目录

[1. LoadStoreU类声明 2](#_Toc532847446)

[2. LoadStoreU的构造函数 3](#_Toc532847447)

[3. computerEnergy函数 4](#_Toc532847448)

[4. DisplayEnergy函数 7](#_Toc532847449)

# LoadStoreU类声明

* 该类主要定义了在执行阶段将要使用的部件，包括dcache，LSQ和LoadQ。Dcache对象中继续定义了caches，ifb，missb，prefetchb，wbb，这些对象都是ArrayST类的对象
* LSQ在按序或者乱序处理器中都包括，但是前者中LSQ只作为StoreQ使用
* LoadQ只在乱序处理器中有使用，此时LSQ只作为StoreQ使用

|  |
| --- |
| //Load store单元  class LoadStoreU **:**public Component **{**  public**:**  //获取参数，一样的参数  ParseXML **\***XML**;**  //核标号  int ithCore**;**  InputParameter interface\_ip**;**  CoreDynParam coredynp**;**  enum Cache\_policy cache\_p**;**  double clockRate**,**executionTime**;**  double scktRatio**,** chip\_PR\_overhead**,** macro\_PR\_overhead**;**    //高度，目前猜测是线程数  double lsq\_height**;**  //L1 Dcache,array.h 90  DataCache dcache**;**    //它实际上是存储队列，但是对于inorder处理器，它同时充当loadQ和StoreQ  ArrayST **\*** LSQ**;**  //loadQ  ArrayST **\*** LoadQ**;**  //判断是否存在  bool exist**;**  //构造函数  LoadStoreU**(**ParseXML **\***XML\_interface**,** int ithCore\_**,**  InputParameter**\*** interface\_ip\_**,**  const CoreDynParam **&** dyn\_p\_**,** bool exist\_**=true);**  //计算功耗，峰值功耗和动态功耗  void computeEnergy**(**bool is\_tdp**=true);**  //显示具体信息  void displayEnergy**(**uint32\_t indent **=** 0**,**int plevel **=** 100**,** bool is\_tdp**=true);**  **~**LoadStoreU**();**  **};** |

# LoadStoreU的构造函数

* 构造函数主要定义了类中的各个对象，以及dcache中的对象，然后计算每个对象的面积，累加得到该单元的面积
* 每个对象的面积几乎都是在该类的构造函数中完成，此时调用local\_result.area即可获取该对象的面积
* 每个对象都有一个参数area，用于临时记录该对象的面积信息。Area提供了两个方法area.set\_area()和area.get\_area()，可以方便的记录和获取面积信息
* 细节问题
  + 当cache的替换策略为写回时(cache\_p==Write\_bcak)，dcache会定义wbb对象(write\_back buffer)
  + 当Core的类型为乱序核，并且xml提供了LoadQ的信息，则LoadQ对象也会被定义((coredynp.core\_ty==OOO) && (load\_buffer\_size >0))
* 具体的代码

|  |
| --- |
| LoadStoreU**::**LoadStoreU**(**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\_**),**LSQ**(**0**),**LoadQ**(**0**),**exist**(**exist\_**)**  **{**  **if** **(!**exist**)** **return;**  //定义dcache的caches对象，并获取面积信息，将其累加到Dcache的area和LoadstoreU的area中  dcache**.**caches **=** **new** ArrayST**(&**interface\_ip**,** "dcache"**,** Core\_device**,**  coredynp**.**opt\_local**,** coredynp**.**core\_ty**);**  dcache**.**area**.**set\_area**(**dcache**.**area**.**get\_area**()** **+**  dcache**.**caches**->**local\_result**.**area**);**  area**.**set\_area**(**area**.**get\_area**()+** dcache**.**caches**->**local\_result**.**area**);**  //定义dcache的missb, ifb, prefetchb  //...  //如果替换策略为WB，则定义dcache的wbb  **if** **(**cache\_p**==**Write\_back**){}**  //定义LSQ对象，也是arrayST对象，并获取面积信息，将其累加到LSQ的area和LoadstoreU的area中  //计算lsq的高度，当计算bypass网络的时候需要用到  //每个arraysT对象都会自动计算  lsq\_height**=**LSQ**->**local\_result**.**cache\_ht**\***sqrt**(**cdb\_overhead**);**  //如果为乱序核，并且给出了LoadQ的信息，则定义LoadQ，也是arrayST对象  **if** **((**coredynp**.**core\_ty**==**OOO**)** **&&** **(**XML**->**sys**.**core**[**ithCore**].**load\_buffer\_size **>**0**))**  **{**  //重新计算，需要包括两者，因为此时LSQ只作为StoreQ使用  lsq\_height**=(**LSQ**->**local\_result**.**cache\_ht **+**  LoadQ**->**local\_result**.**cache\_ht**)\***sqrt**(**cdb\_overhead**);**  **}**  //计算整体的面积(cdb\_overhead固定为1.1，未找到解释)  area**.**set\_area**(**area**.**get\_area**()\***cdb\_overhead**);**  **}** |

