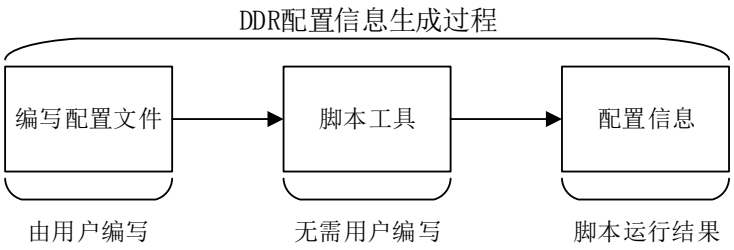
希望本文档将起到以下作用：
A）使你更清晰地理解 DDR 中配置信息的含义
B）使你学会如何配置 DDR 一侧的配置信息
C）使你掌握 DDR 配置信息脚本生成工具

阅读本文档之前，希望你已经阅读过《DDR_v1.1 使用说明书》！

【览前声明】
在此之前，你需要对以下概念有清晰的认识！
1）DDR_v1.1 中包括哪几类“数据”？（数据的类型）
2）数据是如何组织的？（数据的组织形式）
3）一级矩阵、二级矩阵是什么？（数据的组织形式）
4）DDR_v1.1 中的地址通道是什么？有几类？有几个？（地址通道的概念）
5）DDR_v1.1 中的数据通道是什么？有几类？有几个？（数据通道的概念）
6）配置信息中，一级坐标和二级坐标是什么？
7）配置信息中，如何理解配置簇的套数？
8）配置信息中的三方节点和目的节点有何区别？

【快速预览】
1、配置信息生成流程
2、本文档包含哪些实例？

【配置信息生成流程】



配置文件：ddr_cfg_info.vh
脚本工具：ddr_cfg_info_generator.pl
脚本语言：perl
使用方法：参考《DDR_v1.1_配置信息生成工具使用说明.txt》
快速链接：配置信息脚本生成工具目录

