ban环境学习：

1. gf 表示进入makefile中的路径，:br表示返回。
2. dfhl用来看机器空间的大小。
3. 怎么get环境，查看~/.cshrc

alias gchx001 "/cpuwrk/chx001/sim/get\_chx001 ."

alias gchx002 "/cpuwrk/chx002/sim/get\_chx002"

alias gchx002nbsb "/cpuwrk/chx002/sim/get\_chx002\_nbsb"

alias gcha "/cpuwrk/cha/sim/env\_nb/get\_cha001"

alias gchanbsb "/cpuwrk/cha/sim/get\_cha\_nbsb"

Chip(NB grd) 环境：

/cpuwrk/chx002/sim/chx002\_grd grd1 peg2 GOLD

module level的环境:

DRAMC: /cpuwrk/chx002/users/franco/get\_dramc\_env

GMINT： /logic/apolloz/bin/get\_gmint

1. linux terminal中，ctrl+A光标移到开头，ctrl+E光标移到结尾。
2. NB的pad checklist主要有dram，pcie以及pll。

pad\_chklist = $(dram\_pad\_chklist) $(pcie\_pad\_chklist) $(nb\_pad\_chklist)

1. SB的pad checklist主要有pad\_monitor\_check\_chklist和func\_mux\_check\_chklist。生成的checklist需要根据实际情况对应在两者下面。
2. IRS：internal register set 各个device内容寄存器的配置说明书。
3. hg st:用来查看文件是否改动； hg revert aa用来恢复改动的文件aa；hglg查看版本，使用q退出；

过程如下：首先hg pull将central的高版本拉到local中，再利用hg up和local的低版本（与central对应的低版本相比修改过）进行merge，然后hg ci –m“eric test”用来commit提交产生最高版本, hg rollback可用来撤销；最后hg push将loacl的最高版本放到central中。

正常来讲，merge失败后hg工具会禁止commit，需人工merge并hg resolve –mark后才行。

版本控制常用命令：cp instance.v instance.v.版本号；hg up –r +版本号； g –d instance.v instance.v.版本号。

1. 更新环境：sim/upenv
2. NB is Device 0

• Registers of P6IF, DRAMC, AGPC and PCI1

SB is Device 17

• Registers of ISA, IDE, USB, PMU, AC97

11、sed命令的一般格式为：sed [选项] “脚本” 文件

（下文中“命令”都是指sed的命令部分。）

选项：

sed的选项形如-n, -e, -f等，都有个'-'。

-I 选项可以修改原文件

-n    忽略默认输出（默认输出就是整个文件的内容）。

-e    执行多个编辑任务

-f     执行文件中的命令集合

|  |  |
| --- | --- |
| 元字符 | 功能 |
| ^ | 行首定位符 |
| $ | 行尾定位符 |
| . | 匹配除换行符以外的单个字符 |
| \* | 匹配零个或多个前导字符 |
| \? | 匹配零个或1个前导字符 |
| \+ | 匹配1个或多个前导字符 |
| \| | 或操作 |
| \s | 匹配单个空白字符（如\t） |
| \S | 匹配单个非空白字符 |
| \w | 匹配单个单词 |
| \W | 匹配单个非单词 |
| [] | 匹配指定字符组内的任一字符 |
| [^] | 匹配不在指定字符组内的任一字符 |
| [-] | 匹配字符集的字符范围 |
| (..) | 保存已匹配的字符（括号前面要加反斜杠\，csdn没显示出来） |
| & | 保存查找串以便在替换串中引用 |
| \< | 词首定位符 |
| \> | 词尾定位符 |
| x\{m\} | 连续m个x |
| x\{m,\} | 至少连续m个x |
| x\{m,n\} | 至少连续m个，但不超过n个x |
| (str) | 正则表达式中对字符串的引用（括号前面要加反斜杠\，csdn没显示出来） |
| 中括号表达式 | 如[:alnum:]，[:alpha:]，[:digit:]，[:lower:]，  [:upper:]，[:punct:]，[:space:] |

sed -n "1p;3,+1p" test.txt和sed -n "1p;3,4p" test.txt一样，都是打印第1行和3到4行；

sed –n “”test.txt表示打印全部文件行；

sed -n "1~2p" test.txt表示打印从1行开始的2的倍数行，即1，3，5等；

sed -n "/you/Ip" test.txt和sed -n "\%you%Ip" test.txt一样，表示打印匹配you的行，其中大写的I意思是忽略大写写；

sed "1d;5,6d" test.txt 表示显示删除1行和5~6行后的文件，注意原文件没有改动；

sed "/you/Id" test.txt和sed "\%you%Id" test.txt一样，表示删除带you的行；

sed "/you/,/we/d" test.txt表示删除带you和we的行后显示的行；

显示p和删除d的格式可以通用，比如sed -n "/you/p" test.txt表示显示带you的行数，同理sed "1d;5,6d" test.txt表示删除第1行和5~6行；

sed -n "/We/! p" test.txt打印不包括We的行；

sed -n "/tell/,/boy/p" test.txt表示匹配包括tell和boy的行；

sed -n "/I.\*you/Ip" test.txt打印包含I到you的行；

sed -n "/I[^a-zA-Z]/p" test.txt匹配包含“I”，但后面不跟字母的行，比如It不会匹配，但I 或者I’am则会匹配；

注意，上面都是针对单个字符的匹配，如果要匹配一个字符串，要加小括号：比如打印包含1到2个连续“you”的行：sed -n "/\(you\)\{1,2\}/p" test.txt

sed -n "s/her/you/gp" test.txt= sed "s/her/you/g" test.txt不修改源文件，表示全局用you替代her；

sed -n "s/her/you/p" test.txt同上，只是一行只能第一个you被替代；

sed -i "s/her/you/g" test.txt 修改源文件，同上都是替代；

sed -i "s/\(tell\) \(you\)/\1-\2/g" test.txt表示将“tell you”替换成“tell-you”,源文件修改，如果g为p，则修改处的结果会多出1行；

sed -n "s/\(tell\) \(you\)/\1-\2/g"源文件不修改，功能同上；

sed "/again/r aa" test.txt读命令，表示文件中出现again行的后一行加上aa的内容；

sed -n "/again/Iw aa" test.txt写命令，表示将文件中出现again的行写进aa文件中，如果有内容会覆盖，不带-n会显示全部test.txt内容；

sed "/see you again/a \your pretty face" test.txt表示在see you again后一行加上your pretty face；

sed "/see you again/i \your pretty face" test.txt 与上的区别在于在see you again 前面一行加上your pretty face;

sed "/see you again/c \your pretty face" test.txt将see you again 该行替换成your pretty face;

sed "/you/{n;d}" test.txt找到you这一行，并删除它的下一行；

sed "1 e date" test.txt在第一行加上时间，如Tue Oct 24 17:18:54 CST 2017；

sed -f aa test.txt == sed -e "s/you/her/g" -e "s/ /,/g" test.txt表示利用aa中的命令执行test.txt文件，aa内容如下： s/you/her/g s/ /,/g

sed "/again/q" test.txt 只在test.txt中匹配到出现again的行；

1. terminal中，ctrl+A表示跳到命令的开头；ctrl+E表示跳到命令的结尾；
2. NB chx002 grd环境：/cpuwrk/chx002/sim/chx002\_grd grd1 peg2 GOLD；

2018/12/26

1. 在python中，“.”可以匹配任何字符，但要匹配自身，需要加“\”；
2. NB grd 环境：/cpuwrk/chx002/sim/chx002\_grd grd1 peg2 GOLD
3. 看机器上空间大小：dfhl
4. g –d instance.v instance.orig 用来比较两个文件的不同。
5. 无法关闭文件时：在对应的机器上gps查看文件号，然后“kill 文件号”即可。
6. >>> print(‘C:\some\name') # here \n means newline!

C:\some

ame

>>> print(r‘C:\some\name') # note the r before the quote

C:\some\name

7、当导入包时，Python 会通过 sys.path 搜索路径查找包含这个包的子目录。

为了让 Python 将目录当做内容包，目录中必须包含 \_\_init\_\_.py 文件；

1. Python 有办法将任意值转为字符串：将它传入 repr() 或 str() 函数。函数 str() 用于将值转化为适于人阅读的形式，而 repr() 转化为供解释器读取的形式（如果没有等价的语法，则会发生 SyntaxError 异常）某对象没有适于人阅读的解释形式的话，str() 会返回与 repr() 等同的值。
2. 恢复文件：g .snapshot/daily\_19.2017-11-1

12/26/2018

1. python 中if的5种形式：if … is not None:/if … is None:/ if + logic express:/if…in…:/if…not in…
2. +:一元一致性，-：一元不一致性；~：按位补码；x.attribute:特性引用。
3. 绑定到对象特性上面的函数称为方法。如object.getprice().
4. 百灵鸟类是鸟类的子类（subclass）,鸟类是百灵鸟类的超类（superclass）
5. 字符串（以及数字和元组）是不可变的，即无法修改，只能用新的值覆盖。
6. from xml.sax.handler import ContentHandler

from xml.sax import parse

from optparse import OptionParser

xml是可扩展标记语言，是对超文本标记语言（HTML）的补充，用来传输和存储数据，其焦点是数据的内容，而超文本标记语言是用来显示数据，其焦点是数据的外观，xml是独立于软件和硬件的信息传输工具。

在 XML 语法分析器中，事件与用户操作无关，而与正在读取的 XML 文档中的元素有关。有对于以下方面的事件：元素开始和结束标记，元素内容，实体，语法分析错误。

语法分析器传递了足够信息以构建 XML 文档的文档树，但是与 DOM 语法分析器不同，它并不显式地构建该树。经验法则是在需要更多控制时使用 SAX；要增加方便性时，则使用 DOM。

使用Python解析XML的时候，需要 import xml.sax 和 xml.sax.handler。

parse方法，创建一个SAX解析器并解析xml文档：

xml.sax.parse(filename\_or\_stream, handler[, error\_handler])

* **filename\_or\_stream**：xml文件名
* **handler**：必须是一个ContentHandler的对象
* **error\_handler**：如果指定该参数，errorhandler必须是一个SAX ErrorHandler对象

characters(content)方法

\*调用时机

\*从行开始，遇到标签之前，存在字符，content的值为这些字符串。

\*从一个标签，遇到下一个标签之前， 存在字符，content的值为这些字符串。

\*从一个标签，遇到行结束符之前，存在字符，content的值为这些字符串。

\*标签可以是开始标签，也可以是结束标签。

1. **gen\_pad\_mon\_table.py**：
2. from optparser import OptionParser

两种方式来初始化一个parser，一是带有usage选项，这样的好处是可以模拟带有帮助的终端使用，如下：

usage = "myprog[ -f <filename>][-s <xyz>] arg1[,arg2..]"

parser=OptionParser(usage) #这里为类添加了参数usage

二是不带usage选项，正常来说可以不用设置参数：

parser = OptionParser()

作为核心就是指add\_option的其他的参数。有如下几个：

action: 验证输入数据类型是否和type 匹配，并将符合要求的这个参数存储到dest变量中。有以下几个属性：

store 默认值

- store\_false 标记而已

配合下边的那个store\_true来进行代码的“标记”，辅助流程控制。

-store\_true 标记而已

type ： 指的是对应于参数，如-f,-n等的接下来的那个参数的数据类型，有string，int，float等等

dest ： 用于保存临时变量，其值可以作为options的属性进行访问。存储的内容就是如-f,-n 等紧挨着的那个参数内容。

help： 提供用户友好的帮助信息，一般可以用来解释本add\_option方法的功能阐述。

关于default参数的作用，其实就是给dest的默认值。

使用optionparser模块来解析

optionparser的执行过程：

导入optionparser ： from optparse import OptionParser

构造optionparser的对象：parser = OptionParser()

