# Datamover(S2MM)调试总结

## 基本介绍

### ****简介****

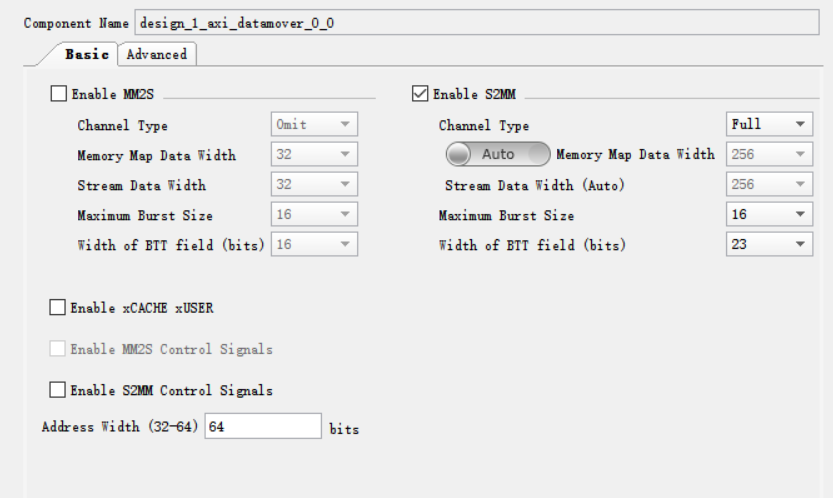
**Datamover主要用于AXI\_Stream和AXI4这两种总线之间数据的传输和完成协议的转换. Datamover由MM2S和S2MM两个相互独立的子模块组成,分别负责AXI4写数据给AXI\_Stream和AXI\_Stream写数据给AXI4.**

**AXI\_Stream总线优点是传输速度快,不需要起始地址且不限制Burst传输数据量的大小,但不适用于内存映射(Memory-Mapped)的设备(如DDR); AXI4是面向地址映射的接口，允许最大256Byte的数据Burst传输；因此为了将AXI\_Stream转换成AXI4总线,要给出Datamover地址和设置最大Burst传输大小,分别通过给Datamover发送命令和Customize IP中.**

**注:** **Burst在字典中译成“突发”,但不易理解；我一般把它译为“成组”**

**本次设计是为了将数据高速传输给DDR,需要用到Datamover 的 S2MM模块,接下来的论述也将围绕S2MM模块展开.**

### ****配置信息(customize block):****



1. 仅使用S2MM模块时,勾除“**Enable MM2S**” .
2. **Channel Type**:高性能模式选择为”**FULL”**(Basic模式适用于资源限制重于性能要求时).
3. **Address Width**: 为地址位占用命令”**s\_axis\_s2mm\_cmd\_data”**中的位宽.

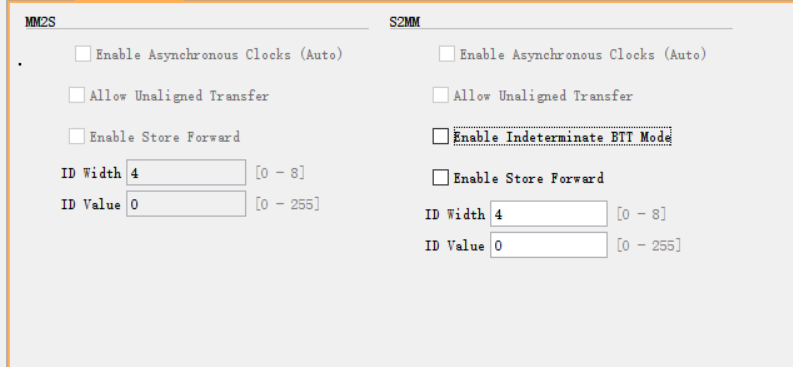
当为64bits时, **“s\_axis\_s2mm\_cmd\_data”共104位,为32bits时共72位.**

1. **Width of BTT field**: “**s\_axis\_s2mm\_cmd\_data”**中BTT(Bytes to Transfer )的有效位宽.