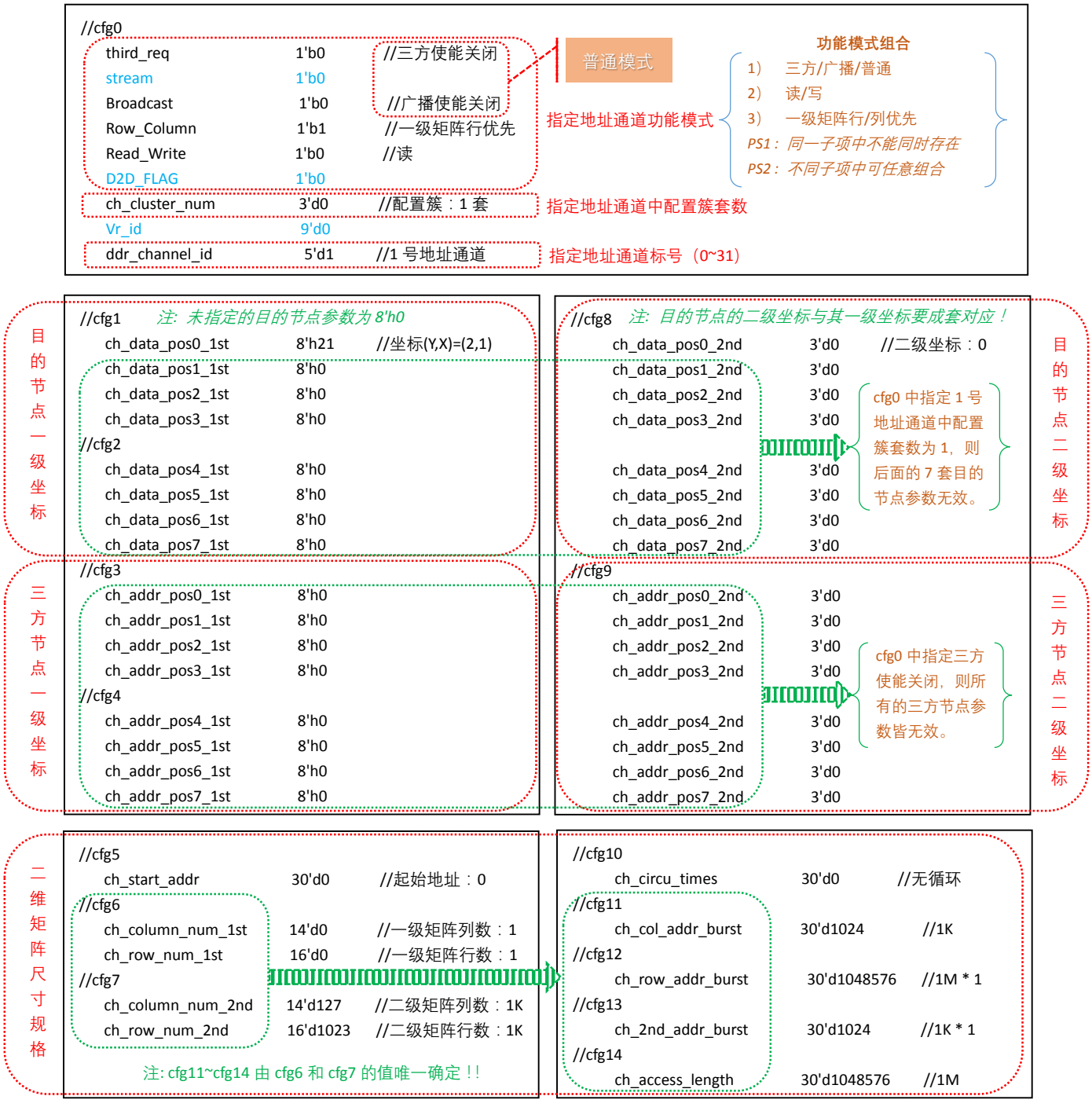
【实例展示】

实例1:　一个普通的读操作

任务：簇 A 从 DDR 中读取一批源数据，数据批量 1M（默认 64-bits），只读取一次。

已知：簇 A 的一级坐标为 8'h21

配置：



注 1：cfg15 保留不用，默认全零，由脚本工具自动添加，用户无需关心，后续将不再出现该提示！

注 2：cfg0 中的蓝色标记的配置项在当前系统中不使用，默认为零，后续将不再出现该提示！

注 3：或许你存在下面的疑惑：

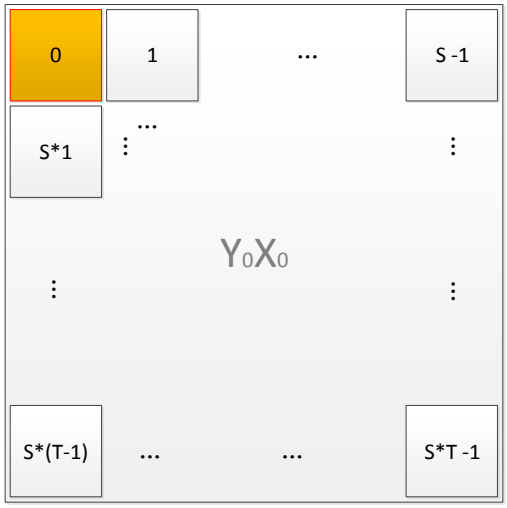
"既然 cfg11~cfg14 由 cfg6~cfg7 唯一确定，为什么还要配置 cfg11~cfg14 呢，岂不是多此一举？"

这里给出答案：从反面考虑，如果我们不做"计算出 cfg11~cfg14"这件事，那么就需要硬件去做这件事，然而，在硬件中实现一个乘法器，特别是大位宽的乘法器，是需要付出很大代价的。从正面考虑，如果我们"代替"硬件完成该"计算"，则一方面可以减轻硬件的压力，另一方面也可以使设计者"校验"cfg6~cfg7 参数是否正确！