往optionparser对象中增加option ：parser.add\_option()

调用optionparser的解析函数：(options, args) = parser.parse\_args()

在options中使用解析到的options，在args中使用其他的args。

add\_option()中参数的意义：

short option string: 为第一个参数，表示option的缩写，例如-f;

long option string: 为第二个参数，表示option的全拼，例如--file;

后面的参数皆为命名参数，命名参数为可选参数;

action=: 表示对此option的处理方式，默认值为store，表示存储option的值到解析后的options对象的成员中。action还可以有其他的值：对于bool值，使用store\_true来默认存储true，使用store\_false来默认存储false，store\_const用来存储const设置的值到此option，append表示增加option的参数到list中，此时此option是一个list，可能包含多个值，count表示对counter增加1，callback表示调用指定的函数。

该项会影响metavar，使之不显，另外也会影响default和dest，使他们的值不显，而是现实自己存储的值，如store\_true，无论default如何，都是True（有文件输入时，否则还是default的值）。

所有的action值如下：

store + store\_true + store\_false + store\_const + append + count + callback

type=:表示此option的值的类型，默认为string，可以指定为string, int, choice, float and complex；

dest=：表示此option在经过optionparser解析后的options对象中成员的名字，默认使用long option string；

default=:表示此option的默认值；

metavar=:表示显示到help中/option的默认值；

const=:当action为store\_const的时候，需要设置此值；

choices=:当设置type为choices时，需要设置此值；

如果options很多的时候，可以进行分组，使用如下：

group = OptionGroup(parser)

group.add\_option()

parser.add\_option\_group(group)

1. def cat(\*args):调用多个tuple参数，如果只有一个参数，obj=args[0],否则obj=args；如果obj==None，返回空字符串，否则判断obj是否是tuple或list类型，是的话for item in obj:string = string+cat(item),string初始值为0，obj不是tuple类型的话将其强制转换为string。总结其作用是将cat的args转换为字符串（string）。
2. def is\_match\_case(m\_case):判断m\_case列表不为空，否则返回False。
3. class file\_info(object):def \_\_init\_\_中FILE的默认值为None，如果更改则将FILE中的所有行以list的形式读出（readlines(),注：readline（）是每次只读一行），并赋值给self.\_file\_list。如三行的FILE的赋值形式为：[‘love\n', ‘the world\n', ‘eric\n']。
4. class pin\_grp\_file\_info(file\_info):该类为file\_info的子类，两者具有继承关系。pin\_group.def文件传给该class下的FILE。目的是建立一个词典，通过查询group来确定hier和owner。
5. options.func\_mux\_file: 如果有func\_mux\_file文件输入，结果不为False，为该文件名，进行gen function mux table操作，此时如果后面的参数args不为3，就错误退出，否则进行异常（try/except）检查，执行一系列class操作。

如果没有func\_mux\_file文件，只有参数（args），如果参数满足7个，则进行gen pad mon table操作，并进行异常（try/except）检查，执行一系列class操作。

except IOError是指忽略“试图打开不存在文件时引发的error”；

Exception: 所有异常的基类

AttributeError：特性引用或赋值失败时引发

IndexError：在使用序列中不存在的索引时引发

KeyError：在使用映射中不存在的键时引发

NameError：在找不到名字（变量）时引发

SyntaxError：在代码为错误形式时引发

TypeError：在内建操作或者函数应用于错误类型的对象时引发

ValueError：在内建操作或者函数应用于正确类型的对象，但是该对象使用不合适的值引发

ZeroDivisionError：在除法或者模除操作的第二个参数为0时引发

1. func\_mux\_info(FUNC\_MUX):从func mux file（func\_mux.v）中获取信息。class func\_mux\_info是基类file\_info的子类，参数FUNC\_MUX传递给class func\_mux\_info中def \_\_init\_\_(self,FILE)中的FILE；然后传递给基类file\_info，如果非None则将FILE中的所有行以list的形式读出.

之后进行函数def get\_info(LOG):其中self.\_file\_list = FILE.readlines()，info为func.mux.v文件的list形式，然后self.\_info\_map(info)，转移到函数self.\_info\_map(self,info)函数，其中的self.\_title = re.sub(r'\s+','',info[0])是将该list元素中的空格摘除，该函数的主要目的是为了将赋值的ball name 统一append到mux\_ball\_list中。

该class func\_mux\_info会建立一个func\_grp词典，存入关键词为pin\_name,ball\_name和attr的value。

1. class gen\_mux\_table：用来产生\*\_function\_group.v文件。
2. class macro\_def\_file\_info：用来产生嵌套词典，key pin\_grp的value是self.\_macro,key self.\_macro的value是self.\_inst
3. class pin\_list\_file\_info:
4. **gen\_pad\_check.py**：
5. def cat(\*args):调用多个tuple参数，如果只有一个参数，obj=args[0],否则obj=args；如果obj==None，返回空字符串，否则判断obj是否是tuple类型，是的话for item in obj:string = string+cat(item),string初始值为0，obj不是tuple类型的话将其强制转换为string。总结其作用是将cat的args转换为字符串（string）。

2）def is\_match\_case(m\_case):判断m\_case列表不为空，否则返回False。

3）def get\_xtalok\_case(pad\_hier):将各个engine的PIN\_\*\*\*在pad\_hier中进行匹配，将结果append到m\_case中，然后返回m\_case。

4）def get\_pcirst\_case(pad\_hier):同上，针对pci。

5）class file\_info(object):def \_\_init\_\_中FILE的默认值为None，如果更改则将FILE中的所有行以list的形式读出（readlines(),注：readline（）是每次只读一行），并赋值给self.\_file\_list。如三行的FILE的赋值形式为：[‘love\n', ‘the world\n', ‘eric\n']。

6）class IRSHandler:

7）

12/26/2018

1 UTF-8（8-bit Unicode Transformation Format）是一种针对Unicode的可变长度字符编码，又称万国码。使用2个字节就可以编码地球上几乎所有地区的文字，这就是UNICODE编码。

UTF-8编码规则：如果只有一个字节则其最高二进制位为0；如果是多字节，其第一个字节从最高位开始，连续的二进制位值为1的个数决定了其编码的字节数，其余各字节均以10开头。

2 module level的环境： /cpuwrk/chx002/users/franco/get\_dramc\_env

3 grd 脚本：/logic/franco/script/run\_grd

4 dramc module level只能跑dramc\_开头的checklist。make后面不要加“.src”

5 Hi, Eric

DDR4 sepc重点要看的章节是第3、4、5章，第三章有5小节，第四章有33小节，第五章有6小节，加上准备PPT的时间共计两周

下面是这个sechdule做为参考，你可以根据自己的情况做调整

|  |  |
| --- | --- |
| 第二章、第三章 | 2天（11/21-11/22） |
| 第四章 | 3.5天(11/23-11/28) |
| 第五章 | 1.5天(11/28-11/29) |
| DIMM spec | 1天(11/30-11/30) |
| 准备PPT | 2天(12/1-12/4) |

6 GOLD: this is used to set register value in mixcfg file. If GOLD is defined then default value is used.

ALLONE: If ALLONE is defined register can be toggled is assigned 1.

ALLZERO: If ALLZERO is defined register can be toggled is assigned 0.

RDMCFG: If RDMCFG is defined register can be toggled is random value.

7 带宽对于DDR来讲是指数据的传输能力，即每秒传输多少Byte，单位一般为GB/s.

8 现代计算机中内存空间都是按照byte划分的，从理论上讲似乎对任何类型的变量的访问可以从任何地址开始，但实际情况是在访问特定类型变量的时候经常在特定的内存地址访问，这就需要各种类型数据按照一定的规则在空间上排列，而不是顺序的一个接一个的排放，这就是对齐.

DW对齐需要地址数据的后2bit为固定（4Byte=2^2）；

QW对齐需要地址数据的后3bit为固定（8Byte=2^3）；

4QW对齐需要地址数据的后5bit为固定（32Byte=2^5）；

8QW对齐需要地址数据的后6bit为固定（64Byte=2^6）.

（注：计算方式其实就是地址位数和存储容量之间的对应关系。比如32位操作系统，一般对应的就是4G内存）

9 page size: it is the number of byte of data delivered from the array to the internal sense amplifiers when an Active command is registered. Page Size is per bank.

Page Size = 2^(列地址数)\*（I/Obit数）/8 （单位：B）

I/O bit数对于DRAM颗粒而言就是单个颗粒的存储位宽（x4/x8），而对整个内存而言就是一次传输的位宽为64bit。

bank中的每一行组成一个page。

10 DDR的总位宽是64bit，4-Gb DDR4x8:表示DRAM单个颗粒大小为4Gb，每个颗粒的位宽为8bit，所以总共有8颗颗粒，大小为32Gb。

11 8bit预取技术。直观的解释是在I/O控制器发出访问请求之前，存储单元已经预先准备好了8bit数据。具体实现步骤是：Logic Bank一次传送8bit的数据到数据锁存器，再分为两路4bit数据传送到复用器，最后由差分时钟电路分上下沿两次分别传输4bit到内存控制器。打个形象的比方，这就好像是一个蓄水池里接了8根进水管，却只接了一根出水管，出水速度将是相当大的。其实通俗理解起来，就是将8个存储单元并联起来，在一个时钟周期内同时发送数据到I/O，然后由I/O一起传输出去。每个时钟周期上沿传输的数据不是1bit而是4bit，因此200MHz的内部时钟其实等效于800MHz的外部时钟，也就是I/O控制器工作的频率。

接下来我们知道在CPU-Z中这个显示的800MHz要乘以2才等于1600MHz，这又是为什么呢？这是因为DDR的全称是Double Date Rate，双倍速率，通过差分时钟电路CLK#和DQS，可以在时钟上下沿同时传输数据。也就是说，I/O控制器的工作频率为800MHz，也就是1秒钟800兆个周期；但是在每一个时钟周期内，信号增强时（上沿）和衰减时（下沿）I/O都可以传输一次数据，也就是一周期两次数据，因此800兆个周期可以传输1600次数据，等效频率就为1600MHz了。

DDR5支持16bit预取机制（DDR4为8bit,也支持16bit），同样内核频率下理论速度是DDR4的两倍。

10 作为对single-sided和double-sided的补充，DIMM还被分为single-rank和double-rank（也就是我们在内存的label上经常能看到的1R，2R）。内存的rank的定义，是指在DIMM上通过一部分或者所有的内存颗粒产生的一个64-bit的area或者说block。对ECC DIMM来说，一个内存rank有72个数据位（64 bits data+8 bits ECC).

ECC的作用可以概括为：1bit纠错，2bit report.

ECC support:

ECC can correct one bit and report two bits data error from DRAM 64 bit device. ECC function is transparent for all masters, DRAMC will generate ECC message (8 bits) when write cycle and check ECC message (8 bits) when read cycle.

