目录

[1. RENAMING类声明 2](#_Toc532913896)

[2. RENAMING的构造函数 3](#_Toc532913897)

[3. computerEnergy函数 4](#_Toc532913898)

[4. DisplayEnergy函数 7](#_Toc532913899)

# RENAMING类声明

* 该类是Core中的重命名单元的实现。类中定义7个ArrayST对象，其中RAHT虽然有声明，但是没有使用，因此没有考虑。同时还定义了针对于浮点和定点的资源冲突检测逻辑dep\_resource\_conflict\_check

|  |  |  |
| --- | --- | --- |
| RENAMING | iFRAT(ArrayST)  fFRAT(ArrayST)  ifreeL(ArrayST)  ffreeL(ArrayST) | iRRAT(ArrayST)  FRRAT(ArrayST)  fdcl(dep\_resource\_conflict\_check)  idcl(dep\_resource\_conflict\_check) |

* 尽管重命名逻辑一般只在乱序处理器中出现，但是也会在按序处理器中出现，例如Itainium，一个VLIW处理器。但是McPAT中没有考虑按序处理器包含重命名逻辑
* FRAT(Front RAT), RRAT(Retire RAT)。每一条指令都会读两次FRAT和写一次FRAT。当指令提交时，每条指令会写一次RRAT，当上下文切换时，所有RRAT都会被读一次
* 如果处理器支持硬件多线程，则需要为每一个线程复制一份重命名逻辑
* 对于基于PRF的乱序处理器而言，需要支持双RAT结构，即FRAT+RRAT，对于基于RS的乱序处理器，则只需要使用FRAT即可。（之后有解释）
* dep\_resource\_conflict\_check只需要考虑处理器的调度策略是什么，PRF/RS，其它没有关系。同时检查逻辑并没有考虑面积开销
* 在乱序的RS的情况下，freelist只有ifreeL一个，用于代表一个整体的freelist
* 声明代码

|  |
| --- |
| class RENAMINGU **:**public Component **{**  public**:**  ParseXML **\***XML**;**  int ithCore**;**  InputParameter interface\_ip**;**  double clockRate**,**executionTime**;**  CoreDynParam coredynp**;**    //int/float Front RAT  ArrayST **\*** iFRAT**;**  ArrayST **\*** fFRAT**;**  //Int float RetireRAT  ArrayST **\*** iRRAT**;**  ArrayST **\*** fRRAT**;**    //int/float free list  ArrayST **\*** ifreeL**;**  ArrayST **\*** ffreeL**;**    //相关资源冲突检查逻辑  //logic.h 72  dep\_resource\_conflict\_check **\*** idcl**;**  dep\_resource\_conflict\_check **\*** fdcl**;**    //没有被使用  ArrayST **\*** RAHT**;**//register alias history table Used to store GC  bool exist**;**  RENAMINGU**(**ParseXML **\***XML\_interface**,** int ithCore\_**,**  InputParameter**\*** interface\_ip\_**,**  const CoreDynParam **&** dyn\_p\_**,** bool exist\_**=true);**  void computeEnergy**(**bool is\_tdp**=true);**  void displayEnergy**(**uint32\_t indent **=** 0**,**int plevel **=** 100**,** bool is\_tdp**=true);**  **~**RENAMINGU**();**  **};** |

# RENAMING中的FRAT，RRAT，RAHT解释

* RAHT是检查点的硬件结构，用于减少分支预测错误所带来的影响（根据alpha 21264设计）
* 基于物理寄存器堆的设计中，物理寄存器文件中包括了推测的数据和非推测的数据。对于非推测式的数据必须保证在线程切换或者中断的时候能够被正确保存下来
* 因此Hinton提出了双RAT的设计方案，McPAT就是基于这样的设计，类似的结构有（Intel Netburst, Sandy Bridge）
* FRAT(frontend RAT) 用于正常情况下的重命名
* RRAT(retired RAT)则是存储着当前体系结构寄存器的映射。RRAT只会在提交阶段被存取，不会影响FRAT
* FRAT可以是CAM结构或者是RAM结构，但是RRAT只是RAM结构。并且RRAT的写端口数量和提交宽度相同