实例 1 中，DDR 的重要参数（真实值）提取如下：

1 号地址通道、普通读模式、行优先[1]、无循环、1 套配置簇、一级矩阵 1 行 1 列、二级矩阵 1K 行 1K 列、一级矩阵起始地址为 0。

数据组织示意图如下：



左图参数

一级矩阵 1 行 1 列 ===>> 一级矩阵总块数：1

二级矩阵 T 行 S 列 ===>> 二级矩阵大小：S*T

令(S, T) = (1024, 1024)，可得到实例 1 中的参数！

二级矩阵行加一跳变距离：(S*1) *1

一级矩阵行加一跳变距离：(S*1) *T

一级矩阵列加一跳变距离：S

行间地址连续

若一级矩阵只有 1 列，二级矩阵相邻行行间地址连续！

因此，当一级矩阵列数为 1 时，二级矩阵的行列数可以任意变换，但要遵循两个原则：

1） 二级矩阵行、列数的乘积不变

2） 二级矩阵列数必须是 8 的整数倍

实例延伸（实例 1）

延伸1-1. 读→写

如果将 cfg0 中的“Read_Write”设置为 1，则簇 A 将不执行读操作，而是写操作，写入的位置由二维矩阵参数指定，数据批量不变！

延伸1-2. 不改变读取次数，增大改数据量

由于每次访问的是一个一级矩阵块——即一个二级矩阵的大小，所以，只需调整二级矩阵的行、列参数值，使二者的乘积增大或减小即可！

延伸1-3. 在单次数据批量不变的情况下，增加读取次数

在非三方模式（广播、普通）下，所有的一级矩阵块都是需要被访问的数据对象，且必须被某个配置簇所请求。

如果循环不开启，则一级矩阵总块数由下述公式决定：

一级矩阵总块数 = 一级矩阵列数 * 一级矩阵行数 ……公式一

如果循环开启，比如循环 L 次，则：

一级矩阵总块数 = 一级矩阵列数 * 一级矩阵行数 * (1+L) ……公式二

由于无循环时，L=0，则一级矩阵总块数都可以用公式二统一表示。

本实例中循环不开启，增加读取次数，只需增大一级矩阵行、列数。

但请注意以下两点：

要点一：在二级矩阵的行、列数保持不变的前提下，若改变一级矩阵列数，则二级矩阵行加一地址跳变距离和一级矩阵行加一地址跳变距离也会变化！

要点二：在非三方模式下，当访问一级矩阵块时，访问顺序由行/列优先决定，同时又受到当前地址通道中配置簇套数（共 8 套，分别对应一级、二级坐标的唯一组合）的影响。

[1] 模式中的“行/列优先”这一概念的对象只有一级矩阵，所以后续提及的“行（列）优先”都直接省略了“一级矩阵”字眼！

假设只有一个配置簇，一级矩阵 3 行 4 列，行优先、列优先的访问次序差异如下图。其中，绿色 YMXN 表示一级矩阵块在一级矩阵中的坐标为(M, N)。第一行对应行坐标为 0，第一列对应列坐标为 0。红色数字代表一级矩阵块被访问的顺序。

Y0X0 1	Y0X1 2	Y0X2 3	Y0X3 4
Y1X0 5	Y1X1 6	Y1X2 7	Y1X3 8
Y2X0 9	Y2X1 10	Y2X2 11	Y2X3 12

行优先

Y0X0 1	Y0X1 4	Y0X2 7	Y0X3 10
Y1X0 2	Y1X1 5	Y1X2 8	Y1X3 11
Y2X0 3	Y2X1 6	Y2X2 9	Y2X3 12

列优先

图示 ： 行、列优先级下的一级矩阵块的访问次序对比（1套配置簇）

假设存在多个配置簇，一级矩阵 3 行 4 列，下图形象地刻画出 3 个簇时行优先、列优先的访问次序差异。其中，绿色 YMXN 表示一级矩阵块在一级矩阵中的坐标为(M, N)。第一行对应行坐标为 0，第一列对应列坐标为 0。红色字母 A+数字代表该一级矩阵块被簇 A 访问的次序，An 只能被簇 A 访问不能被其他簇访问。An 的访问、Bm 的访问、Ct 的访问，三者之间没有任何联系。簇 A 的 4 次请求将依次访问 A1/A2/A3/A4 这四个块。

Y0X0 A1	Y0X1 B1	Y0X2 C1	Y0X3 A2
Y1X0 B2	Y1X1 C2	Y1X2 A3	Y1X3 B3
Y2X0 C3	Y2X1 A4	Y2X2 B4	Y2X3 C4

行优先

Y0X0 A1	Y0X1 A2	Y0X2 A3	Y0X3 A4
Y1X0 B1	Y1X1 B2	Y1X2 B3	Y1X3 B4
Y2X0 C1	Y2X1 C2	Y2X2 C3	Y2X3 C4

列优先

图示 ： 行、列优先级下的一级矩阵块的访问次序对比（3套配置簇）

延伸1-4. 实例 1 和延伸 1-1 都是用一个簇完成读或写操作，那么可以用一个簇来完成读写操作吗？

答案当然是可以的，但是经过分析发现存在几个小问题（附带解决手段）：

Q1. 一个地址通道只能读或者只能写。

用两个地址通道，一个用于读，一个用于写，只需要多配置一段配置流，此时要注意两个地址通道要用不同的地址通道标号来区分！

Q2. 在网络中，同一个运算簇 A 的位置（一级坐标）是唯一的，那么 DDR 侧如何知道运算簇的当前请求是读请求，还是写请求呢？

只需要判断出当前请求属于哪个地址通道，并查询对应地址通道的读写模式即可知道是否是读请求。对于 DDR，其两个地址通道的一级坐标都指向簇 A 在网络中的坐标，单靠一级坐标已经无法区分当前请求属于哪个地址通道，此时“二级坐标”的作用被凸显出来，我们可以通过在不同的地址通道中配置不同的二级坐标值，辅助一级坐标来判断请求与地址通道的所属关系。

实例2:　多个簇并存的读操作

任务：簇 A、簇 B、簇 C 分别从 DDR 中读取源数据。

已知：簇 A、簇 B、簇 C 的一级坐标依次为 8'h21、8'h33、8'h45

配置：

```
//cfg0
    third_req              1'b0        //三方使能关闭
    stream                 1'b0
    Broadcast              1'b0        //广播使能关闭
    Row_Column             1'b1        //一级矩阵行优先
    Read_Write             1'b0        //读
    D2D_FLAG               1'b0
    ch_cluster_num         3'd2        //配置簇：3 套
    Vr_id                  9'd0
    ddr_channel_id         5'd0        //0 号地址通道
```