RMW function for ECC:

ECC message are generated by 64bit data, so Read-Modify-Write operation are executed if all BE in write datas are not opened together (partial write). The RMW operation is reading same adderss from memory, modifying(merging) related data, and writing full-write data with correct ECC message to memory.

(就是将相同地址中原来的数据和新来的部分改动数据进行merge，保留不动的部分，改动新数据的部分，这样就可以保持write data bus 的width不变)

ECC 一般存在服务器中； ECC的纠错，report等都是在DRAMC中完成，和DDR device 没有关系。

不同于ECC，CRC只support write,对read cycle没有效(DDR4中如是，DDR5中也支持read了)，CRC是在timing上进行操作，而不是data bus上，另外，DDR device会对CRC的报错拉ALERT信号，也就是说CRC 操作和DDR device有关系。ALERT信号低有效，除了针对CRC ERROR，还有command and address parity error,如果DRAMC检测到ALERT信号assert，就会recovery所有刚发的cycle，即DRAMC内部有单独的module来store request cycle.

11 DDR4架构上采用了8n预取的Bank Group分组，包括使用两个或者四个可选择的Bank Group分组，这将使得DDR4内存的每个Bank Group分组都有独立的激活、读取、写入和刷新操作，从而改进内存的整体效率和带宽。如此一来如果内存内部设计了两个独立的Bank Group，相当于每次操作16bit的数据，变相地将内存预取值提高到了16n，如果是四个独立的Bank Group，则变相的预取值提高到了32n。

如果说Bank Group是DDR 4内存带宽提升的关键技术的话，那么点对点总线则是DDR4整个存储系统的关键性设计，对于DDR3内存来说，目前数据读取访问的机制是双向传输。而在DDR4内存中，访问机制已经改为了点对点技术，这是DDR4整个存储系统的关键性设计。

12 在DDR3内存上，内存和内存控制器之间的连接采用是通过多点分支总线来实现。这种总线允许在一个接口上挂接许多同样规格的芯片。我们都知道目前主板上往往为双通道设计四根内存插槽，但每个通道在物理结构上只允许扩展更大容量。这种设计的特点就是当数据传输量一旦超过通道的承载能力，无论你怎么增加内存容量，性能都不见的提升多少。这种设计就好比在一条主管道可以有多个注水管，但受制于主管道的大小，即便你可以增加注水管来提升容量，但总的送水率并没有提升。因此在这种情况下可能2GB增加到4GB你会感觉性能提升明显，但是再继续盲目增加容量并没有什么意义了，所以多点分支总线的好处是扩展内存更容易，但却浪费了内存的位宽。

因此，DDR4抛弃了这样的设计，转而采用点对点总线：内存控制器每通道只能支持唯一的一根内存。相比多点分支总线，点对点相当于一条主管道只对应一个注水管，这样设计的好处可以大大简化内存模块的设计、更容易达到更高的频率。不过，点对点设计的问题也同样明显：一个重要因素是点对点总线每通道只能支持一根内存，因此如果DDR4内存单条容量不足的话，将很难有效提升系统的内存总量。当然，这难不道开发者，3DS封装技术就是扩增DDR4容量的关键技术。

13 DBI: data bus inversion. Mitigate(减轻，缓和) I/O noise and power.

CAL: CMD/ADDR latency. Efficient command scheduling.

CRC: data bus write CRC, error detection of data traffic.

Gear-down mode: supper CMD/ADDR faster tck.

Connectivity test mode: improve “part down” manufacturability.

Programmable READ and Write Preambles: improved data bus signaling.

READ preamble training: improve READ training.

ACT\_n: multiplexs 3 inputs.

TCR: temperature controlled refresh, lower refresh current without impacting controller.

Fine Granularity Refresh: Efficient refresh scheduling.

MPR: Multi-Purpose Register. Provides additional specialty readout.

Write Leveling: De-skew fly-by layout/topology used by modules.

DQ Driver(STD): Optimized for DIMM applications.

DQ Driver(ALT): Optimized for PtP applications.

DQ Bus Termination: Optimized for higher data rates.

DQ Bus: Mitigate I/O noise and power.

14 CKE高有效，读写访问时维持高；为低时提高precharge power-down and self-refresh operation(all banks idle), or active power-down(row active in any bank).

CKE is synchronous for self-refresh exit.在自刷新期间，ODT功能失效，power down mode不会表现任何refresh operation。

在self-Refresh operation期间，VPP和VREF(VrefCA) must be maintained.

15 地址总线A0-A17：在ACT命令中作为行地址，在读写命令中作为列地址，从而可定位到存储阵列中的确定位置。(A10/AP, A12/BC\_n, RAS\_n/A16, CAS\_n/A15 与 WE\_n/A14可作为额外的地址总线使用。在MRS命令中，地址总线还作为操作码使用，即写入模式寄存器的值。A17仅在x4系统中可用。

在ACT命令发出时，相应的地址会选定需要激活的BANK地址与ROW地址

16 RESET\_n和TEN两个信号为轨到轨的CMOS类型的信号，DC高低电平分别为VDD的80%与20%。

17 存储控制器通过配置MR1寄存器的A7为1来启动Write Leveling（以下简称WL）。一旦进入WL模式，DQ引脚就会进入未定义的驱动模式。在WL过程中，仅有DESELECT命令可以发送，就像修改Qoff（MR1[12]）位以及退出WL（MR1[7]）位的命令允许被发出来。当退出WL模式时，MRS命令在改变MR1[7]的同时，也会更改A12-A8与A2-A1。由于在控制层面每次仅使能一个RANK，所有其他Rank的输出将会被禁用，通过设置MR1[12]为1，可达到此目的。控制器会在tMOD时间后将ODT信号置1，此时DRAM已经准备好接受ODT信号。

18 CAS latency: 列地址选通延迟

RAS latency:行地址选通延迟

19 2tck WL = CWL + 1=10; differ bank group 的read/write transaction中的tWTR或tCCD为short， same bank group的为long。其中tCDD\_S = 4，tCDD\_S/L = 5(1tCK),tCDD\_L = 6(2tck),tWTR\_S = 2, tWTR\_L = 4.

20 RDIMM即Registered DIMM，表示控制器输出的地址和控制信号直接到达DIMM上的DRAM芯片; Registered内存本身有两种工作模式，即 Registered模式（寄存器模式）和Buffered（缓冲器模式）模式。在支持Registered工作模式的主板上工作时，Registered内存工作于 Registered模式，这时主板上的地址信号和控制信号会比数据信号先一个时钟周期到达DIMM，送入Register芯片后会在其中停留一个时钟周S期，然后在下一个时钟信号的上升沿从Register输出，与此时从主板上到达DIMM的数据信号一起同时传送到SDRAM。当Registered内存工作在普通的主板上时，为Buffered工作模式，这时所有的信号也基本上是同时到达DIMM再同时传送到SDRAM，Register芯片这时在功能上只相当于一个简单的Buffer，其输入到输出之间是直通的，只简单的起到改善地址信号和控制信号的作用，时序上与Unbuffered内存是一样的。 （一般用于服务器）

为了保证并行传输的有效性，RDIMM在内存条上加了一个寄存器进行转发。它位于CPU和内存颗粒之间，这样就减少了并行传输的距离。同时由于寄存器效率很高，因此RDIMM的密度和频率就容易提高。RDIMM目前是较为主流的内存条，单条容量在2～32GB之间，频率也有1.33GHz和1.6GHz两种选择。绝大多数2路通用配置的服务器出厂时通常都会配置这种类型的内存.

UDIMM:无缓冲双信道内存模块 (Unbuffered Dual In-Line Memory Modules，UDIMM).（一般常用的内存条，用于一般家用）. UDIMM 表示控制器输出来的地址和控制的信号直接到达DIMM的DRAM芯片上。它不支持服务器内存满配，就是最高容量了，因为使用UDIMM内存时最大使用每通道只能用2个插槽，但支持3通道，所以只能每边插6条，一共12条内存，不能插满18个插槽，虽然性能会有所下降，但是对于预算比较有限的用户来说，是个很好的方案。

当数据从CPU传到每个内存颗粒时，UDIMM要求保证CPU到每个内存颗粒之间的数据传输距离相等，这样并行传输才会有效。这需要极高的制造工艺，极难做到高密度、高频率。因此UDIMM容量和频率都较低。不过，UDIMM由于在CPU和内存之间没有任何缓存，因此同频率下时延较小。目前常见的是单条容量2GB/4GB，最高主频也只能到达1.33GHz。

21自刷新模式包含一种自动自刷新的机制，DRAM内含一个温度传感器，根据此传感器的数据，可自动选择刷新命令的间隔与周期。

22 在SR模式中，是可以完全保证数据正确性的。而在PD模式中，PD模式持续的时间如果不大于9\*tREFI，则DRAM可以保证数据的正确性。

SR模式通过SRE命令进入，SRX命令退出。PD模式则是在DES命令下，将CKE信号拉低成低电平来进入，通过将CKE信号拉高成高电平来退出，并且CS\_n为高。

在SR模式中，所有IO Buffer都处于打开状态。

在PD模式中，除了CK\_t, CK\_c, CKE与RESET\_n的IO Buffer，其他所有的IO Buffer都是处于关闭状态的。

23 Bank Group架构具体来说就是每个Bank Group可以独立读写数据，这样一来内部的数据吞吐量大幅度提升，可以同时读取大量的数据，内存的等效频率在这种设置下也得到巨大的提升。DDR4架构上采用了8n预取的Bank Group分组，包括使用两个或者四个可选择的Bank Group分组，这将使得DDR4内存的每个Bank Group分组都有独立的激活、读取、写入和刷新操作，从而改进内存的整体效率和带宽。如此一来如果内存内部设计了两个独立的Bank Group，相当于每次操作16bit的数据，变相地将内存预取值提高到了16n，如果是四个独立的Bank Group，则变相的预取值提高到了32n。

每个bank内部都包括行地址解码器，列地址解码器，传感放大器以及DRAM内存阵列。

对于DDR3，我们通常说它是8n-prefetch(这儿n是指每个rank的bank数目)，因为DDR3，每个颗粒有8个bank，**每个bank读取数据的最小单位是1bit**，一个byte。每次数据读取request，都会读取8\*8bit=64bit data，而不管这些数据是否都是我们所需要的，比如我们只需要其中的某个byte，但读request会读取8个byte。

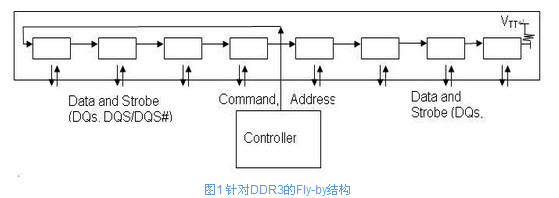
一个bank每次只能打开一个page(这儿打开是指把page内容放入到传感放大器)，对于处于打开状态的page，我们可以进行读写操作，如果不需要再对该page进行读写操作，可以关闭该page, 把该page内容写入bank的行列单元对应的page中，以便对其它page进行读写操作。这个关闭操作通过发射一个Precharge命令实现（即将传感放大器中的内容放入page中），precharge命令可以关闭某一个bank，也可以关闭rank中所有打开的bank。