# 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 execution 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
* 初始化计算energy\_per\_cycle的参数，保存在stats\_t中。这些状态基本上表示一个周期内部件接收到的某些操作的次数，例如访问次数，写入次数等。LSU\_duty\_cycle代表占空比，即每个周期内部件工作的时间比例

|  |
| --- |
| //初始化Dcache的caches的参数  //每周期的访问次数  dcache**.**caches**->**stats\_t**.**readAc**.**access **=**  0.67**\***dcache**.**caches**->**l\_ip**.**num\_rw\_ports**\***coredynp**.**LSU\_duty\_cycle**;**  //每周期读取miss次数  dcache**.**caches**->**stats\_t**.**readAc**.**miss **=** 0**;**  //每周期读取hit次数  dcache**.**caches**->**stats\_t**.**readAc**.**hit **=**  dcache**.**caches**->**stats\_t**.**readAc**.**access **-** dcache**.**caches**->**stats\_t**.**readAc**.**miss**;**  //每周期写入的次数  dcache**.**caches**->**stats\_t**.**writeAc**.**access **=**  0.33**\***dcache**.**caches**->**l\_ip**.**num\_rw\_ports**\***coredynp**.**LSU\_duty\_cycle**;**  //每周期写入的miss次数  dcache**.**caches**->**stats\_t**.**writeAc**.**miss **=** 0**;**  //每周期写入的hit次数  dcache**.**caches**->**stats\_t**.**writeAc**.**hit **=**  dcache**.**caches**->**stats\_t**.**writeAc**.**access **-** dcache**.**caches**->**stats\_t**.**writeAc**.**miss**;**  //初始化dcache的missb状态  dcache**.**missb**->**stats\_t**.**readAc**.**access **=**  dcache**.**missb**->**l\_ip**.**num\_search\_ports**\***coredynp**.**LSU\_duty\_cycle**;**  dcache**.**missb**->**stats\_t**.**writeAc**.**access **=**  dcache**.**missb**->**l\_ip**.**num\_search\_ports**\***coredynp**.**LSU\_duty\_cycle**;**  //初始化ifb的状态  dcache**.**ifb**->**stats\_t**.**readAc**.**access **=**  dcache**.**ifb**->**l\_ip**.**num\_search\_ports**\***coredynp**.**LSU\_duty\_cycle**;**  dcache**.**ifb**->**stats\_t**.**writeAc**.**access **=**  dcache**.**ifb**->**l\_ip**.**num\_search\_ports**\***coredynp**.**LSU\_duty\_cycle**;**  //初始化prefetchb的状态  dcache**.**prefetchb**->**stats\_t**.**readAc**.**access **=**  dcache**.**prefetchb**->**l\_ip**.**num\_search\_ports**\***coredynp**.**LSU\_duty\_cycle**;**  dcache**.**prefetchb**->**stats\_t**.**writeAc**.**access **=**  dcache**.**ifb**->**l\_ip**.**num\_search\_ports**\***coredynp**.**LSU\_duty\_cycle**;**  //如果包含wbb，则初始化其状态  **if** **(**cache\_p**==**Write\_back**)**  **{**  dcache**.**wbb**->**stats\_t**.**readAc**.**access **=** dcache**.**wbb**->**l\_ip**.**num\_search\_ports**;**  dcache**.**wbb**->**stats\_t**.**writeAc**.**access **=** dcache**.**wbb**->**l\_ip**.**num\_search\_ports**;**  **}**  //初始化LSQ的状态  LSQ**->**stats\_t**.**readAc**.**access **=** LSQ**->**stats\_t**.**writeAc**.**access **=**  LSQ**->**l\_ip**.**num\_search\_ports**\***coredynp**.**LSU\_duty\_cycle**;**  **if** **(**如果LoadQ存在**)**  **{**  LoadQ**->**stats\_t**.**readAc**.**access **=** LoadQ**->**stats\_t**.**writeAc**.**access **=**  LoadQ**->**l\_ip**.**num\_search\_ports**\***coredynp**.**LSU\_duty\_cycle**;**  **}** |