要点说明

普通读模式、行优先、无循环
0 号地址通道、3 套配置簇
入口地址 1M、
一级矩阵 3 行 4 列、二级矩阵 10 行 1K 列

目的节点一级坐标

```
//cfg1
    ch_data_pos0_1st       8'h21       //坐标(Y,X)=(2,1)
    ch_data_pos1_1st       8'h33       //坐标(Y,X)=(3,3)
    ch_data_pos2_1st       8'h45       //坐标(Y,X)=(4,5)
    ch_data_pos3_1st       8'h0
//cfg2
    ch_data_pos4_1st       8'h0
    ch_data_pos5_1st       8'h0
    ch_data_pos6_1st       8'h0
    ch_data_pos7_1st       8'h0
```

有用信息

目的节点二级坐标

```
//cfg8
    ch_data_pos0_2nd       3'd0        //二级坐标：0
    ch_data_pos1_2nd       3'd0        //二级坐标：0
    ch_data_pos2_2nd       3'd0        //二级坐标：0
    ch_data_pos3_2nd       3'd0

    ch_data_pos4_2nd       3'd0
    ch_data_pos5_2nd       3'd0
    ch_data_pos6_2nd       3'd0
    ch_data_pos7_2nd       3'd0
```

```
//cfg3
//cfg4
```

三方节点一级坐标

```
//cfg9
```

三方节点二级坐标

由于三方使能关闭，因此所有与三方节点相关的设置都无效，为节约篇幅，不显示！

二维矩阵尺寸规格

```
//cfg5
    ch_start_addr          30'd1048576 //起始地址：1M
//cfg6
    ch_column_num_1st      14'd3       //一级矩阵列数：4
    ch_row_num_1st         16'd2       //一级矩阵行数：3
```

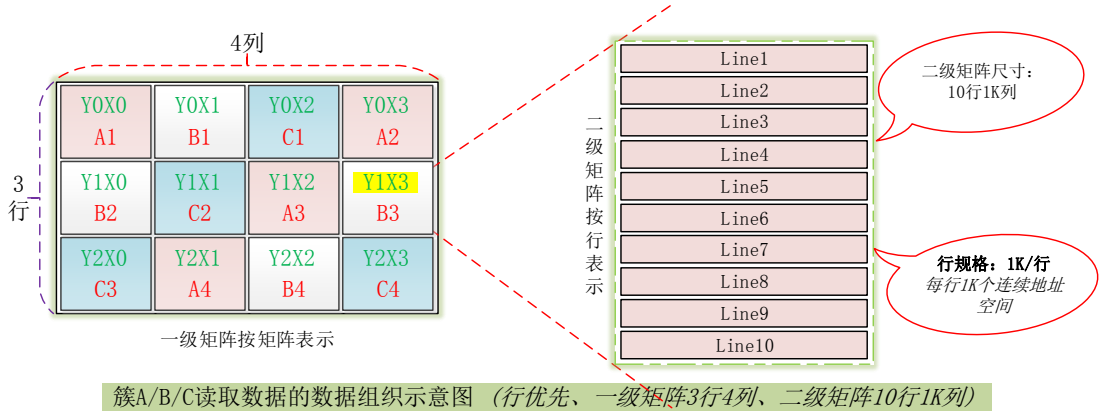
一级矩阵总块数 = 4 * 3 = 12

```
//cfg7
    ch_column_num_2nd      14'd127     //二级矩阵列数：1K
    ch_row_num_2nd         16'd9       //二级矩阵行数：10
```

一级矩阵大小：1K * 10 = 10K

```
//cfg10
    ch_circu_times         30'd0       //无循环
//cfg11
    ch_col_addr_burst      30'd1024    //1K
//cfg12
    ch_row_addr_burst      30'd40960   //10K * 4
//cfg13
    ch_2nd_addr_burst      30'd4096    //1K * 4
//cfg14
    ch_access_length       30'd10240   //10K
```

数据组织结构图如下图所示（*数据块的所属关系由红色字母标识，同延伸 1-3*）：



簇A/B/C读取数据的数据组织示意图 *（行优先、一级矩阵3行4列、二级矩阵10行1K列）*

实例延伸（实例 2）

延伸2-1. 实例 1、实例延伸 1-1~1-4、实例 2 中都是用的普通模式（MODE），何时用到三方、广播模式呢？

选择哪一种模式是根据任务的目的来决定的，也可从数据访问特征方面进行考虑。

普通模式：目的节点访问哪一个数据块是顺序固定的，每一个数据块只属于唯一的一个配置簇，并仅由该目的节点决定；数据传递网络为 PCC 网络；一个数据块被请求响应时只会与一个簇进行数据交互；支持循环读/写。

广播模式：目的节点访问哪一个数据块是顺序固定的，但每一个数据块都被所有的配置簇共享，当且仅当所有配置簇都发出访存请求时才会开始进行数据通讯；数据传递网络为广播网络；一个数据块被请求响应时会同时与所有配置簇进行数据通讯；支持循环、不支持写模式。