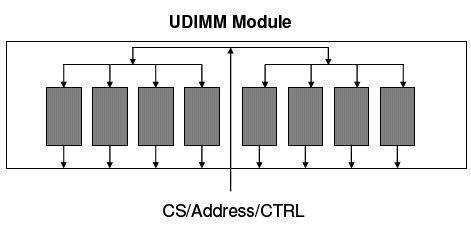
24 DDR4内存采用了TCSE ( Temperature Compensated Self-Refresh，温度补偿自刷新，主要用于降低存储芯片在自刷新时消耗的功率)、TCARtemperature Compensated Auto Refresh，温度补偿自动刷新，和T CSE类似）、DBI（Data Bus Inversion，数据总线倒置，用于降低VDDQ电流，降低切换操作）等新技术。

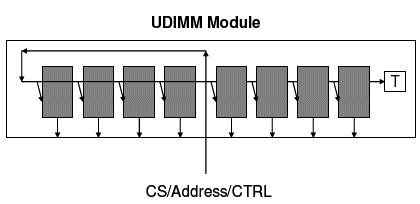
25 3DS（3-Dimensional Stack，三维堆叠）技术是DDR4内存中最关键的技术之一，它用来增大单颗芯片的容量。

26 存储器控制器（DRAMC）延迟了DQS，一次一步(step)，直到检测到CK信号从0过渡到到1。这将再次对齐DQS和 CK，以便DQ总线上的目标数据可以可靠地被捕获。由于这是由DDR3存储器控制器自动做的，电路板设计人员无须担心实施的细节。设计人员会从额外的裕度中得到好处，这是由DDR3存储器控制器中的写调整（write leveling）的特性所创建的。

fly-by信号是命令、地址，控制和时钟信号。如下图所示，源于存储器控制器的这些信号以串行的方式连接到每个DRAM器件。通过减少分支的数量和分支的长度改进了信号完整性。然而，这引起了另一个问题，因为每一个存储器元件的延迟是不同的，取决于它处于时序的位置。通过按照DDR3规范的定义，采用读调整和写调整技术来补偿这种延迟的差异（write leving, read leveling）。fly-by拓扑结构在电源开启时校正存储器系统。这就要求在DDR3控制器中有额外的信息，允许校准工作在启动时自动完成。



T-branch topology

Fly-by topology

27 DRAM读取具有破坏性，也就是说，在读操作中会破坏内存单元行中的数据。因此，必需在该行上的读或写操作结束时，把行数据写回到同一行中。这一操作称为预充电，是行上的最后一项操作。必须完成这一操作之后，才能访问新的行，这一操作称为关闭打开的行。

预充电用来在破环性读取后进行行数据的恢复工作，由于是读写操作的最后一步，称为行关闭步骤。

28 内存访问中最常用的类型是读取顺序的内存地址。这是合理的，因为读取计算机指令一般要比数据读取或写入更加常用。此外，大多数指令读取在内存中顺序进行，直到发生到指令分支或跳到子例程。

29 DRAM的一个行称为内存页面(page)，一旦打开行，您可以访问该行中多个顺序的或不同的列地址。这提高了内存访问速度，降低了内存时延，因为在访问同一个内存页面中的内存单元时，其不必把行地址重新发送给DRAM.结果，行地址是计算机的高阶地址位，列地址是低阶地址位。由于行地址和列地址在不同的时间发送，因此行地址和列地址复用到相同的DRAM 针脚上，以降低封装针脚数量、成本和尺寸。一般来说，行地址尺寸要大于列地址，因为使用的功率与列数有关。

30 DRAM 内存单元必需刷新，避免丢失数据内容。这要求丢失电荷前刷新电容器。刷新内存由内存控制器负责，刷新时间指标因不同DRAM内存而不同。内存控制器对行地址进行仅RAS# 循环，进行刷新。在仅RAS# 循环结束时，进行预充电操作，恢复仅RAS# 循环中寻址的行数据。一般来说，内存控制器有一个行计数器，其顺序生成仅RAS# 刷新周期所需的所有行地址。

31 早期的DRAM 演进及实现了DRAM IC 上的刷新计数器，处理顺序生成的行地址。在DRAM IC 内部，刷新计数器是复用器输入，控制着内存阵列行地址。另一个复用器输入来自外部地址输入针脚的行地址。这个内部刷新计数器不需要内存控制器中的外部刷新计数器电路。部分DRAM 在RAS# 周期前支持一个CAS#,以使用内部生成的行地址发起刷新周期。

32 Activate （激活）命令向SDRAM发送一个行地址，打开内存的一个行（页面）。然后是一个Deselect （反选）命令序列，在对列地址发送Read 或Write 命令前满足定时要求。一旦使用Activate命令打开内存的行（页面），那么可以在内存的该行（页面）上运行多个Read和Write命令。要求Precharge（预充电）命令，关闭该行，然后才能打开另一行。

33 降低工作电压可以降低内存系统的功率；它的的另一个优势是降低了逻辑电压摆幅。在转换速率相同时，电压摆幅下降会提高逻辑转换速度，支持更快的时钟速率。此外，数据选通可以编程为差分信号。使用差分数据选通信号降低了噪声、串扰、动态功耗和EMI （电磁干扰），提高了噪声余量。差分或单端数据选通操作配置有DDR2 SDRAM 扩展模式寄存器。

34 DDR2 SDRAM 引入的一种新功能是附加时延（AL），它使得内存控制器中心能够在Activate命令后，更快地灵活发送Read 和Write 命令。这优化了内存吞吐量，通过使用DDR2 SDRAM扩展模式寄存器对附加时延编程来配置。DDR2 SDRAM使用八个内存条，改善了1Gb和2GbDDR2 SDRAM 的数据带宽。通过隔行扫描不同的内存条操作，八个内存条提高了访问大型内存DDR2 SDRAM的灵活性。此外，对大型内存，DDR2 SDRAM支持最多八个内存条的突发长度。

35 一个重要差异是DDR2 模式寄存器规定了读出操作的CAS 时延，写入时延则是1减去模式寄存器读出时延设置。DDR3模式寄存器对CAS 读出时延和写入时延的设置是唯一的。

36 DDR3 SDRAM 模式寄存器可以编程为支持飞行突变，这会把传送8个数据字缩短到传送4个数据字，这在读出或写入命令期间把地址行12 设为低来实现。飞行突变在概念上与DDR2 和DDR3 SDRAM 中地址行10 的读出和写入自动预充电功能类似。

37 值得一提的另一个DDR3 SDRAM属性是差分的数据选通信号DQS,DDR2 SDRAM数据通信号则可以由模式寄存器编程为单端或差分。

DDR3新增ASR（Automatic Self-Refresh）、SRT（Self-Refresh Temperature）等两种功能，让内存在休眠时也能够随着温度变化去控制对内存颗粒的充电频率，以确保系统数据的完整性。

38 DDR4 新增4个Bank Group 数据组的设计，各个Bank Group具备独立启动操作读、写等动作特性，Bank Group 数据组可套用多任务的观念来想象，亦可解释为DDR4 在同一频率工作周期内，至多可以处理4 笔数据，效率明显好过于DDR3。

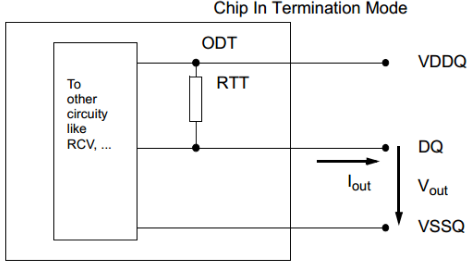
39 另外DDR4增加了DBI（Data Bus Inversion）、CRC（Cyclic Redundancy Check）、CA parity等功能，让DDR4内存在更快速与更省电的同时亦能够增强信号的完整性、改善数据传输及储存的可靠性。

40 SDRAM：为同步动态随机存取内存，SDRAM是为了与CPU的计时同步化所设计，这使得内存控制器能够掌握准备所要求的数据所需的准确时钟周期，因此CPU从此不需要延后下一次的数据存取。

41 POD（pseudo）作为DDR4新的驱动标准，最大的区别在于接收端的终端电压等于VDDQ，而DDR3所采用的SSTL接收端的终端电压为VDDQ/2。这样做可以降低寄生引脚电容和I/O终端功耗，并且即使在VDD电压降低的情况下也能稳定工作。

42 根据POD的特性，当数据为高电平时，没有电流流动，所以降低DDR4功耗的一个方法就是让高电平尽可能多，这就是DBI技术的核心。

43 为了提升信号质量, 从DDR2开始将DQ, DM, DQS/DQS#的Termination电阻内置到Controller和DRAM中, 称之为ODT (On Die Termination)。Clock和ADD/CMD/CTRL信号仍需要使用外接的Termination电阻。



在DRAM中，On-Die Termination的等效电阻值通过Mode Register (MR)来设置，ODT的精度通过参考电阻RZQ来控制，DDR4的ODT支持240, 120, 80, 60, 48, 40, 34 欧姆。

44 和DDR3不同的是，DDR4的ODT有四种模式：Data termination disable, RTT\_NOM,RTT\_WR, 和 RTT\_PARK。Controller可以通过读写命令以及ODT Pin来控制RTT状态，RTT\_PARK是DDR4新加入的选项，它一般用在多Rank的DDR配置中，比如一个系统中有Rank0, Rank1以及Rank2, 当控制器向Rank0写数据时，Rank1和Rank2在同一时间内可以为高阻抗(Hi-Z)或比较弱的终端(240,120,80,etc.), RTT\_Park就提供了一种更加灵活的终端方式，让Rank1和Rank2不用一直是高阻模式，从而可以让DRAM工作在更高的频率上。

45 众所周知，DDR信号一般通过比较输入信号和另外一个参考信号（Vref）来决定信号为高或者低，然而在DDR4中，一个Vref却不见了，先来看看下面两种设计，可以看出来，在DDR4的设计中，VREFCA和DDR3相同，使用外置的分压电阻或者电源控制芯片来产生，然而VREFDQ在设计中却没有了，改为由芯片内部产生，这样既节省了设计费用，也增加了Routing空间。DRAM内部VREFDQ通过寄存器(MR6)来调节，主要参数有Voltage range, step size, VREF step time, VREF full step time

46 每次开机的时候，DRAM Controller都会通过一系列的校准来调整DRMA端输入数据信号的VREFDQ，优化Timing和电压的Margin，也就是说，VREFDQ 不仅仅取决于VDD, 而且和传输线特性，接收端芯片特性都会有关系，所以每次Power Up的时候，VREFDQ的值都可能会有差异。

因为Vref的不同，Vih/Vil都会有差异，可以通过调整ODT来看Vref的区别，用一个仿真的例子来说明。对于DDR3，调整ODT波形会上下同步浮动，而调整DDR4 OOT的时候，波形只有一边移动。

47 single-ended单端总线，是指通过将单根连线上的电压和参考电压进行对比来获取一位数据信息。如果高于参考值则为1否则为0；

Differential差分信号，是指通过将两根连线上的电压值进行对比来获取一位数据信息。无论两根连线中哪根线上有相对较高的电压，这bit都会被读作1，否则为0.