* 初始化计算total\_energy的参数，保存在stats\_t中，主要记录的是整个运行过程中，组件的一些操作次数。具体可见代码
* 计算每个组件的能耗，并保存起来（per cycle/total的计算方式相同）。计算的方式：操作的次数(总的/每周期的)\*操作的次数。（以dcache的caches举例）

|  |
| --- |
| //根据操作次数和每个操作的能耗，计算caches的energy\_per\_cycle/total\_energy  //并将结果累加到dcache的临时变量中  dcache**.**power\_t**.**readOp**.**dynamic **+=** **(**  dcache**.**caches**->**stats\_t**.**readAc**.**hit**\***  dcache**.**caches**->**local\_result**.**power**.**readOp**.**dynamic**+**  dcache**.**caches**->**stats\_t**.**readAc**.**miss**\***  dcache**.**caches**->**local\_result**.**power**.**readOp**.**dynamic**+**  dcache**.**caches**->**stats\_t**.**writeAc**.**miss**\***  dcache**.**caches**->**local\_result**.**tag\_array2**->**power**.**readOp**.**dynamic**+**  dcache**.**caches**->**stats\_t**.**writeAc**.**access**\***  dcache**.**caches**->**local\_result**.**power**.**writeOp**.**dynamic  **);**  **if** **(**cache\_p**==**Write\_back**)** //如果是写回策略，则写入miss也会带来功耗  dcache**.**power\_t**.**readOp**.**dynamic **+=**  dcache**.**caches**->**stats\_t**.**writeAc**.**miss**\***  dcache**.**caches**->**local\_result**.**power**.**writeOp**.**dynamic**;** |

* 能耗已经计算完成，然后将所有组件的其它类型的功耗累加在起来(leakage和gate\_leakage)，最终将所有部件的power/rt\_power参数累加，放到LoadStoreU的power/rt\_power中(Component自带的属性)。当is\_tdp为true时，结果放入power中，否则放入rt\_power中

|  |
| --- |
| **if** **(**is\_tdp**)**  **{**  //const double pppm\_lkg[4] = {0,1,1,0}  //累加power中的第二个参数和第三个参数，leakage和gate\_leakage  dcache**.**power **=** dcache**.**power\_t **+** **(**dcache**.**caches**->**local\_result**.**power **+**  dcache**.**missb**->**local\_result**.**power **+**  dcache**.**ifb**->**local\_result**.**power **+**  dcache**.**prefetchb**->**local\_result**.**power**)** **\***pppm\_lkg**;**    **if** **(**cache\_p**==**Write\_back**)**  **{**  dcache**.**power **=** dcache**.**power **+** dcache**.**wbb**->**local\_result**.**power**\***pppm\_lkg**;**  **}**  LSQ**->**power **=** LSQ**->**power\_t **+** LSQ**->**local\_result**.**power **\***pppm\_lkg**;**  power **=** power **+** dcache**.**power **+** LSQ**->**power**;**  **if** **((**coredynp**.**core\_ty**==**OOO**)** **&&** **(**XML**->**sys**.**core**[**ithCore**].**load\_buffer\_size **>**0**))**  **{**  LoadQ**->**power **=** LoadQ**->**power\_t **+** LoadQ**->**local\_result**.**power **\***pppm\_lkg**;**  power **=** power **+** LoadQ**->**power**;**  **}**  **}** |

# DisplayEnergy函数

* 显示LoadStoreU的详细结果，包括面积，峰值功耗，漏电功耗和动态功耗

|  |
| --- |
| void LoadStoreU**::**displayEnergy**(**uint32\_t indent**,**int plevel**,**bool is\_tdp**)**  **{**  **if** **(!**exist**)** **return;**    **if** **(**is\_tdp**)**  **{**  /\*  Data Cache:    Area = dcache.area.get\_area()\*1e-6  Peak Dynamic = dcache.power.readOp.dynamic\*clockRate  Subthreshold Leakage = (long\_channel?  dcache.power.readOp.longer\_channel\_leakage:dcache.power.readOp.leakage )  Subthreshold Leakage with power gating = (long\_channel?  dcache.power.readOp.power\_gated\_with\_long\_channel\_leakage :  dcache.power.readOp.power\_gated\_leakage)  Runtime Dynamic = dcache.rt\_power.readOp.dynamic/executionTime  \*/    **if** **(**coredynp**.**core\_ty**==**Inorder**)**  **{**  //Load/Store Queue: LSQ  **}**  **else**  **{**  **if** **(**XML**->**sys**.**core**[**ithCore**].**load\_buffer\_size **>**0**)**  **{**  //LoadQ:  **}**  //StoreQ: LSQ  **}**  **}**  **else**  **{**  //Peak Dynamic = dcache.rt\_power.readOp.dynamic\*clockRate  //Subthreshold Leakage = dcache.rt\_power.readOp.leakage  //Gate Leakage = dcache.rt\_power.readOp.gate\_leakage  **if** **(**coredynp**.**core\_ty**==**Inorder**)**  **{**  //LSQ  **}**  **else**  **{**  //LSQ  //LoadQ  **}**  **}**  **}** |