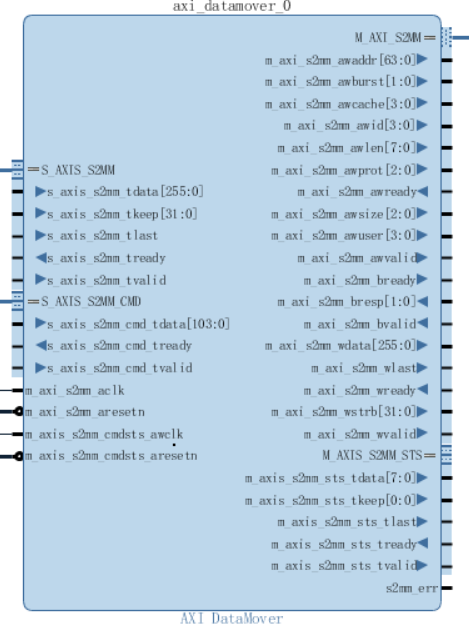
默认16,发送数据不超过2^16 bytes即可保持默认16位.

1. **Maximum Burst Size :** 即单组数据的最大传输数据量.如果设置为16,则单个Burst传输的数据最大为16个(共占位宽为16\*Stream Data Width).
2. **Advanced**: “Enable IBTT”和 “Store Forward” 只能选择一项,

当发送数据量已知时,选择”Store and Forward”;未知发送数据量时选择”Enable IBTT”.



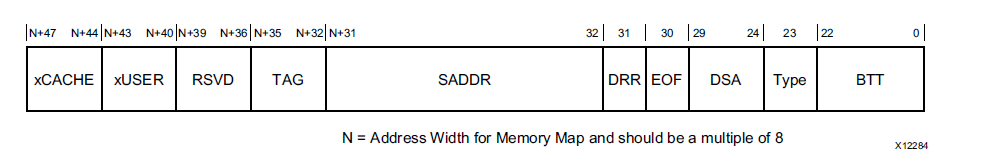
### ****S2MM模块的重要端口:****



\*\_valid和\*\_ready**: 分别是”s\_axis\_s2mm\_valid””s\_axis\_s2mm\_ready”和”s\_axis\_s2mm\_cmd\_valid”“s\_axis\_s2mm\_cmd\_ready”两组握手信号,分别负责数据和命令传输过程中的握手.**

**Master发送数据或命令给Slave时,需要将valid置1表示数据有效.Slave设备置ready信号为1表示可以接收数据和命令.一般情况下Datamover的ready都一直为1. 传输的命令和数据在\*\_valid 和\*\_ready 同时为高时有效.**

**\*\_cmd\_data:主设备发送的命令给从设备表示一个burst传输的开始,命令的格式如图所示:**



**其中N为地址的宽度(即在Customize block 中”address width” 的值),命令中主要的注意为”BTT”和”EOF”:**

**EOF表示为end of frame 当发送最后一组(Last Burst)数据的时候应当置1,容易弄混的是跟tlast信号.tlast信号是发送最后一组(Last Burst)数据的最后一个数据(Last Data Beats)时置1,EOF是最后一组数据时候置1.**

**举例:如果第一组数据发送15个数据,第二组数据发送30个数据,tlast在第二组的第30个数据发送时给出.EOF在第二组数据发送之前的命令中置1.**

**BTT表示传输数据的BYTE数.在Customize block中默认是为16位有效,当用户超过2^16数据时,需要设置更高的位数,最多23位.**

# ****在Advanced选项中传输模式分两种:IBTT和 Store and Forward****

## 传输模式(store and forward )**:**

1. **传输单组(Single Burst)数据的时候:**

状态机为:Idle(空闲) ---Sendcmd(发送命令)---Wait(等待)---Senddata(发送数据)---Idle(空闲).