差分总线允许更快的转换速度，因为它不易受到电气噪声的影响。如果单端信号的电压受到干扰而改变，就可能得到错误的数值。

48 带宽：数据每秒内或单位时间内传输的数据个数或数据的byte数；

位宽：数据每个周期可以传输的bit数，即一次能传输的数据量或数据宽度。

带宽 = 频率X 位宽（bit）/ 8 （byte）

49 NB负责和处理器之间的通信，以及和那些需要最高连接带宽的部件之间的通信。因为这部分通常包括主存储器，所以NB有时也被称为内存控制器（memory controller hub，MCH）。

SB又被成为输入输出控制器（Input output Controller hub, ICH）。

50 SRAM是静态随机访问存储器，只要供电还在数据就会一直保持，这种存储器需要6个晶体管才能存1bit数据；因为需要较多的硅片面积所以成本也较高，但SRAM的访问速度更快。

DRAM是动态随机访问存储器，所有电容上的电荷必须约每15ms更新一次，这种存储器只需要1个晶体管就能存1bit数据。

SDRAM是同步的DRAM，设计使用了共享的时序信号，来得到更高的数据传输带宽。

51 虽然可以对SDRAM总线时钟频率进行设置，允许每个时钟周期内进行一次数据传输，但这仅仅在连续地址传输时才可以，即burst mode，它决定了可能的最大数据带宽。

对非连续的地址访问时，存在额外的延时（AL），既提高最大数据带宽又改善了平均访问延时。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Abbr. | CS\_n | ACT\_n | cmd | | | CKE | | A10/AP |
| RAS/A16 | CAS/A15 | WE/A14 | Previous | Current |
| MRS | 0 | 1 | **0** | | | 1 | 1 |  |
| REF | 0 | 1 | **1** | | | 1 | 1 |  |
| SRE | 0 | 1 | **1** | | | 1 | 0 |  |
| SRX | 1 | x | xxx | | | 0 | 1 |  |
| 0 | 1 | **7** | | | 0 | 1 |  |
| ACT | 0 | 0 | Row Address | | | 1 | 1 |  |
| RD | 0 | 1 | **5** | | | 1 | 1 |  |
| WR | 0 | 1 | **4** | | | 1 | 1 |  |
| ZQCL | 0 | 1 | **6** | | | 1 | 1 | 1 |
| ZQCS | 0 | 1 | **6** | | | 1 | 1 | 0 |
| PRE | 0 | 1 | **2** | | | 1 | 1 | 0 |
| PREA | 0 | 1 | **2** | | | 1 | 1 | 1 |
| PDE | 1 | x | xxx | | | 1 | 0 |  |
| PDX | 1 | x | xxx | | | 0 | 1 |  |
| DES | 1 | x | xxx | | | 1 | 1 |  |

1 run\_soc: 只能rerun刚刚结束的checklist；

2 reg\_cat：用来run regress，比如reg\_cat DRAMC1, 其中DRAMC1是包含多条checklist的group，如果只想跑group中的一条checklist，只需要make+对应得checklist；

3 reg\_cat所跑的group在chx002\_rlist, chx002\_rlist中的checklist对应到makefile中。

关系示意图：reg\_cat->chx002\_rlist ->makefile

4 hg resolve -> hg merge

The below content is aimed to dramc module level env.

# How to get env

Execute script：/logic/franco/get\_dramc\_env

# How to run checklist

Execute cmd “make dramc\_chk\_basic1” at sim directory, and “dramc\_chk\_basic1” is an example of checklist that you want to run.

If you want customize some define when run checklist, you can add define after above cmd, just like “make dramc\_chk\_basic1 MACASE=MACSE4”.

# How to run grd

Execute cmd “make grd1 PARAM=GOLD NO\_GRD\_ENG=NONE” for example at sim directory.

And grd1 can be replaced by grd1-25, GOLD can be replaced by GOLD, ALLONE, ALLZERO, RDMCFG.

Therefore, there are 25x4=100 grd cases in dramc module level env.

no page mode: request with auto-precharge

page hit: row一直打开，所有cycle都在一个page中；

page conflict: row进行切换，cycle发到不同的page中，有precharge

VIM use：

1、折叠行的打开：zv 或‘空格’

2、折叠行的打开：zm

3、vim中包含include 的头文件，从源文件进入头文件命令：gf；返回命令：ctrl+o

4、vim中同时打开多个文件，使用：buffers查看各个文件的次序，然后使用b1,b2…来选择对应的文件。使用br是返回第一个文件，同b1.

make dramc\_chk\_basic1  HARDINIT=HARDINIT

奇偶检验：传输数据有偶数个1，奇校验时校验位置为1，偶校验时为0；传输数据有奇数个1，偶校验时校验位置为1，奇校验时为0。

即奇校验保证传输数据和校验位中1的总数为奇数；偶校验保证传输数据和校验位中1的总数为偶数。

$random%6表示随机-5~5；

Delay1 = 20\*({$random}%6); 表示delay1在0~100ns的范围变化。

Dout= 1<<({$random}%10); 表示dout的0~9位中随机出现1.

Python

对于简单的 object，用 shallow copy 和 deep copy 没区别

复杂的 object， 如 list 中套着 list 的情况，shallow copy 中的 子list，并未从原 object 真的「独立」出来。也就是说，如果你改变原 object 的子 list 中的一个元素，你的 copy 就会跟着一起变。这跟我们直觉上对「复制」的理解不同

pass :不做任何事情，只起到占位的作用

continue: 跳出本次循环

break：结束循环

exit（）：结束整个程序

DRAMC spec：

Cell：

颗粒中的一个数据存储单元叫做一个Cell，由一个电容和一个N沟道MOSFET组成。

内存中的cell按矩阵形排列，每一行和每一列都会有一个对应的行地址线路（正规叫法叫做word line）和列地址线路（正规叫法是bit line），每个具体的cell就挂接在这样的行地址线路和列地址线路上，对应一个唯一的行号和列号，把行号和列号组合在一起，就是内存的地址。

每个cell的容量是1byte。

寻址:

数据要写入内存的一个cell，或者从内存中的一个cell读取数据，首先要完成对这个cell的寻址。寻址的过程，首先是将需要操作的cell的对应行地址信号和列地址信号输入行/列地址缓冲器，然后先通过行解码器（Row Decoder）选择特定的行地址线路，以激活特定的行地址。每一条行地址线路会与多条列地址线路和cell相连接，为了侦测列地址线路上微弱的激活信号，还需要一个额外的感应放大器（Sense Amplifier）放大这个信号。当行激活之后，列地址缓冲器中的列地址信号通过列解码器（Column Decoder）确定列地址，并被对应的感应放大器通过连接IO线路，这样cell就被激活，并可供读写操作，寻址完成。从行地址激活，到找到列地址这段时间，就是tRCD。

Bank：

ddr中每个颗粒有8个bank，数据位宽共8bit。一根内存条是64bit，如果是单面就是8个8bit颗粒，如果是双面，那就是16个4bit的颗粒分别在两面，不算ECC颗粒。

Rank：

内存PCB的一面所有颗粒叫做一个rank，目前在Unbuffered台式机内存上，通常一面是8个颗粒，所以单面内存就是1个rank，8个bank，双面内存就是2个rank，8个bank。Bank与rank的定义是SPD信息的一部分，在AIDA64中SPD一栏可以看到。

(serial presence detect: 串行存在检测，内存上的EEPROM，以往BIOS必须侦测内存，有了SPD就不必每次做侦测动作，BIOS直接读取SPD来获取内存的相关信息。

SPD是内存模组上面的一个可擦写的ROM，里面记录了该内存的许多重要信息，诸如内存的芯片及模组厂商、工作频率、工作电压、速度、容量、电压与行、列地址带宽等参数。SPD信息一般都是在出厂前，由内存模组制造商根据内存芯片的实际性能写入到ROM芯片中。)

DIMM：

指一条可传输64bit数据的内存PCB，也就是内存颗粒的载体，算上ECC芯片，一条DIMM PCB最多可以容纳18个芯片。

CAS Latency（CL）：

CAS即Column Address Strobe，列地址信号，它定义了在读取命令发出后到数据读出到IO接口的间隔时间。由于CAS在几乎所有的内存读取操作中都会生效（除非是读取到同一行地址中连续的数据，4bit颗粒直接读取间隔3个地址，8bit颗粒直接读取间隔7个地址，这时候CAS不生效），因此它是对内存读取性能影响最强的。

tCAS（ns）=（CAS\*2000）/内存等效频率  
例如，DDR3-1333 CL9内存实际CAS延迟时间=（9\*2000）/1333=13.50 ns

相关的列地址被选中之后，将会触发数据传输，但从存储单元中输出到真正出现在内存芯片的 I/O 接口之间还需要一定的时间（数据触发本身就有延迟，而且还需要进行信号放大），这段时间就是非常著名的 CL。

DRAM RAS to CAS Delay（tRCD）：

RAS的含义与CAS类似，就是行（Row）地址信号。它定义的是在内存的一个rank（内存的一面）之中，行地址激活（Active）命令发出之后，内存对行地址的操作所需要的时间。每一个内存cell就是一个可存储数据的地址，每个地址都有对应的行号和列号，每一行包含1024个列地址，当某一行地址被激活后，多个CAS请求会被发送以进行读写操作。简单的说，已知行地址位置，在这一行中找到相应的列地址，就可以完成寻址，进行读写操作，从已知行地址到找到列地址过去的时间就是tRCD。当内存中某一行地址被激活时，我们称它为“open page”。在同一时刻，同一个rank可以打开8个行地址（8个bank，也就是8个颗粒各一个）。tRCD值由于是最关键的寻址时间，它对内存最大频率影响最大，一般想要上高频，在加电压和放宽CL值不奏效的时候，我们都要放宽这个延迟。

DRAM RAS(ROW) Precharge Time（tRP）：

RAS预充电时间。它定义的是前一个行地址操作完成并在行地址关闭（page close）命令发出之后，准备对同一个bank中下一个行地址进行操作，tRP就是下一个行地址激活信号发出前对其进行的预充电时间。由于在行地址关闭命令发出之前，一个rank中的多个行地址可能正在被读写，tRP对内存性能影响不如CL和tRCD。虽然tRP的影响会随着多个行地址激活与关闭信号频繁操作一个bank而加大，但是它的影响会被bank interleaving（bank交叉操作）和command scheduling（命令调配）所削弱。交叉读写会交替使用不同的bank进行读写，减少对一个bank的操作频率，从而可以提高performance；命令调配则是由CPU多线程访问不同的内存地址，同样是减少对一个bank的频繁操作次数。例如SNB CPU的内存控制器可以对读写操作命令进行有效地重新分配，以使得行地址激活命中率最大化（如果重复激活一个已经处于激活状态的行地址，那就是RAS激活命令未命中），所以tRP在SNB平台对性能的影响不大，并且放宽它有可能可以帮助提升稳定性。

DRAM RAS Active Time（tRAS）：

行地址激活的时间。它其实就是从一个行地址预充电之后，从激活到寻址再到读取完成所经过的整个时间，也就是tRCD+tCL的意思。这个操作并不会频繁发生，只有在空闲的内存新建数据的时候才会使用它。太紧的tRAS值，有可能会导致数据丢失或不完整，太宽的值则会影响内存性能，尤其是在内存使用量增加的时候。所以一般为了稳定性，我们设置**tRAS≥tRCD+CL+tRTP**即可（tRTP不是tRP，将在第二时序中介绍），尤其是PCB不好或者跑高频的时候，多几个周期比较稳妥。

