为字节单位)其中为1的bit对应tdata的有效字节

1.2.5tuser信号（灵活位宽）：用户定义信号，其含义用法位宽都由用户自行定义。

B.数据传输方式：以数据包形式进行传输，每个数据包由数据字节组成，该协议不涉及地址寻址和读写控制，数据传输在发送方和接收方的匹配是通过时钟同步实现的

2.对代码深入的理解：

首先是对整体模块的理解：这是一个适配于AXIStream协议的interconnecter模块。首先看模块的参数和端口定义部分，定义了数据总线位宽为128bit，定义了tkeep（数据有效字节标识信号）的位宽为数据位宽/8，定义了tkeep的使能信号，定义了用户自定信号的位宽

在端口定义上：同时定义了从机和主机的输入输出端口，本模块中从机的输入输出端口面向数据传输链路中上一个主机的端口，与之进行握手，本模块中主机的输入输出端口面向数据传输链路中下一个从机的端口，与之进行握手并传输数据。根据上述协议中规定的数据总线和控制信号的规则进行端口定义。

中间定义了stready和stready\_early的信号以及与之相对应的寄存器

由于wire信号是电路连线，并没有记忆能力，因此需要使用寄存器对控制信号进行寄存，因此下一块寄存器声明代码主要用来对模块中主机部分的数据总线以及各路控制线配置对应的寄存器并进行初始化。并为这些寄存器声明了对应的temporary即缓冲寄存器，用来对信号与数据进行暂存。

在此之后声明了store\_axis\_int\_to\_output; store\_axis\_int\_to\_temp; store\_axis\_temp\_to\_output;

三个寄存器，用来存储模块中间的值（注释上说是数据路径控制，其实就是定义不同的情况标识）

之后通过assign语句将主机部分的数据线与控制信号和对应的寄存器进行了接线

接下来是关键语句的理解：

目前理解的是s\_axis\_tready\_early有点像s\_axis\_tready的下一状态next，在rst不为0的情况下将会比s\_axis\_tready延迟一个时钟周期，用于确定本模块的从机部分是否准备好接受数据（ready），而且通过这一行语句，可以实现在握手的期间保持s\_axis\_tready为高电平。

每次通过判断条件，去修改s\_axis\_tready\_early，而s\_axis\_tready\_early的值会在下一个上升沿赋给s\_axis\_tready，如果一直满足传输条件，则s\_axis\_tready可以一直保持高电平，维持握手状态，如果不满足条件，s\_axis\_tready\_early将会变成0，并在下一周期将s\_axis\_tready拉低。

assign s\_axis\_tready\_early = m\_axis\_tready || (!temp\_m\_axis\_tvalid\_reg && (!m\_axis\_tvalid\_reg || !s\_axis\_tvalid))

按语句表面意思理解的话：

s\_axis\_tready\_early 在以下几种情况下将会置为高电平：

1. m\_axis\_tready判断语句即模块的主机部分输入的tready信号为1，这代表着与模块的主机部分相连的链路上的下一个模块的从机部分输出的tready信号为1，即下一个模块的从机已经准备好接受本模块主机发送的数据了。
2. temp\_m\_axis\_tvalid\_reg寄存器为0，且m\_axis\_tvalid\_reg为0，且s\_axis\_tvalid为1
3. temp\_m\_axis\_tvalid\_reg寄存器为0，且m\_axis\_tvalid\_reg为1，且s\_axis\_tvalid为0
4. temp\_m\_axis\_tvalid\_reg寄存器为0，且m\_axis\_tvalid\_reg为0，且s\_axis\_tvalid为0

（没怎么理解好）

只要接收端准备好接收数据（m\_axis\_tready 为真，数据链路打通），或者前一个周期中没有输入数据且当前周期中没有输入数据（temp\_m\_axis\_tvalid\_reg 为假，并且 m\_axis\_tvalid\_reg 为假 或者 s\_axis\_tvalid 为假），那么在下一个周期就能使能输入数据

下面一块always语句块（作用是分出不同的握手状态）本质上是一个有限状态机

通过先前定义的m\_axis\_tvalid\_next寄存器装载主机valid信号的下一状态，然后在每次循环开始时将本状态赋给下一状态进行初始化。temp\_m\_axis\_tvalid\_next也是和上述是一样的。

然后对s\_axis\_tready\_reg即从机ready信号进行判断，如果从机信号准备好接受输入数据则：如果m\_axis\_tready=1即主机接收到下一从机的准备就绪信号或者m\_axis\_tvalid = 0即主机部分并未准备好向下一从机发送数据，在这样的条件下，主机的valid信号的下一状态被赋值为从机接收到的valid即上一个主机的valid

