**DDR日记**

文档简单，有用就好！

文档定位：用于DDR的设计修改等信息的记录！

记 录 员：袁亚鹏

开始日期：2017/1/1

结束日期：2017/12/31

DDR的整体框架见附件：[DDR簇设计](file:///E:\03%20%20数字设计\03%20%20项目相关\DDR_cluster.docx)

**2017/1/1**

**DDR簇设计中各个子模块的信息统计**

**<ddr\_cfg\_decoder>**

包含子模块：<cfg\_pos\_ok\_v2>

上游模块：<配置网>

下游模块：<ach\_top>

功 能：接收来自配置层网络的配置/请求信息，解析并产生下游模块需要的信息！

1. 配置：需要借助<cfg\_pos\_ok\_v2>计算配制簇的首次请求的二级矩阵的坐标！
2. 请求：需要识别出是指令/数据请求！

时序特征如下：

1. 时 钟：sysclk = 100M
2. 复 位：低电平、异步触发方式
3. 输入方式：部分来自寄存器，部分来自组合逻辑！
4. 输出方式：只有两个信号是逻辑输出！
5. *这两个信号可以修改为寄存器输出方式！*
6. *每一个输出信号的扇出很大，以总线的形式接至32个通用地址通道！*

**<cfg\_pos\_ok\_v2>**

内嵌模块：系模块<ddr\_cfg\_port>的子模块！

功 能：在配置方式下，协助父模块计算配制簇的首次请求的数据块的坐标！

时序特征：

1. 时 钟：sysclk = 100M
2. 复 位：低电平、异步触发方式
3. 输入方式：全部来自寄存器！
4. 输出方式：有两个信号直接接地，其他都为寄存器输出！

*【注】经综合发现，该模块的存在会极大地拉低系统频率，需要修改升级！*

**<ach\_top>**

内含众多子模块，架构如图：[ach\_top内部架构图](ach_top.docx)（地址通道）

**<addr\_ch\_MC>**

不包含子模块，但包括一个同步FIFO，连接FIFO的写入端，寄存器输出同步写入方式！

FIFO参数：

MC\_CMD\_FIFO参数

Clocking scheme: Common Clock

Memory type: Distributed RAM

Write\_width: 75 Read\_width: 75

Write\_depth: 16 Read\_depth: 16

Almost Full Flag: Not selected

Almost Empty Flag: Not selected

上游模块：<ddr\_cfg\_decoder>

下游模块：<Task\_arbiter\_v2>

功 能：在MC请求时产生请求需要的必要信息，打包放入MC\_CMD\_FIFO中！

时序特征如下：

1. 时 钟：sysclk = 100M
2. 复 位：低电平、异步触发方式
3. 输入方式：只有请求信号来自组合逻辑输出，其他都来自寄存器输出！
4. 输出方式：FIFO的读端口！

**<addr\_ch\_COP>**

不包含子模块，但包括一个同步FIFO，连接FIFO的写入端，寄存器输出同步写入方式！

FIFO参数：

COP\_CMD\_FIFO参数

Clocking scheme: Common Clock

Memory type: Distributed RAM

Write\_width: 45 Read\_width: 45

Write\_depth: 16 Read\_depth: 16

Almost Full Flag: Not selected

Almost Empty Flag: Not selected

上游模块：<ddr\_cfg\_decoder>

下游模块：<Task\_arbiter\_v2>

功 能：在MC请求时产生请求需要的必要信息，打包放入MC\_CMD\_FIFO中！

时序特征如下：

1. 时 钟：sysclk = 100M
2. 复 位：低电平、异步触发方式
3. 输入方式：只有请求信号来自组合逻辑输出，其他都来自寄存器输出！
4. 输出方式：FIFO的读端口！

**<general\_ach>**

包含众多子模块，有：

1. <addr\_ch\_xx>： 内含子模块<one\_cluster>

输出类型：寄存输出！

1. <MUX\_4\_IN>：4选1寄存（输出类型：寄存输出！）
2. <MUX\_2\_IN>：2选1寄存（输出类型：寄存输出！）
3. <MULTIPLE\_ROW>：例化的乘法器IP

MULTIPLE\_ROW参数

P (29:0)= A (Unsigned 30-bits)\* B (Unsigned 16-bits)

Multiplier Type: Parallel Multiplier

Pipeline Stages: 4

Use LUT: LUT6s 490个（不包括STLs）

Use Mults: DSP48 slices 2个

1. <MULTIPLE\_COL>：例化的乘法器IP

MULTIPLE\_COL参数

P (29:0)= A (Unsigned 30-bits)\* B (Unsigned 14-bits)