DRAM Command Mode（Command Rate，CR）：

首命令延迟，也就是我们平时说的1T/2T模式。是指从选定bank之后到可以发出行地址激活命令所经过的时间。CR可能对性能的影响有比较大的变数：如果CPU所需要的数据都在内存的一个行地址上，就不需要进行重复多次的bank选择，CR的影响就很小；但是如果一个rank中同时多个bank要激活行地址，或者不同的rank中不同bank需要同时激活的时候，CR对性能的影响就会提升。但是随着内存频率的提升，CR=1T/2T的时间差越短，它的影响就会越来越小，这就是我们看到DDR1的时候1T/2T对性能影响挺大，但是到了DDR3影响就很小的其中一个原因。但是为了性能最大化，我们尽量把CR设为1T，但是如果bank数很多的时候，例如插满四条内存，就有32个bank，bank选择随机性增大，1T的首命令时间可能会不稳定。

所以，内存的基本读取操作的时序角度流程就是：预充电（Precharge）-激活行地址（Active）-寻找列地址并发送读取命令（CMD/Read）-读出数据(Data)，这四步操作之间的三个延迟就分别是tRP、tRCD和CL。

内存是根据行和列寻址的，当请求触发后，最初是tRAS（Activeto Precharge Delay），预充电后，内存才真正开始初始化RAS。一旦tRAS激活后，RAS（Row Address Strobe ）开始进行需要数据的寻址。首先是行地址，然后初始化tRCD，周期结束，接着通过CAS访问所需数据的精确十六进制地址。期间从CAS开始到CAS结束就是CAS延迟。所以CAS是找到数据的最后一个步骤，也是内存参数中最重要的。

**（注：内存条上的“8-8-8-24”这一类的数字序列，对应的参数为：CL-tRCD-tRP-tRASMin）**

DRAM CAS Write Latency（tWCL）：

列地址写入延迟，也就是DRAM的最小写入操作时间，与CL刚好是读写对应关系，一般跟CL值设为同一个值就是可以稳定的。由于内存读取之前必须先写入，所以这个值可以说与CL一样重要。但是在BIOS里一般没得设置，可能是与CL绑定了。

DRAM Row Cycle Time（tRC）：

行周期时间。定义了**同一bank**两次行激活命令所间隔的最小时间，或者说是一个bank中完成一次行操作周期（Row Cycle）的时间，即tRP+tRAS（预充电加上激活的整个过程），tRC设得太紧可能会直接点开不了机，一般只要能进系统再多加一两个周期都是可以稳定的。

DRAM Row Refresh Cycle Time（tRFC）：

行地址刷新周期，定义了一个bank中行地址刷新所需要的时间。重提一下刷新的含义，由于cell中电容的电荷在MOSFET关闭之后一段时间就会失去，为了维持数据，每隔很短一段时间就需要重新充电。这里多提一句，Intel平台和AMD平台对tRFC的含义不一样，AMD平台的tRFC是DRAM刷新延迟时间，单位是ns，通常有90/110/160/300几个值可以调整，也就是说它的tRFC时钟周期会随着频率的提升而提升；而Intel平台的单位则直接是时钟周期，相反地延迟时间会随着频率的提升而降低。容量大的bank行地址和cell会更多，刷新时间也更长，因此tRFC也要更高。另外，tRFC如果太快会导致数据出错，太慢则影响性能，但可以增加稳定性。

DRAM Refresh Interval（tREFI）：

内存刷新时间间隔，也就是内存的刷新命令生效前要经过的时间。刷新的时间间隔一般取决于内存颗粒的容量（density），容量越大，就越需要频繁刷新，tREFI值就要越低。另外tREFI的时间也会受到内存工作温度与内存电压（Vdimm）影响，因为温度越高电容漏电越快。一般在AMD主板的BIOS里，这个值只有3.9us和7.8us可选，而在SNB平台，则是按时钟周期算，例如DDR3-1333下默认值为5199T，换算过来就是2000/1333x5199=7800ns，也就是7.8us。一般DRAM颗粒的spec中都是规定工作温度大于85度时采用3.9us。

DRAM RAS to RAS Delay（tRRD）：

行地址间延迟，定义的是同一rank**不同bank**间两个连续激活命令的最短延迟，在DDR3时代一般最小是4T。它的作用和CR有点像，不过比CR更多的时候对性能有较大的影响，所以这个时序可尽量缩小。

DRAM Write Recovery Time（tWR）：

内存写入恢复时间，它定义了内存从写入数据结束到下一次预充电间隔的时间，也就是tRP的前一个操作。如果这个时间设得太短，可能会导致前一次写入未完成就开始下一次预充电，进行寻址，那么前一次写入的数据就会不完整，造成丢数据的情况。这个周期也是第二时序中比较长的，DDR3-2000一般需要10-14个周期，甚至更高。

因为选通三极管与电容的充电必须要有一段时间，所以数据写入存储电容需要一定的时间。为了保证数据的可靠写入，都会留出足够的写入/校正时间tWR， 这个操作也被称作写回（Write Back)。

DRAM Read to Precharge Time（tRTP）：

与tWR类似，定义了同一rank上内存从读取数据结束到tRP之前的间隔时间，但是它在读取完成并且行地址关闭之后才会生效。单颗128MB的内存颗粒可以在DDR3-2000下运行在4到6个时钟周期，如果bank容量增大时，这个时序有可能要放宽。

DRAM Four Active Window（tFAW）：

它定义了同一rank中允许同时发送大于四个行激活命令的间隔时间，因此最小值应该不小于tRRD的四倍。在DDR3上，tRRD的最小值是4T，因此tFAW的最小值就是16T。这个tFAW由于是在一个rank中大于四个bank同时激活之后才生效，因此在内存不是很繁忙的时候，它对性能的影响并不是很大。但是对一些频繁读写内存的操作（例如SuperPI 32M），tFAW对性能的影响可能会加大。由于现在内存用满的几率非常非常小，两根双面的内存更是有4个rank，配合上interleaving，一个rank中同时激活大于四个bank的几率应该不大，所以通常我们把它设为tRRD的四倍应该就不会出问题。

DRAM Write to Read Delay（tWTR）：

内存写-读延迟，它定义的是内存写入数据结束到下一个读取命令之间的时间间隔，最小为4T，与tRTP类似，提升内存的频率或者容量提升时，这个值需要提高。

对时序有了个大致的了解，现在应该可以知道一些时序设置时要注意什么了。比如tFAW要设为tRRD的四倍，tRAS不能设太低等等。还是那句话，内存是辅助CPU超频的，时序设置只是为了放开内存更多的超频空间，时序本身对性能的影响很小，并且随着频率的提升，或者bank数的增加，这种影响可能会进一步减小。

突发操作是指在同一行中相邻的存储单元连续进行数据传输的方式，连续传输所涉及到存储单元的数量就是突发长度（Burst Lengths，简称BL）。 但对于DDR来讲，由于采用了预取技术，突发长度不再指所连续寻址的存储单元数量，而是指连续的传输周期数。

突发传输技术，只要指定起始列地址与突发长度，内存就会依次地自动对后面相应数量的存储单元进行读/写操作而不再需要控制器连续地提供列地址。这样，除了第一笔数据的传输需要若干个周期（一般的是tRCD+CL）外，其后每个数据只需一个周期的即可获得。

(page open, 一行打开，众列读写)

差分时钟是DDR的一个必要设计，但对CK的作用，并不能理解为第二个触发时钟，而是起到触发时钟校准的作用。

DQS是DDR中的重要功能，它主要用来在一个时钟周期内准确区分出每个传输周期，并便于接收方准确接受数据。它实质上是数据真正的同步信号。每一颗芯片都有一个DQS信号线，它是双向的，在写入时它用来传送由DRAMC发来的DQS信号，读取时，则由DDR生成DQS向北桥发送。完全可以说，它就是数据的同步信号。

在读取时，DQS与数据信号同时生成（也是在CK与CK#的交叉点）。而DDR内存中的CL也就是从CAS发出到DQS生成的间隔，DQS生成时，芯片内部的预取已经完毕了，由于预取的原因，实际的数据传出可能会提前于DQS发生（数据提前于DQS传出）。

DQS 在读取时与数据同步传输，那么接收时也是以DQS的上下沿为准吗？不，如果以DQS的上下沿区分数据周期的危险很大。由于芯片有预取的操作，所以输出时的同步很难控制，只能限制在一定的时间范围内，数据在各I/O端口的出现时间可能有快有慢，会与DQS有一定的间隔，这也就是为什么要有一个tAC规定的原因。而在接收方，一切必须保证同步接收，不能有tAC之类的偏差。这样在写入时，DDR芯片不再自己生成DQS，而以发送方传来的DQS为基准，并相应延后一定的时间，在DQS的中部为数据周期的选取分割点（在读取时分割点就是上下沿），从这里分隔开两个传输周期。这样做的好处是，由于各数据信号都会有一个逻辑电平保持周期，即使发送时不同步，在DQS上下沿时都处于保持周期中，此时数据接收触发的准确性无疑是最高的。在写入时，以DQS的高/低电平期中部为数据周期分割点，而不是上/下沿，但数据的接收触发仍为DQS的上/下沿

由于芯片体积的原因，存储单元中的电容容量很小，所以信号要经过放大来保证其有效的识别性，这个放大/驱动工作由S-AMP负责，一个存储体对应一个S- AMP通道。但它要有一个准备时间才能保证信号的发送强度（事前还要进行电压比较以进行逻辑电平的判断），因此从数据I/O总线上有数据输出之前的一个时钟上升沿开始，数据即已传向S-AMP，也就是说此时数据已经被触发，经过一定的驱动时间最终传向数据I/O总线进行输出，这段时间我们称之为 tAC（Access Time from CLK，时钟触发后的访问时间）。

ODT（on-die termination, 片内终端），所谓termination，就是让信号在电路的终端被吸收掉，从而不会在电路上形成反射，引起噪声，破坏信号完整性。

在DDR3系统中，对于内存系统工作非常重要的参考电压VREF将分为两个信号，即VREFCA（服务于命令和地址信号）和VREFDQ（服务于数据总线）。这将有效地提高系统数据总线的信噪等级。

如果BL=4，那么也就是说一次就传送4×64bit的数据。但是，如果其中的第二笔数据是不需要的，怎么办？还都传输吗？为了屏蔽不需要的数据，人们采用了数据掩码（Data I/O Mask，简称DQM）技术。通过DQM，内存可以控制I/O端口取消哪些输出或输入的数据。这里需要强调的是，在读取时，被屏蔽的数据仍然会从存储体传出，只是在“掩码逻辑单元”处被屏蔽。DQM由北桥控制，为了精确屏蔽一个P-Bank位宽中的每个字节，每个DIMM有8个DQM 信号线，每个信号针对一个字节。这样，对于4bit位宽芯片，两个芯片共用一个DQM信号线，对于8bit位宽芯片，一个芯片占用一个DQM信号，而对于 16bit位宽芯片，则需要两个DQM引脚。

ZQCL: 上电初始化后，用完成校准ZQ电阻。ZQCL会触发DRAM内部的校准引擎，

      一旦校准完成，校准后的值会传递到DRAM的IO管脚上，并反映为输出驱动和ODT阻值。

ZQCS: 周期性的校准，能够跟随电压和温度的变化而变化。校准需要更短的时间窗口，

      一次校准，可以有效的纠正最小0.5%的RON和RTT电阻。

由于DDR3的预取为8bit，所以突发传输周期（BL，Burst Length）也固定为8(fixed 8)，而对于DDR2和早期的DDR架构的系统，BL=4也是常用的，DDR3为此增加了一个4-bit Burst Chop（突发突变）模式，即由一个BL=4的读取操作加上一个BL=4的写入操作来合成一个BL=8的数据突发传输，届时可通过A12地址线来控制这一突发模式。而且需要指出的是，任何突发中断操作都将在DDR3内存中予以禁止，且不予支持，取而代之的是更灵活的突发传输控制（如4bit顺序突发）。

Operation Mode for Software DDR Initialization

When BIOS does DDR initialization, it issues a memory read cycle to each rank on the DIMM. This bit defines

what type of memory command cycle it will appear on the DRAM bus for that read cycle.