# RENAMING的构造函数

* 初始化参数，创建对象，计算各个对象的面积，并且得到整体的面积参数

|  |
| --- |
| RENAMINGU**::**RENAMINGU**(**ParseXML**\*** XML\_interface**,** int ithCore\_**,**  InputParameter**\*** interface\_ip\_**,**  const CoreDynParam **&** dyn\_p\_**,**bool exist\_**)**  **:**XML**(**XML\_interface**),**ithCore**(**ithCore\_**),**interface\_ip**(\***interface\_ip\_**),**  coredynp**(**dyn\_p\_**),**iFRAT**(**0**),**fFRAT**(**0**),**  iRRAT**(**0**),**fRRAT**(**0**),**  ifreeL**(**0**),**ffreeL**(**0**),**  idcl**(**0**),**fdcl**(**0**),**  RAHT**(**0**),**exist**(**exist\_**)**  **{**  **if** **(!**exist**)** **return;**    //乱序处理器的情况，继续细分为PRF/RS，然后在细分重命名表的结构设计 RAM/CAM  **if** **(**coredynp**.**core\_ty**==**OOO**)**  **{**  **if** **(**coredynp**.**scheu\_ty**==**PhysicalRegFile**)**  **{**  **if** **(**coredynp**.**rm\_ty **==**RAMbased**)**//rename\_scheme  **{**  iFRAT **=** **new** ArrayST**(&**interface\_ip**,** "Int FrontRAT"**);**  //由于多线程的情况是直接复制多份，因此需要乘以倍数  //area=iFRAT->local\_result.area\* number\_hardware\_threads  fFRAT **=** **new** ArrayST**(&**interface\_ip**,** "FP FrontRAT"**);**  **}**  **else** **if** **((**coredynp**.**rm\_ty **==**CAMbased**))**  **{**  //iFRAT，fFRAT  **}**  //iRRAT，fRRAT，ifreeL，ffreeL  iRRAT **=** **new** ArrayST**(&**interface\_ip**,** "Int RetireRAT"**);**  //area=iRRAT->local\_result.area\* number\_hardware\_threads)    //资源检查逻辑，没有考虑面积问题  idcl **=** **new** dep\_resource\_conflict\_check**();**  fdcl **=** **new** dep\_resource\_conflict\_check**();**  **}**  **else** **if** **(**coredynp**.**scheu\_ty**==**ReservationStation**)**  **{**  **if** **(**coredynp**.**rm\_ty **==**RAMbased**){**  iFRAT **=** **new** ArrayST**(&**interface\_ip**,** "Int FrontRAT"**);**  //mcpat的解释是弥补GC带来的影响  iFRAT**->**local\_result**.**adjust\_area**();**  iFRAT**->**local\_result**.**power**.**readOp**.**dynamic **\*=**1.5**;**  iFRAT**->**local\_result**.**power**.**writeOp**.**dynamic **\*=**1.5**;**  //area=iFRAT->local\_result.area\* number\_hardware\_threads  fFRAT **=** **new** ArrayST**(&**interface\_ip**,** "FP FrontRAT"**);**  fFRAT**->**local\_result**.**adjust\_area**();**  fFRAT**->**local\_result**.**power**.**readOp**.**dynamic **\*=**1.5**;**  fFRAT**->**local\_result**.**power**.**writeOp**.**dynamic **\*=**1.5**;**  //area=fFRAT->local\_result.area\* number\_hardware\_threads  **}**  **else** **if** **((**coredynp**.**rm\_ty **==**CAMbased**))**  **{**  //iFRAT,fFRAT  **}**  //此时只有一个整体的free list  ifreeL **=** **new** ArrayST**(&**interface\_ip**,** "Unified Free List"**);**  //area  idcl **=** **new** dep\_resource\_conflict\_check**();**  fdcl **=** **new** dep\_resource\_conflict\_check**);**  **}**  **}**  **if** **(**coredynp**.**core\_ty**==**Inorder**&&** coredynp**.**issueW**>**1**)**  **{**  //按序多发射的情况下，也需要检查发射指令之间的资源冲突问题  idcl **=** **new** dep\_resource\_conflict\_check**();**  fdcl **=** **new** dep\_resource\_conflict\_check**();**  **}**  **}** |

# computerEnergy函数

* ComputerEnergy函数包括一个参数is\_tdp，bool类型
* 当is\_tdp为true时，power = energy\_per\_cycle\* clock\_rate。在该函数中只计算得到每个周期该组件会消耗的能量energy\_per\_cycle，在displayEnergy函数中，将会使用该公式计算得到峰值power，即每个周期都在工作时的功率。此时的计算结果保存在power中
* 当is\_tdp为false时，power = total energy / Total RENAMINGtion time。同样该函数中只计算该组件在整个执行过程中会消耗的所有能量（使用组件的访问次数等计算），在displayEnergy函数中，将其除以整体的执行时间（cycle count / clock rate），得到运行时的动态功耗。此时的计算结果保存在rt\_power中
* 初始化组件的状态信息
* 初始化状态之后，需要计算能耗(操作个数\*操作消耗的能量)。
* 最后计算泄露功耗(需要考虑线程数)，将结果保存在RENAMING的power/rt\_power中

# DisplayEnergy函数

* 显示RENAMING的详细结果，包括面积，峰值功耗，漏电功耗和动态功耗

|  |
| --- |
| void RENAMINGU**::**displayEnergy**(**uint32\_t indent**,**int plevel**,**bool is\_tdp**)**  **{**  **if** **(!**exist**)** **return;**  **if** **(**is\_tdp**)**  **{**  **if** **(**coredynp**.**core\_ty**==**OOO**)**  **{**  /\*  Int Front End RAT:iFRAT  FP Front End RAT:fFRAT  Free List:ifreeL    Area = iFRAT->area.get\_area()\*1e-6  Peak Dynamic = iFRAT->power.readOp.dynamic\*clockRate  Subthreshold Leakage = (long\_channel?  iFRAT->power.readOp.longer\_channel\_leakage  :iFRAT->power.readOp.leakage)  Subthreshold Leakage with power gating = (long\_channel?  iFRAT->power.readOp.power\_gated\_with\_long\_channel\_leakage :  iFRAT->power.readOp.power\_gated\_leakage)  Runtime Dynamic = iFRAT->rt\_power.readOp.dynamic/executionTime  \*/  **if** **(**coredynp**.**scheu\_ty**==**PhysicalRegFile**)**  **{**  /\*  Int Retire RAT:iRRAT  FP Retire RAT:fRRAT  FP Free List:ffreeL  \*/  **}**  **}**  **else**  **{**  /\*  Int DCL:idcl  FP DCL:fdcl  \*/  **}**  **}**  **else**  **{**  **}**  **}** |