Multiplier Type: Parallel Multiplier

Pipeline Stages: 4

Use LUT: LUT6s 426个（不包括STLs）

Use Mults: DSP48 slices 2个

1. <GENERAL\_DATA\_FIFO>：例化的FIFO

COP\_CMD\_FIFO参数

Clocking scheme: Common Clock

Memory type: Block RAM

Write\_width: 234 Read\_width: 234

Write\_depth: 256 Read\_depth: 256

Almost Full Flag: Not selected

Almost Empty Flag: Not selected

**1/3/2017**

设计中的跨时钟域问题，主要体现在以下几个方面：

1. DDR\_IP初始

**重 要 性**：DDR簇中大多数子模块都需要该信号参与判断控制，系关键信号！

**处理手段**：对于单根跨时钟域信号，一般采用简单同步器（simple synchronizer），就是由至少两级CE端合Reset/Clear端接死的寄存器序列来处理；另外在XDC中施加ASYNC\_REG约束！

**现 状**：跨时钟域处理不到位，只采用了一级寄存同步，未施加XDC约束！

DDR\_wrapper中：增加简单同步器

reg phy\_init\_done\_r1;

reg phy\_init\_done\_r2;

reg phy\_init\_done\_syn;

always @ (posedge sys\_100M)

begin

phy\_init\_done\_r1 <= phy\_init\_done;

phy\_init\_done\_r2 <= phy\_init\_done\_r1;

phy\_init\_done\_syn <= phy\_init\_done\_r2;

end

XDC约束中：控制最大最大延时、将同步寄存器放置在同一SLICE中

set\_max\_delay -form [get\_cells –hier –filter {NAME =~ \*/u\_DDR\_wrapper/phy\_init\_done\_r2\_reg}] \

-to [get\_cells –hier –filter {NAME =~ \*/u\_DDR\_wrapper/phy\_init\_done\_syn\_reg}] \

-datapath\_only 8

set\_property ASYNC\_REG TRUE \

[get\_cells –hier –filter {NAME =~ \*/u\_DDR\_wrapper/phy\_init\_done\_r?\_reg}]

1. 由用户侧送至DDR\_IP的复位信号sys\_rst

由于两侧的时钟域并不相同，所以牵涉到跨时钟域问题！

经追踪发现，该信号受400M时钟的驱动，指向DDR\_IP内部一个pll复位端！

**现 状**：未做任何处理、未施加任何约束！

**2017/1/4**

**建议一：去除COP指令专用地址通道中的二级坐标，只用一级坐标参与判断！**

关于COP指令专用地址通道，预设二级坐标的原意是考虑到同一个COP连续请求两次或者两次以上的取指令请求的情况，这时需要根据二级坐标来区分不同的请求！但如今看来，这种考虑完全是没有必要的，因为指令请求时会附带请求的偏移量，而所有的COP指令请求共用同一基址。虽然COP的请求包含指令请求和数据请求，但这两种请求会被解析模块解析出来，给出一个指令请求或者数据请求的标志，所以，在COP指令专用地址通道看来，完全可以准确的正常运作！

**建议二：解决phy\_init\_done的跨时钟域问题！**

**建议三：更换DDR\_IP、建议DDR\_IP\_PHY侧信号名在模块传递中保持不变，更加规范化！**

原有的DDR\_IP，是探索摸索中使用的，在例化DDR\_IP核时，内部的选项可以做修改，使得生成的IP 文件不需要做什么修改就可以使用！

**建议四：对设计中的文件建立更好的管理机制！**

XDC文件可以按照约束对象的不同来将设计中的约束文件分放到不同的约束文件下，便于同一管理！

设计中的XDC约束按照设计中是否为IP核可以分为：

1. 用户设计类XDC约束

这类XDC约束需要用户自己设定约束文件！

1. IP类XDC约束

XDC约束文件在IP例化完成时自动生成，无需用户设定！

设计中的XDC约束按照约束对象的不同可以分为：

1. 时序约束
2. 基本时序约束

主时钟、生成时钟、虚拟时钟

输入输出延时

1. 高级时序约束

最大最小延时、多周期路径、虚假路径

1. 物理约束

物理约束，又名管脚绑定，就是将设计顶层的端口与FPGA开发板的管脚互联！

【注】设计中遇到跨时钟域问题需要运用上述时序约束手段来对跨时钟域进行约束！