这么做的原因：上述两层判断分别确定了本模块的从机准备好接受数据（ready=1），以及下一模块的从机准备好接受数据（ready=1）或者说本模块的主机还没有准备好数据，这条数据链路上的握手还没有完成：上一个模块的主机还没有给出valid信号，如果上一模块给出valid信号，那么本模块和上一模块握手成功，同时本模块的主机向下一模块给出valid信号，这代表数据以及能够被读进当前模块并且准备好传输到下一模块中。如果上一模块没有给出valid信号，那么本模块也不会给出valid信号，状态量被改变，跳转到下一个always块中的发送数据的分支，但是此时本模块的tkeep信号被上一模块的tkeep信号赋值，如果上一模块没有准备好valid，那么tkeep信号会全为0，数据即便被发送也会被声明是无效的

如果本模块输出还没有准备好，那么先把输入数据读取到缓冲区中，同时把上一模块的主机valid状态赋给中间变量。

如果本模块的输入尚未准备就绪，但是主机输出已经收到准备就绪的信号，则把缓冲区的主机数据就绪信号赋值给主机数据就绪信号的下一状态。同时更改其他状态值。（并不太懂）

这一段代码完成的是状态和标志量在不同条件下的改变，是为下一段代码服务的。（这段代码也是认为上一模块的主机已经准备好传输数据）

在下一段加入时序的always语句块是整个模块的真正逻辑。（其实结构上也相当于是交换机）

首先是复位模块，如果复位，则将会将本模块主机部分和从机部分的握手信号都初始化为0，并将主机的握手信号valid的缓冲值初始化为0。如果初始化信号不为0，即正常工作状态，则每经过一个时钟上升沿都将握手信号的状态替换为下一状态，通过这样去不断检测握手信号的高低电平。

此外，每经过一次时钟上升沿，根据先前定义的几个中间状态量，对信号进行接线：如果输入输出均已就绪：则将主机部分输出的数据线接到从机部分输入数据线上，直接打通数据总线，其他控制信号也是如此，直接打通，如果输入就绪，输出尚未就绪，则将各路信号进行寄存，如果输入未就绪，但输出就绪，则将信号从缓冲区拉到输出

最后是结合这个例子去说明为什么axistream协议可以很好地完成数据包转发任务。要回答这个问题，首先要明确数据包转发任务的特点和要素：1.需要对数据包进行解析，即对包头、标识字段、界定字段等进行解析，这些工作会占据一定的开销，2.路由决策、3.转发操作、4.流量控制、5.错误处理、6.QOS支持，7.路径选择。在底层通信协议层面上涉及到的一般是1.数据包解析、2.流量控制。这两个功能都可以被axistream协议很好地完成。因为axistream协议仅关注连续的数据流传输，而不涉及复杂的地址寻址问题以及界定问题，通过tlast等控制信号去标识数据包的边界，通过同步时钟完成数据传输的匹配，可以很大程度上减少数据包解析的开销。同时连续的数据流方式可以适应各种大小的数据包，另外tuser的用户附加信息也可以为数据包传输提供额外的灵活性，比如标识数据包优先级、错误检测等。而数据包的流量控制也可以通过握手信号实现。在本示例中，

那个灵魂代码就实现了流量控制的功能，考虑到了缓冲区可以缓冲数据量的大小，去控制握手信号的电平，从而实现流量控制。

正常情况下后面的模块都可以ready，最好不过，本模块也继续ready如果一旦后面的模块不ready了，怎么办，如果tmp\_valid为0，证明temp里面没有数据，同时m\_valid也为0，证明没有可以接受数据，那么可以继续ready，相当于是把ready信号提前一拍给出来，代价就是需要一组寄存器去存储这一拍带来的数据，存在temp里

反过来说，只有引入了缓冲的tem寄存器来暂存，我们才能提前一拍知道ready的情况。

那么有两个问题：

1. 如果不能提前知道ready的情况，会怎么样，作为中间模块，会不敢给前面一个模块ready信号，从协议上来说，valid信号只要有数据就一直置高，因此握手成功的关键在于ready信号。Ready信号一定要准确给出。
2. 目前提前一拍知道了，但是引入的temp\_reg也只能缓冲一拍，如果多于一拍还是会承受不住。作为流水线上的中间模块，如果下面的模块承受不了这么多数据，向本模块给出被压信号，则本模块停止向下一模块传输数据，并将数据存到temp里面，本模块temp存满了，无法继续接受数据，则向上一模块给出被压信号，被压信号一路向上传递，用大容量FIFO来处理这些数据。

有个问题：

Valid信号的高低电平是怎么处理的。作为中间模块，data是128bit并行输入的wire型数据，没有记忆能力，就是电路上保持一拍的电平信号，那valid是不是在某个时钟上升沿，检测到data数据线上出现电平，识别到数据，就把valid置为高电平，m\_valid为1，那么按这样子说，m\_valid为0是不是就代表着此时模块的128bit的data线上没有数据等待读取或者传送到下一模块。