000: Read Command. This is the Normal Mode, i.e., after the initialization, BIOS should set these registers to this value for the system to work normally.

001: NOP Command

010: All-Banks-Precharge Command

011: MRS (Mode Register Set) Command (Note)

100: CBR (CAS-before-RAS refresh) Command

101: NOP Command

110: Long ZQ Calibration Command

111: NOP Command

Besides this software based DDR initialization, this chip provides another hardware initialization mechanism. Please refer to Rx196[0] (RDMINIT) for more details.

dramc\_chk\_slfrf.src: self-refresh will be issued when RSFREFRK\*EN is asserted, it will be 8 read. 自刷新模式下，DDR4可以在没有外部时钟的条件下保留数据。

postsim:

1 /logic/kellyz/chkload\_new/rep\_load\_all –d 180104,180104 -t 00,09 -dir /logic/kellyz/chkload\_new/log

-d表示日期，-t表示时间

生成rep\_load180104\_180104\_00,09.log，如果memory列小于 20表示机器空闲，可以再丢。

2 Update命令顺序Get ECO1 grd env script：

/logic/mollyz/get\_chx002\_post\_sim\_eco\_new\_sdf\_grd grdx pegx GOLD mindelays

3 查看文件空间大小：

/logic/taowang/bin/report\_disk\_usage

或者df –h

查看机器大小：dfhl

WRITE\_FLUSH\_DRAMC和WRITE\_MERGE\_DRAMC这两个attributes都脱胎于之前的WRITE\_FLUSH\_DRAMC。

最大的变化就是将原来的LP变成了3个master的HP。

对于NB env 来讲，masters为：CPU， AZALIA 和GFX。

而对于DRAMC\_level env来讲，masters为：CPU，MCUTRF，GMINT。

同时为了更好的测试merge功能，增添了WRITE\_MERGE\_DRAMC这个attribute。

WRITE\_FLUSH\_DRAMC的主要功能是实现data sequence，即在相同address下的data能够从小到大依次传输，具体表现为下一笔data等于上一笔data加1。

其中每个master对应的高16bit data都有特殊的标志位，如下所示：

cpu\_data[31:0] = 'h0001\_0001;

azalia\_data[31:0] = 'h0010\_0001;

gfx\_data[31:0] = 'h0100\_0001.

另外，每个master对dramc write结束后，该attribute会随意产生占read/write command 1%概率的read将write data flush到DDR中。

WRITE\_MERGE\_DRAMC独立出来主要是为了更清晰地测试write data能够自动进行merge的功能。具体实现过程是随机产生相同address和随机address（各50%的概率），它们的write data分别对应为具有初始值和随机产生两种情况，然后调用task进行write操作。

其中read cycle的情况与WRITE\_FLUSH\_DRAMC类似，都是利用随机产生的1% read 命令将write data flush 到DDR中。

dramc\_chk\_auto\_precharge.src

MRS命令（cmd=0）

MR3 -> MR6(MAA=400), **A10**=1, tCCD\_L=5nCK

-> MR5(MAA=0), **A5=0,** ODT input buffer is active

-> MR1(MAA=101), **[A10:A8]**=001, RZQ/4,A0=1, DLL enable

-> MR0(MAA=31d), **[A13,A11:A9]**=0001, WR=12 and RTP=6 for auto precharge, **A8=1**, DLL reset yes, **[A12,A6:A4，A2]**=00011, CL =12,; A3=1, read burst type is interleave, **[A1:A0]**=01, BC4 or 8(on the fly).

ZQCL命令（cmd=6）A10 =1

ZQCS命令（cmd=6）A10 =0

ZQCL: 上电初始化后，用完成校准ZQ电阻。ZQCL会触发DRAM内部的校准引擎，

      一旦校准完成，校准后的值会传递到DRAM的IO管脚上，并反映为输出驱动和ODT阻值。

ZQCS: 周期性的校准，能够跟随电压和温度的变化而变化。校准需要更短的时间窗口，

      一次校准，可以有效的纠正最小0.5%的RON和RTT电阻。

MRS命令，cmd=0，MR4（MAA=0）,A12: write preamble. A11: read preamble.

ACT 命令，ACT\_n =0

PREA 命令cmd =2, A10 =1

REF命令 cmd =1

DES命令，csa=1

PDE/PDX命令，csa=1， and CKE have a change between 1 and 0

dramc\_chk\_slfrf.src

SRE/SRX命令， cmd=1，and CKE have a change between 1 and 0

/top/DRAMC\_TOP/ddr4\_udimm0/ddr4\_rank0/ddr4\_model\_0/get\_tRCD

DRAMC<->DDRIO interface,

Please check M1AD[127:0] M2AD[127:0] M1AD\_A?\_WRDY for write data and MCAI[127:0] MCA2I[127:0] for read data.

以DIO\_DDRPHY\_A0为例，代表DIMM上的一个DDR4 8bit颗粒。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Group | Basic IP name | Size  (before shrink) | Number in PHY | Note |
| Digital | DIO\_DX2DD |  | 8 | TX data |
| DIO\_DX2DS |  | 2 | TX DQS |
| DIO\_TXSDQ |  | 2 | TX data select |
| DIO\_DMOE |  | 2 | TX output enable |
| DIO\_DQSIEN |  | 2 | RX input enable |
| RXDIO\_SEL\_b\* |  | 8 | RX data |

1 DIO\_DX2DD\*:

The DIO\_DX2DD\* modules have 8 numbers totally in DIO\_DDRPHY\_A0, every module represents 1 bit. The top module is DIO\_DDRPHY\_A0.

The input data signals M1DL0, M1DH0, M1DL1, M1DH1 come from DDRIO\_TX\_BYTE\_PIPE\_A0, and output data signals MDL0, MDH0, MDL1, MDH1 will enter DDRPHYA\_A0/DX2DD/ modules.

And the data relation between DRAMC and DDRIP is following:

M1DL0<---------->M1AD[63:0]

M1DL1<---------->M2AD[63:0]

M1DH0<---------->M1AD[127:64]

M1DH1<---------->M2AD[127:64]

The input clock signals CLKI, CLKO and CLKO\_DDR represent 3 different clock domain, their frequencies are 800M, 1.6G and 1.6G.

And TX\_PHASE\_OUT (from DDRIO\_TX\_BYTE\_PIPE\_A0) can select different phase TX data.

2 DIO\_DX2DS:

The DIO\_DX2DS modules have 2 numbers totally in DIO\_DDRPHY\_A0, every module represents 1 bit. The top module is DIO\_DDRPHY\_A0.

The input DQS signals SDL, SDH comes from DDRIO\_TX\_BYTE\_PIPE\_A0 module.

And output DQS signals SDLXX\_PRE,SDHXX\_PRE will enter DDRPHYA\_A0/DX2DS\_X8/.

The input clock signals CLKI, CLKO and CLKO\_DDR represent 3 different clock domain, their frequencies are 800M, 1.6G and 1.6G.

3 DIO\_TXSDQ:

The DIO\_TXSDQ modules have 2 numbers totally in DIO\_DDRPHY\_A0, every module represents 1 bit. The top module is DIO\_DDRPHY\_A0.

The input WRDY comes from DDRIO\_TX\_BYTE\_PIPE\_A0 module.

The output WRDY\_IN will enter DDRPHYA\_A0/DX2DS\_X8/.

The input clock signals CLKI, CLKO and CLKO\_DDR represent 3 different clock domain, their frequencies are 800M, 1.6G and 1.6G.

4 DIO\_DMOE:

The DIO\_DMOE\_GEN modules have 2 numbers totally in DIO\_DDRPHY\_A0, every module represents 1 bit. The top module is DIO\_DDRPHY\_A0.

The input signals DQSOEDD, MDOEDD comes from DDRIO\_TX\_BYTE\_PIPE\_A0 module.

The output signals DQSOEDDX\_PRE, MDOEDDX will enter DDRPHYA\_A0/DMOE\_GEN/ module.

The input clock signals CLKI, CLKO and CLKO\_DDR represent 3 different clock domain, their frequencies are 800M, 1.6G and 1.6G.

5 DIO\_DQSIEN:

The DIO\_DQSIEN modules have 2 numbers totally in DIO\_DDRPHY\_A0, every module represents 1 bit. The top module is DIO\_DDRPHY\_A0.

The input signals DQSIN\_SEED0, DQSIN\_SEED2, LEADINGOE\_SEED, LEADINGOE\_COMP, DDOWPG\_TE\_SEED, DDOWPG2 comes from DDRIO\_TX\_BYTE\_PIPE\_A0 module.

The output signals DQSIEN\_HEAD0X, DQSIEN\_HEAD2X, LEADINGOE\_X, LEADINGOE\_ODT\_X, DDWPG2\_X will enter DDRPHYA\_A0/DMOE\_GEN/ module.

The input clock signals CLKI, CLKO and CLKO\_DDR represent 3 different clock domain, their frequencies are 800M, 1.6G and 1.6G.

6 RXDIO\_SEL\_b\*:

The RXDIO\_SEL\_b\* modules have 8 numbers totally in DIO\_DDRPHY\_A0, every module represents 1 bit. The top module is DIO\_DDRPHY\_A0.

The output data signals DRL1, DRH1, DFL1, DFH1 will enter DDRIO\_RX\_BYTE\_PIPE\_A0 module.

And the data relation between DRAMC and DDRIP is following:

DRL1<---------->M2AI[63:0]

DRH1<---------->M2A2I[63:0]

DFL1<---------->M2AI[127:64]

DFH1<---------->M2A2I[127:64]

And the input data signals LD0F\_b, LD1F\_b, LD2F\_b, LD3F\_b, LD4F\_b, LD5F\_b, LD6F\_b, LD7F\_b,

LD0R\_b, LD1R\_b, LD2R\_b, LD3R\_b, LD4R\_b, LD5R\_b, LD6R\_b, LD7R\_b come from DDRPHYA\_A0/RXDIO\_1 module.

The input CLKI signal is 800MHz, which is used to sample the input select signals SEL0 and SEL1 from DDRIO\_TX\_BYTE\_PIPE\_A0.

