# dep\_resource\_conflict\_check类声明

|  |
| --- |
| //相关资源冲突检查  class dep\_resource\_conflict\_check **:** public Component**{**  public**:**  /\*  输入参数：  l\_ip = configure\_interface  coredynp = dyn\_p\_  compare\_bits = compare\_bits\_  is\_default = \_is\_default  \*/  dep\_resource\_conflict\_check**(**const InputParameter **\***configure\_interface**,**  const CoreDynParam **&** dyn\_p\_**,** int compare\_bits\_**,** bool \_is\_default**=true);**    //传递参数使用，包含部分cacti的参数  InputParameter l\_ip**;**  //没有用到似乎  uca\_org\_t local\_result**;**    double WNORn**,** WNORp**,** Wevalinvp**,** Wevalinvn**,** Wcompn**,** Wcompp**,** Wcomppreequ**;**  //core的动态参数  //用到了coredynp.decodeW， coredynp.core\_ty  CoreDynParam coredynp**;**  //每个比较器的比较位数  //传入时只需要传递物理寄存器的索引位数/ROB的索引位数  int compare\_bits**;**  bool is\_default**;**  statsDef tdp\_stats**;**  statsDef rtp\_stats**;**  statsDef stats\_t**;**  powerDef power\_t**;**  //计算检查资源冲突的功耗  void conflict\_check\_power**();**  //估计一个比较器的功耗  double compare\_cap**();**  **~**dep\_resource\_conflict\_check**(){**  local\_result**.**cleanup**();**  **}**  //没有被使用  void leakage\_feedback**(**double temperature**);**  **};** |

# 定义dep\_resource\_conflict\_check的对象

* 该类主要用于计算检查指令之间的资源冲突所带来的逻辑电路的功耗
* 在RENAMING类中定义该类的对象，包括两个idcl和fdcl
* 在乱序核心中，根据调度策略不同（PRF，RS）输入参数将会不同，但是实例化的参数是一样的，形式也是一样的。但是phy\_ireg\_width的值不同

|  |
| --- |
| //coredynp.phy\_ireg\_width =  // int(ceil(log2(XML->sys.core[ithCore].phy\_Regs\_IRF\_size)));  //物理寄存器的索引位宽（bits）  //coredynp.phy\_ireg\_width =  // int(ceil(log2(XML->sys.core[ithCore].ROB\_size)));  //索引ROB的位宽（bits）  idcl **=** **new** dep\_resource\_conflict\_check**(&**interface\_ip**,**coredynp**,**coredynp**.**phy\_ireg\_width**);**  fdcl **=** **new** dep\_resource\_conflict\_check**(&**interface\_ip**,**coredynp**,**coredynp**.**phy\_freg\_width**);** |

* 另一种定义情况是按序处理器，但是发射宽度（译码宽度）要大于1，这个时候需要做相关检查，定义方式和之前一样

# dep\_resource\_conflict\_check的构造函数

|  |
| --- |
| dep\_resource\_conflict\_check**::**dep\_resource\_conflict\_check**(**  const InputParameter **\***configure\_interface**,**const CoreDynParam **&** dyn\_p\_**,**  int compare\_bits\_**,**bool \_is\_default**)**  **:** l\_ip**(\***configure\_interface**),**coredynp**(**dyn\_p\_**),**  compare\_bits**(**compare\_bits\_**),**is\_default**(**\_is\_default**)**  **{**  Wcompn **=** 25 **\*** l\_ip**.**F\_sz\_um**;**  Wevalinvp **=** 25 **\*** l\_ip**.**F\_sz\_um**;**  Wevalinvn **=** 100 **\*** l\_ip**.**F\_sz\_um**;**  Wcomppreequ **=** 50 **\*** l\_ip**.**F\_sz\_um**;**  WNORn **=** 6.75 **\*** l\_ip**.**F\_sz\_um**;**  WNORp **=** 38.125 **\*** l\_ip**.**F\_sz\_um**;**    //好像也没有使用到  local\_result **=** init\_interface**(&**l\_ip**);**    //compare\_bits比较器的位数  //并没有区分按序或者乱序  //第一个16代码操作数的位数  //第二个8代表共享资源的索引位数  //第三个8代表  **if** **(**coredynp**.**core\_ty**==**Inorder**)**  compare\_bits **+=** 16 **+** 8 **+** 8**;**  **else**  compare\_bits **+=** 16 **+** 8 **+** 8**;**    //计算冲突检测的功耗  conflict\_check\_power**();**  double sckRation **=** g\_tp**.**sckt\_co\_eff**;**  power**.**readOp**.**dynamic **\*=** sckRation**;**  power**.**writeOp**.**dynamic **\*=** sckRation**;**  power**.**searchOp**.**dynamic **\*=** sckRation**;**  **}** |

# conflict\_check\_power函数

|  |
| --- |
| void dep\_resource\_conflict\_check**::**conflict\_check\_power**()**  **{**  double Ctotal**;**  //比较器个数  int num\_comparators**;**  //译码的n条指令，其中每一条需要和其它n-1条进行比较，n=decodeW  //比较包括2个源操作之间的比较和1个目的操作数之间的比较  //当译码宽度为1时，不需要dcl逻辑  num\_comparators **=** 3**\*((**coredynp**.**decodeW**)** **\*** **(**coredynp**.**decodeW**)-**coredynp**.**decodeW**);**  //估计得到每个比较器的功耗\*个数  Ctotal **=** num\_comparators **\*** compare\_cap**();**  power**.**readOp**.**dynamic**=**Ctotal**\***g\_tp**.**peri\_global**.**Vdd**\***g\_tp**.**peri\_global**.**Vdd**;**  power**.**readOp**.**leakage**=**  num\_comparators**\***compare\_bits**\***2**\***simplified\_nmos\_leakage**(**Wcompn**,** **false);**  double long\_channel\_device\_reduction **=**  longer\_channel\_device\_reduction**(**Core\_device**,** coredynp**.**core\_ty**);**  power**.**readOp**.**longer\_channel\_leakage **=**  power**.**readOp**.**leakage**\***long\_channel\_device\_reduction**;**  power**.**readOp**.**gate\_leakage**=**  num\_comparators**\***compare\_bits**\***2**\***cmos\_Ig\_leakage**(**Wcompn**,** 0**,** 2**,** nmos**);**  double pg\_reduction **=** power\_gating\_leakage\_reduction**(false);**  power**.**readOp**.**power\_gated\_leakage **=** power**.**readOp**.**leakage**\***pg\_reduction**;**  power**.**readOp**.**power\_gated\_with\_long\_channel\_leakage **=**  power**.**readOp**.**power\_gated\_leakage **\*** long\_channel\_device\_reduction**;**  **}** |