三方模式：当目的节点访问 DDR 时，目的节点并不知道数据来自哪个数据块（读操作时）或者将数据写入哪个数据块（写操作时），它仅仅知道自身需要从 DDR 获取一批数据（读操作）或者将一批数据写入 DDR（写操作），数据块在一级矩阵中的坐标需要三方节点来指定；要想访问一个数据块，必须满足两个条件：a) 目的节点 A 的访存请求到来 && b) 三方节点 PA（对应目的节点 A）的访存请求到来；数据块只会与目的节点进行数据通讯，而不会与三方节点进行数据通讯；支持读/写模式，不支持循环（此时循环无效）。

选择具体模式的方法

1) 如果想把一批数据同时传递给多个运算簇，则可以选择广播模式；
2) 如果运算簇不知道自身需要取哪些数据，或者不知道将结果数据写入 DDR 中的什么位置，则可以选择三方模式；
3) 否则选取普通模式；

功能模式	三方节点		目的节点		三方节点和目的节点是否成对
	使用	数量	使用	数量	
三方	√	1	√	1	√
广播	×	——	√	全部	——
普通	×	——	√	1	——

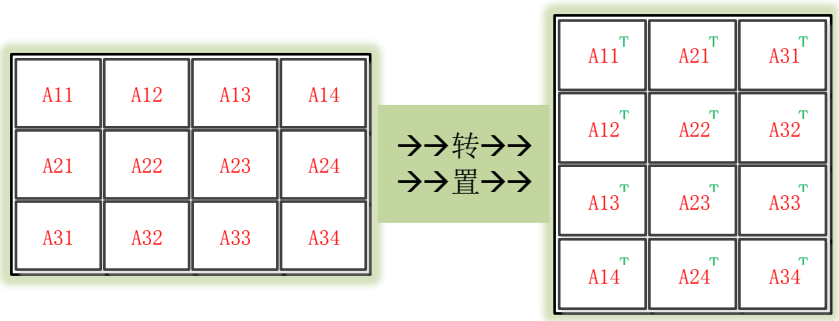
实例3:　矩阵转置

本实例主要用于使大家对 DDR 中的数据组织形式有更为清晰的认识。

目标：完成一个大规模矩阵（8M 实数矩阵，1K*8K，单位 64-bits）的转置任务。
资源：片上 RAM 资源（distributed-RAM、BlockRAM）总量不能超过 2*36Kb。

资源分析： 2*36Kb RAM 资源量支持如下规格的 RAM 配置：
64K*1-bits、32K*2-bits、16K*4-bits、8K*9-bits、4K*18-bits、2K*36-bits、1K*72-bits
由于数据位宽为 64 位，则最大可使用的 RAM 资源规格为 1K*72-bits，地址深度最大 1K。

对于一个大矩阵的转置，利用矩阵相关知识，将大矩阵拆分成小矩阵，对小矩阵进行转置，是可行且实际的方案。



那么，关键点在于矩阵拆分，拆分后的小矩阵（矩阵单元）的规格多大才好呢？下面两点将会对你的决定起到帮助：

1） 受 DDR 中突发长度的局限，每次访存 DDR 的数据必须为 8 的整数倍，这决定了矩阵单元的最小规格为 8*8。
2） 系统为粗粒度运算，一次尽可能搬运较多的数据。

矩阵单元可取的规格：8*128、16*64、32*32、64*16、128*8。

不如选取 16*64，则原矩阵 A（1K*8K）拆分后组成一个二维矩阵，其参数如下：
一级矩阵：64 行 128 列
二级矩阵：16 行 64 列
对应地，转置矩阵 A^T 也组成一个二维矩阵，参数如下：
一级矩阵：128 行 64 列
二级矩阵：64 行 16 列

通道分配方面：开辟两个地址通道，分别用于获取原矩阵 A、将转置矩阵 A^T 写入 DDR。

对原矩阵 A 的 64*128 个矩阵单元分别做转置，可以只使用一个运算簇单独完成。由于各个矩阵单元的转置互不相关，因此也可以使用多个运算簇共同完成，但请注意以下两点：

1） BRAM 资源将增加，增加倍数等于总的转置簇的个数！
2） 一级矩阵总块数（64*128）必须是簇个数的整数倍！

这里不妨使用 8 个簇（8'h11/8'h22/8'h33/8'h44/8'h55/8'h66/8'h77/8'h88）来共同完成。
问：这种情况下，数据块属于哪个配置簇呢？（*请大家自行脑补*）

下面将只给出原始配置信息文件的结果！

假设 1：原矩阵的数据已经写入 DDR 的一段连续地址空间中，且起始地址为 0。

首先：为原矩阵 A 开辟一个地址通道，普通读模式，行优先，用于获取源数据。

```
//cfg0
    third_req            1'b0      //三方使能关闭
    stream               1'b0
    Broadcast            1'b0      //广播使能关闭
    Row_Column           1'b1      //一级矩阵行优先
    Read_Write           1'b0      //读
    D2D_FLAG             1'b0
    ch_cluster_num       3'd7      //配置簇：8 套
    Vr_id                9'd0
    ddr_channel_id       5'd0      //0 号地址通道
```