**注意**:此处的Wait 应当是5个Clk以上.

1. **传输多组(Several Burst)数据(以两组为例):**

Idle(空闲)--Sendcmd1(发送第一组命令)—Wait(等待1)—Senddata1(发第一组送数据)—Wait(等待2)—Sendcmd2(发送第二组命令)—Wait(等待3)—Senddata2(发送第二组数据)—Idle(空闲)

#### 注意事项:

a.发送命令和数据之间(等待1和等待3)应当约有10个周期的延迟,当延迟小于7个周期时会有数据丢失的情况,当少于5个周期则完全无法传输数据. 两次独立的数据发送间隔(即等待2)可以为1个CLK.

b.在非IBTT模式下,如果命令中把EOF置为1 ,则代表这是发送的最后一组数据,接下来datamover将不会接收发送来的数据.因此EOF只能在最后一组数据的命令中为高.

## 传输的模式(IBTT):

相对于存储与转发的模式的BTT数量等于发送的实际数量,IBTT模式下的BTT数量是要大于实际发送数量的(如果小于则只发送到BTT规定的数目).

IBTT传输的关键在于每一组数据都要给出tlast信号.当传输两组独立的数据,第一组数据发完是要给tlast信号的,不然第二组的数据就会接着发送到第一组命令中声明的地址上去.(EOF在这里不是关键,0和1都可以正常工作.)

IBTT模式的改进(重要):

如果在知道接下来发送几组数据的地址的情况下,可以不用sendcmd1-senddata1-sendcmd2-senddata2这样的模式,可以使用sendcmd1-sendcmd2-senddata1-senddata2 这样的状态机,唯一注意的是需要在senddata1后面再加上tlast信号. 当这样传输数据会减少一次等待的时间(约10clk).

# 调试中遇到的问题及解决

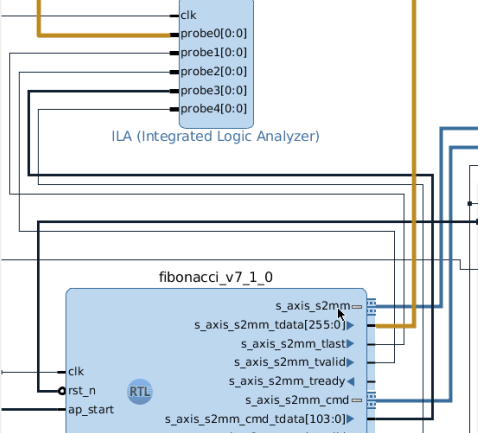
### ****使用ILA去监测Datamover 无法收到Trigger的情况.****

原因1: 自身代码有问题,

解决: 建议使用功能仿真来看波形图

原因2: 当模块使用ILA与输出信号分别连接(Native模式),与Datamover的连线又采用AXI总线式,这样会导致总线传输的数据为0,实际模块发送的数据不经由总线

解决: ILA检测输出端口和模块间连接端口的模式应统一,要么都是总线式连接,要么都是分别(Native)连接.



### Datamover接收数据缺漏的情况:

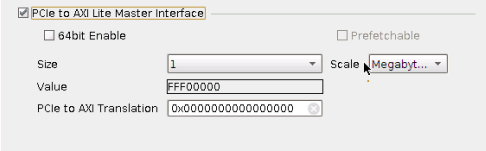
原因:模块内各个信号的传输存在延迟(Latency ),如果延迟不满足,就会存在信息缺漏.

解决办法:增大命令和数据之间的延迟.

### PCIE无法加载的情况:

原因:Datamover模块中遇到的Xdma0的Axi\_lite输出端口悬空的话会使得无法载入PCIE

解决:只需要在Customize block 选项中 勾除掉下列选项即可:



# 这次经历提醒了工程管理的重要性:

每一次调试都要在工程日志上写上具体的思路,当工程不知道哪里错误的时候,只能一条条的修改,这样就需要我们有明确的思路,工程日志也能帮助我们走回头路.

当修改工程发生难以逆转的错误的时候,就需要我们导入原来的工程重新开始.因此每次做完一次成功的工程,将文件夹压缩并加入read me 是一个备份的好习惯.

导入以前成功工程的具体步骤 :新建工程,add source— create design source—create test.v file—将原来工程的src文件夹拷贝到新的文件夹覆盖—再在add source 中一次添加src 文件夹中的bd ,new,ip等设计文件,还有要添加xci约束文件.

导入IP的步骤 : Tools—project settings –IP—Repository manage—添加IP目录

柯乔

2017.9.25