Christine give me 3 questions:

No.1

如何调整register的data delay？具体有哪些方法？

No.2

每个register的clock domain是什么？如何转换？

No.3

为什么要那么多MUX，原因是什么？

DCONV\_WADV\_NEW DNTMDL0Q (.Y(MDL0\_N),.A(MDL0X),.OS2(WOS2),.OS1(WOS1),.OS0(WOS0),.CLK(CLKO),.SCANEN(SCANEN),.ATPGEN(ATPGEN),.RDIMM(RDIMM), .RST\_(RST\_))

MDLOX--->MDLO\_N大概延迟了3个T

可以通过WOS来调整

module “DCONV\_WADV\_NEW”有7个相连的register，如下图所示：

assign SIDD[6] = A & (OS==3'b111);

assign SIDD[5] = SID[6] | A & (OS==3'b110);

assign SIDD[4] = SID[5] | A & (OS==3'b101);

assign SIDD[3] = SID[4] | A & (OS==3'b100);

assign SIDD[2] = SID[3] | A & (OS==3'b011);

assign SIDD[1] = SID[2] | A & (OS==3'b010);

assign SIDD[0] = SID[1] | A & (OS==3'b001);

assign Y = SID[0] ;

其中DIO\_DDRPHY中默认OS==3’b011，即TX data会延迟3T。通过调节OS可以使TX data的output 提前或延迟几T出现。

A是TX data MDL0/1X 或MDH0/1X的例化。

TX data：

For example: 2 cells, A\* represent someone cell, totally 8 number.

|  |  |
| --- | --- |
| Last | Next |
| M1AD[7:0] | M1DL0\_A0[7:0] |
| M2AD[7:0] | M1DL1\_A0[7:0] |
| M1AD[71:64] | M1DH0\_A0[7:0] |
| M2AD[71:64] | M1DH1\_A0[7:0] |
| M1AD[15:8] | M1DL0\_A1[7:0] |
| M2AD[15:8] | M1DL1\_A1[7:0] |
| M1AD[79:72] | M1DH0\_A1[7:0] |
| M2AD[79:72] | M1DH1\_A1[7:0] |

\* DDRIO\_CHX002

rfrfrf

dxxxxx

M1DL0-M1DH0-M1DL1-M1DH1

RX data：

For example: different A\* represent different cell.

|  |  |  |
| --- | --- | --- |
|  | Low bit | High bit |
| Raise  edge | DRL1\_A0\_b7~b0 -> M2AI[7:0] | DRH1\_A0\_b7~b0 -> M2A2I[7:0] |
| DRL1\_A1\_b7~b0 -> M2AI[15:8] | DRH1\_A0\_b7~b0 -> M2A2I[15:8] |
| … | … |
| DRL1\_A7\_b7~b0 -> M2AI[63:56] | DRH1\_A0\_b7~b0 -> M2A2I[63:56] |
| Fall  edge | DFL1\_A0\_b7~b0 -> M2AI[71:64] | DFH1\_A0\_b7~b0 -> M2A2I[71:64] |
| DFL1\_A1\_b7~b0 -> M2AI[79:72] | DFH1\_A1\_b7~b0 -> M2A2I[79:72] |
| … | … |
| DFL1\_A7\_b7~b0 -> M2AI[127:120] | DFH1\_A7\_b7~b0 -> M2A2I[127:120] |

\* DDRIO\_CHX002

rfrfr

xxxxx

DRL1-DFL1-DRH1-DFH1

Clock domain:



以上是data的clock，还有command的clock，也是通过PLLINDDR 1.6GM的时钟经过VCDL转换的，即DIMM clock（DQS），这与RXDIO\_SEL中的内部clock无关，所以SEL0/1信号需要MUX来选择256bit data到fifo中，通过00，01，10，11四个pointer来分别对应4个256bit。然后通过CLK（来自PLLIN）来sync。因为SEL0/1信号本来就是800Mdomain（来自PLLINDDR，但CLKI和CLK对齐）

Summary：

\*DIMM clock MDQSAOP/N 与内部clock无关。

\*

wrt( DDR4) a=00014488e0 d=1356a2dd b=0000 <- 00014488e0

wrt( DDR4) a=00014488e4 d=3744b7f8 b=0000 <- 00014488e4

wrt( DDR4) a=00014488e8 d=3764a4c1 b=0000 <- 00014488e8

wrt( DDR4) a=00014488ec d=1752a6c3 b=0000 <- 00014488ec

wrt( DDR4) a=00014488f0 d=3a7d95d4 b=0000 <- 00014488f0

wrt( DDR4) a=00014488f4 d=214ab8ef b=0000 <- 00014488f4

wrt( DDR4) a=00014488f8 d=244982e8 b=0000 <- 00014488f8

wrt( DDR4) a=00014488fc d=117a8cf2 b=0000 <- 00014488fc

irun.log中的data**都是DW（32bit）对齐**，data地址的后两位以00，04，08，0c，10，14，18，1c…的方式增长。其中，00代表着00，01，10，11，每个地址对应8bit，总共32bit，因此00会对应一个DW data。

**对于length来讲**，DRAM中有1QW，2QW，4QW，8QW的burst, 它们用来表示1个burst下传输的data总量。它们的address wrap boundary的划分如下：

1QW：（=8byte）表示2-beat burst，因此address wrap boundary是00~04;

2QW：（=16byte）表示4-beat burst，因此address wrap boundary是00~0c;

4QW：（=32byte）表示8-beat burst，因此address wrap boundary是00~1c;

8QW：（=64byte）表示16-beat burst，因此address wrap boundary是00~3c;

如果是以38结尾的地址开始，进行2QW length的wrap burst，那么地址顺序为：

38，3c，30，34，48，4c，40，44…(Highlight表示该burst)

如果是以38结尾的地址开始，进行4QW length的wrap burst，那么地址顺序为：

38，3c，20，24，28，2c，30，34，58…(Highlight表示该burst)

注：每个wrap burst都是从对应的地址开始执行。

**同步FIFO空满标志位的产生：**

flag\_gen模块产生FIFO空满标志。本模块设计并不用读写地址判定FIFO是否空满。设计一个计数器，该计数器(pt\_cnt)用于指示当前周期中FIFO中数据的个数。由于FIFO中最多只有16个数据，因此采用5位计数器来指示FIFO中数据个数。具体计算如下：

1)复位的时候，count=0；

2)如果w\_en和r\_en同时有效的时候，count不加也不减；表示同时对FIFO进行读写操作的时候，FIFO中的数据个数不变。

3)如果w\_en有效且full=0，则count+1;表示写操作且FIFO未满时候，FIFO中的数据个数增加了1；

4)如果r\_en有效且empty=0，则count-1; 表示读操作且FIFO未满时候，FIFO中的数据个数减少了1；

5)如果count=0的时候，表示FIFO空，需要设置empty=1;如果count=16的时候，表示FIFO现在已经满，需要设置full=1。

**异步FIFO空满标志位的产生：**

**1.读写指针的工作原理**

　　写指针：总是指向下一个将要被写入的单元，复位时，指向第1个单元(编号为0)。

　　读指针：总是指向当前要被读出的数据，复位时，指向第1个单元(编号为0).

**2.FIFO的“空”/“满”检测**

　　FIFO设计的关键：产生可靠的FIFO读写指针和生成FIFO“空”/“满”状态标志。

　　当读写指针相等时，表明FIFO为空，这种情况发生在复位操作时，或者当读指针读出FIFO中最后一个字后，追赶上了写指针。

当读写指针再次相等时，表明FIFO为满，这种情况发生在，当写指针转了一圈，折回来(wrapped around)又追上了读指针

为了区分到底是满状态还是空状态，可以采用以下方法：

　　方法1：在指针中添加一个额外的位(extra bit)，当写指针增加并越过最后一个FIFO地址时，就将写指针这个未用的MSB加1，其它位回零。对读指针也进行同样的操作。此时，对于深度为2n的FIFO，需要的读/写指针位宽为(n+1)位，如对于深度为8的FIFO，需要采用4bit的计数器，0000～1000、1001～1111，MSB作为折回标志位，而低3位作为地址指针。

* 如果两个指针的MSB不同，说明写指针比读指针多折回了一次；如r\_addr=0000,而w\_addr = 1000,为满。
* 如果两个指针的MSB相同，则说明两个指针折回的次数相等。其余位相等，说明FIFO为空；

**3.  二进制FIFO指针的考虑**

　　将一个二进制的计数值从一个时钟域同步到另一个时钟域的时候很容易出现问题，因为采用二进制计数器时所有位都可能同时变化，在同一个时钟沿同步多个信号的变化会产生亚稳态问题。而使用格雷码只有一位变化，因此在两个时钟域间同步多个位不会产生问题。所以需要一个二进制到gray码的转换电路，将地址值转换为相应的gray码，然后将该gray码同步到另一个时钟域进行对比，作为空满状态的检测。

     使用gray码解决了一个问题，但同时也带来另一个问题，即在格雷码域如何判断空与满。

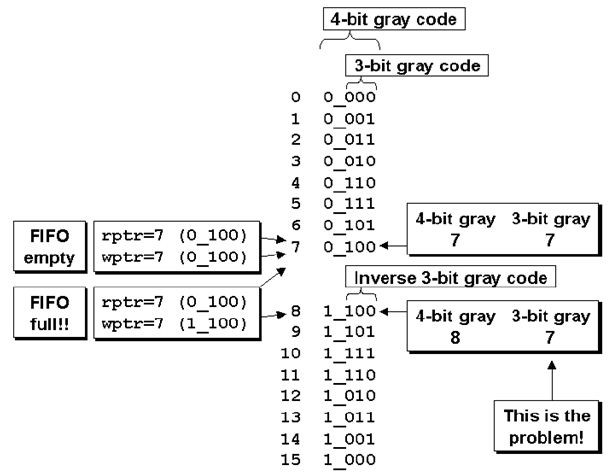
　　对于“空”的判断依然依据二者完全相等(包括MSB)；

而对于“满”的判断，如下图，由于gray码除了MSB外，具有镜像对称的特点，当读指针指向7，写指针指向8时，除了MSB，其余位皆相同，不能说它为满。因此不能单纯的只检测最高位了，在gray码上判断为满必须同时满足以下3条：

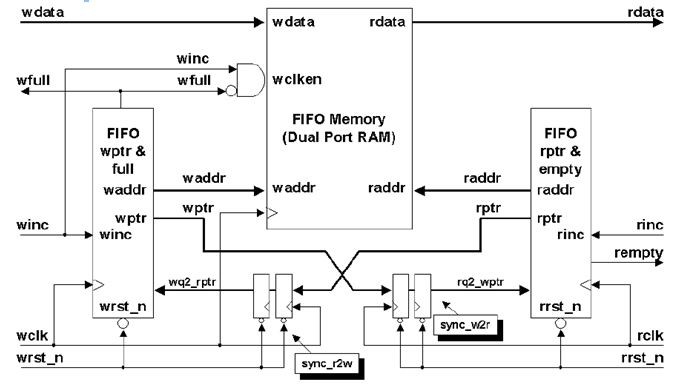
wptr和同步过来的rptr的MSB不相等，因为wptr必须比rptr多折回一次。

wptr与rptr的次高位不相等，如上图位置7和位置15，转化为二进制对应的是0111和1111，MSB不同说明多折回一次，111相同代表同一位置。

剩下的其余位完全相等。



0，1，3，2，6，7，5，4，12（c），13（d），15（f），14（e），10（a），11(b)，9，8



计算机的字长：32位或64位，计算机在同一时间内处理的一组二进制数称为一个计算机的“字”，而这组二进制数的位数就是“字长”。

画图工具：

[‎2018/‎5/‎23 17:39] Nuoxuan Ju(BJ-RD):

https://www.processon.com/diagrams

焦洋：

downstream 、 upstream，跟过不过PXP没关系，不是这么来看的

从CPU或MXU打下来的访问device的就是down（都要过PXPTRF）  
从device打出来的去CPU或者去DRAM，就是up跟PXP没关系