要点说明

普通读模式、行优先、无循环
0 号地址通道、8 套配置簇
入口地址 0
一级矩阵 64 行 128 列
二级矩阵 16 行 64 列

目的节点一级坐标

```
//cfg1
    ch_data_pos0_1st     8'h11     //坐标(Y,X)=(1,1)
    ch_data_pos1_1st     8'h22     //坐标(Y,X)=(2,2)
    ch_data_pos2_1st     8'h33     //坐标(Y,X)=(3,3)
    ch_data_pos3_1st     8'h44     //坐标(Y,X)=(4,4)
//cfg2
    ch_data_pos4_1st     8'h55     //坐标(Y,X)=(5,5)
    ch_data_pos5_1st     8'h66     //坐标(Y,X)=(6,6)
    ch_data_pos6_1st     8'h77     //坐标(Y,X)=(7,7)
    ch_data_pos7_1st     8'h88     //坐标(Y,X)=(8,8)
//cfg3
//cfg4
```

三方节点一级坐标

目的节点二级坐标

```
//cfg8
    ch_data_pos0_2nd     3'd0      //二级坐标：0
    ch_data_pos1_2nd     3'd0      //二级坐标：0
    ch_data_pos2_2nd     3'd0      //二级坐标：0
    ch_data_pos3_2nd     3'd0      //二级坐标：0
    ch_data_pos4_2nd     3'd0      //二级坐标：0
    ch_data_pos5_2nd     3'd0      //二级坐标：0
    ch_data_pos6_2nd     3'd0      //二级坐标：0
    ch_data_pos7_2nd     3'd0      //二级坐标：0
//cfg9
```

三方节点二级坐标

由于三方使能关闭，因此所有与三方节点相关的设置都无效，为节约篇幅，不显示！

二维矩阵尺寸规格

```
//cfg5
    ch_start_addr        30'd0     //起始地址：0
//cfg6
    ch_column_num_1st    14'd127   //一级矩阵列数：128
    ch_row_num_1st       16'd63    //一级矩阵行数：64
```

一级矩阵总块数 = 128* 64 = 8K

```
//cfg7
    ch_column_num_2nd    14'd7     //二级矩阵列数：64
    ch_row_num_2nd       16'd15    //二级矩阵行数：16
```

一级矩阵块大小：64 * 16 = 1K （即二级矩阵大小）

```
//cfg10
    ch_circu_times       30'd0     //无循环
//cfg11
    ch_col_addr_burst    30'd64    //64
//cfg12
    ch_row_addr_burst    30'd131072  //1K * 128
//cfg13
    ch_2nd_addr_burst    30'd8192  //64 * 128
//cfg14
    ch_access_length     30'd1024  //1K
```

下面是原矩阵和转置矩阵的通道参数对比：

对于原矩阵 A

普通读模式、行优先、无循环
0 号地址通道、
8 套配置簇
簇的二级坐标都为 0
入口地址 0
一级矩阵 64 行 128 列
二级矩阵 16 行 64 列

对于转置矩阵 A^T

普通写模式、列优先、无循环
3 号地址通道、
8 套配置簇
簇的二级坐标都为 1
入口地址 10M
一级矩阵 128 行 64 列
二级矩阵 64 行 16 列

两点明显差别：1）两个地址通道的行/列优先级恰好相反；

2）两个地址通道中的配置簇的一级坐标对应相同，二级坐标不同。

假设 2：希望将转置矩阵的数据放入 DDR 的一段连续地址空间中，且起始地址为 10M。

为转置矩阵 A^T 开辟一个地址通道，普通写模式，列优先，用于将结果数据写入 DDR。

```
//cfg0
    third_req              1'b0        //三方使能关闭
    stream                 1'b0
    Broadcast              1'b0        //广播使能关闭
    Row_Column             1'b0        //一级矩阵列优先
    Read_Write             1'b1        //写
    D2D_FLAG               1'b0
    ch_cluster_num         3'd7        //配置簇：8 套
    Vr_id                  9'd0
    ddr_channel_id         5'd3        //3 号地址通道
```

要点说明

普通写模式、列优先、无循环
3 号地址通道、8 套配置簇
入口地址 10M
一级矩阵 128 行 64 列
二级矩阵 64 行 16 列

目的节点一级坐标

```
//cfg1
    ch_data_pos0_1st       8'h11       //坐标(Y,X)=(1,1)
    ch_data_pos1_1st       8'h22       //坐标(Y,X)=(2,2)
    ch_data_pos2_1st       8'h33       //坐标(Y,X)=(3,3)
    ch_data_pos3_1st       8'h44       //坐标(Y,X)=(4,4)
//cfg2
    ch_data_pos4_1st       8'h55       //坐标(Y,X)=(5,5)
    ch_data_pos5_1st       8'h66       //坐标(Y,X)=(6,6)
    ch_data_pos6_1st       8'h77       //坐标(Y,X)=(7,7)
    ch_data_pos7_1st       8'h88       //坐标(Y,X)=(8,8)
```

目的节点二级坐标

```
//cfg8
    ch_data_pos0_2nd       3'd1        //二级坐标：1
    ch_data_pos1_2nd       3'd1        //二级坐标：1
    ch_data_pos2_2nd       3'd1        //二级坐标：1
    ch_data_pos3_2nd       3'd1        //二级坐标：1
    ch_data_pos4_2nd       3'd1        //二级坐标：1
    ch_data_pos5_2nd       3'd1        //二级坐标：1
    ch_data_pos6_2nd       3'd1        //二级坐标：1
    ch_data_pos7_2nd       3'd1        //二级坐标：1
```

```
//cfg3
//cfg4
```

三方节点一级坐标

```
//cfg9
```

三方节点二级坐标

由于三方使能关闭，因此所有与三方节点相关的设置都无效，为节约篇幅，不显示！

二维矩阵尺寸规格

```
//cfg5
    ch_start_addr          30'd10485760        //起始地址：10M
//cfg6
    ch_column_num_1st      14'd63      //一级矩阵列数：64
    ch_row_num_1st         16'd127     //一级矩阵行数：128
```

一级矩阵总块数 = 64* 128 = 8K

```
//cfg7
    ch_column_num_2nd      14'd1       //二级矩阵列数：16
    ch_row_num_2nd         16'd63      //二级矩阵行数：64
```

一级矩阵块大小：16 * 64 = 1K （即二级矩阵大小）

```
//cfg10
    ch_circu_times         30'd0       //无循环
//cfg11
    ch_col_addr_burst      30'd64      //64
//cfg12
    ch_row_addr_burst      30'd65536   //1K * 64
//cfg13
    ch_2nd_addr_burst      30'd1024    //16 * 64
//cfg14
    ch_access_length       30'd1024    //1K
```

***PS：**以上三个实例的配置全部清除之后，其他模式的配置也轻而易举，所以不再给出其他类似的实例，如发现错误，请及时通知我；如有不懂之处，可与我讨论。*

--END--

ANSWER

数据类型

DDR 中的数据包括以下三类：

1） 配置信息类

对各个簇进行配置所需的信息（只读），它其实是 MC 指令信息的一部分！

2） 指令信息类

系统中有两种独特的数据（对于 DDR 只读）：

A)MC 指令信息

B) COP 运算类指令信息

3） 运算类数据

在任务中参与运算的数据，按照因果关系分为三种：

A)源数据

B) 中间数据

C)结果数据

这三种数据对 DDR 来说是没有区别的，但仍有必要清晰地建立这种概念！

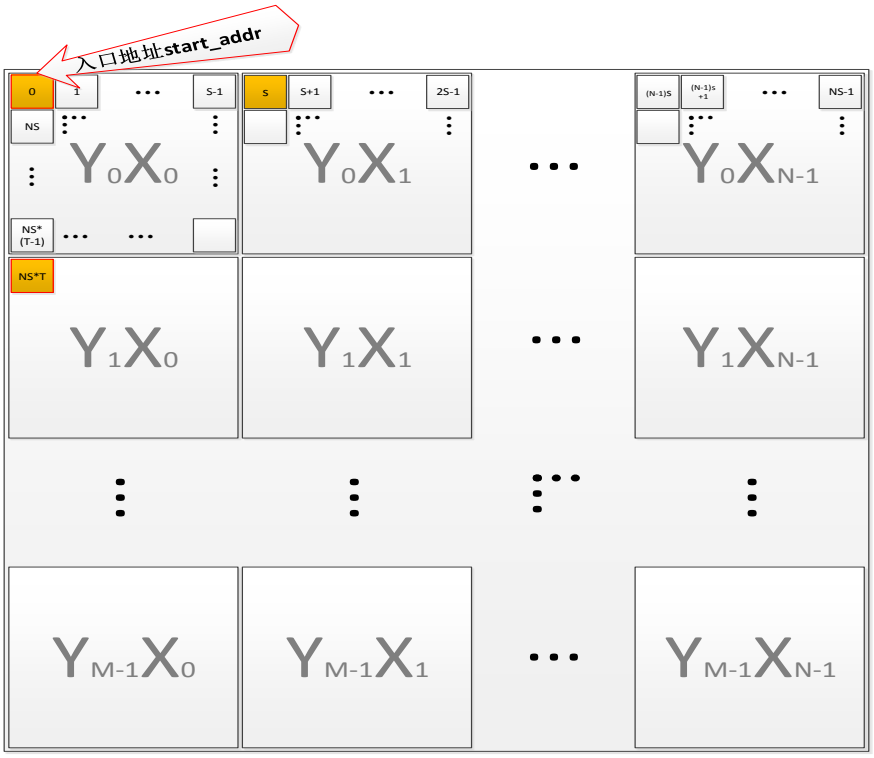
【数据的组织形式】

数据采用“二维矩阵”的形式来组织与管理数据，这主要是为了“迎合”现有系统中粗粒度的需求。每次访存 DDR 的对象是一批数据，在下图中可以看到数据对象是一个一级矩阵块。

误区：数据在内存中也是以“二维矩阵”形式存放！

解析：“二维矩阵”只是一种便于管理数据的方式（手段），从根本上来讲，它是对内存中存储单元对应地址的管理，是一种地址变换手段，要想访问内存空间中的某一单元中的数据，必须在“二维矩阵表示”与“真实访存地址”之间进行地址转换。

下面给出地址转换的规则：



一级矩阵：M行N列
二级矩阵：T行S列
二级矩阵块个数： TS
一级矩阵块$Y_{i-1}X_{j-1}$：一级矩阵表格的第i行j列, 其入口地址为：
(一级矩阵入口地址) + (i-1)*(一级矩阵行加一地址跳变距离) + (j-1)*(一级矩阵列加一地址跳变距离)

注意图中的三个橙色小块，为了说明的方便，三个小块一次命名为P1/P2/P3,相对于一级矩阵入口地址的偏移量依次为0、S、NS*T，那么有如下定义：
一级矩阵块入口地址：P1块在内存空间中的真实地址,图中红色箭头指向的真实地址！
一级矩阵列加一地址跳变距离：P1到P2的距离，即：S-0 = S
一级矩阵行加一地址跳变距离：P1到P3的距离，即：NS*T-0 = NS*T
二级矩阵行加一地址跳变距离：P1到P1块下面的那个小块的距离，即NS-0 = NS

【地址通道的概念】
系统中任务的执行过程是提前调度安排好的，这决定了数据在内存中的存放位置是安排好的，哪一个计算簇需要哪些数据，都是可以确定的。比如 RCU 需要取一批数据参与运算，那么，DDR 一侧至少要知道如下几项必要信息：

1） 这批数据的位置如何唯一确定？
2） 这批数据数据是谁的？
3） 这批数据什么时候被需要？

上述三项信息实则对应三个问题

1） 位置问题
2） 数据所属问题
3） 时机选择

针对问题 1）：

按照“二维矩阵”的形式来组织数据，设置好如下参数：
参数1: 二维矩阵的尺寸规格参数
参数2: 二维矩阵的存取模式（行、列优先）
参数3: 数据在二维矩阵中的位置

针对问题 2）：

设置好如下参数：
参数1: 用于唯一识别运算簇的坐标（一级坐标、二级坐标）

针对问题 3）：

这需要建立在前两个问题的基础上，为简便起见，不考虑！

问：那么我们如何给出上述三项信息呢？
答：在硬件中需要给出一个统一的约定，这种约定通过配置信息来实现。配置信息明确指出问题 1)、问题 2）中需要指定的参数，并配合其他约定（配置层网络协议、REQ 格式）（问题 3）的解决方案），就可以确定一个信息：在什么时间、访问什么地方的数据、这些数据是谁的！那么，问题来了，谁来做这件事情呢？
解决方案：配置解析模块 + 地址通道模块。前者用于解析来自配置网的信息，后者负责做这件事情！

其实，地址通道负责“识别数据的位置、查询数据属于谁、指定一个时机用于发起数据交互”，相当于一个“数据管理员”，控制数据、管理数据。因为一切访存数据的行为必须先被地址通道“允许”后才能获得“完成动作”的资格，因此，地址通道又被称为“控制通路”。

表 1　DDR_v1.1 中的地址通道

类型	功能	数量	说明
MC_ach	主控制器取指令专用地址通道	1	
COP_ach	COP 取指令专用地址通道	1	
General_ach	访存数据通用地址通道	32	可以酌情减半至 16 个

考虑地址通道中的优先级问题，通用地址通道的优先级等同，三种类型地址通道的优先级则按序号依次降低：

1）MC（Main Controller）取指令专用地址通道-------最高优先级
2）COP 取指令专用地址通道
3）取数据通用地址通道----------最低优先级

考虑地址通道内的配置簇个数问题，下面是三者的对比情况：

1）MC_ach 中只包含一个 MC 簇坐标（一级坐标），系统中只有一个 MC。
2）COP_ach 中包含 8 个 COP 簇坐标（一级坐标），坐标个数容易增加
3）单个通用地址通道中包含 8 套配置簇坐标（一级坐标 + 二级坐标）。

【数据通道的概念】

“数据通路”与“地址通道”相呼应，它响应地址通道产生的结果，根据该结果执行相应动作，代表着：在访存 DDR 执行“数据交互”时，数据在 DDR 簇结构中所走过的“路径”。

鉴于带宽压力，DDR_v1.1 中布置 6 路数据通道，就可以将 DDR_SDRAM 的带宽完全挖掘出来（再次提升数据通道个数，DDR_SDRAM 支持不了那么高的带宽）。

假设数据通道编号 1~6，则默认优先级依次降低：（最高）1→2→3→4→5→6（最低）。在响应一个请求时，必须存在空闲的数据通道，在空闲数据通道列表中，选择最高优先级通道，并将其分配给当前请求。

--END--