目录

[1. InstFetchU类声明 2](#_Toc532847446)

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

[3. computerEnergy函数 5](#_Toc532847448)

[4. DisplayEnergy函数 7](#_Toc532847449)

# InstFetchU类声明

* 该类主要定义了Core中的取指阶段。类中定义的对象有指令缓存(icache, 包括icache中的一些组件, caches, missb, ifb, prefetchb)，指令buffer(IB)，BTB，分支预测器BPT和三个译码器（译码器目前不关心）
* 在定义的过程中，首先会根据xml中提供的预测器宽度来判断是否有预测器，如果有此时才会定义BTB和BPT，否则将不会定义
* 分支预测器采用的是alpha 21264中的锦标赛两级分支预测器，该类单独定义。

|  |
| --- |
| //core内的第一个对象，取指单元  class InstFetchU **:**public Component **{**  public**:**  //获取参数  ParseXML **\***XML**;**  //核的标号  int ithCore**;**  //传递给cacti的接口  InputParameter interface\_ip**;**  //核的动态参数，计算使用  CoreDynParam coredynp**;**  //基本参数，与core的一样  double clockRate**,**executionTime**;**  double scktRatio**,** chip\_PR\_overhead**,** macro\_PR\_overhead**;**  //WT/WB,basic\_components.h 77  enum Cache\_policy cache\_p**;**  //指令cache的对象,array.h 70  InstCache icache**;**  //Instruction buffer指令缓冲区，用于缓存从cache中取到的指令  //array.h 90  ArrayST **\*** IB**;**  //BTB  ArrayST **\*** BTB**;**  //分支预测器 branch pattern table  BranchPredictor **\*** BPT**;**  //指令译码器,指令译码，操作数译码，混合指令  //logic.h 95  inst\_decoder **\*** ID\_inst**;**  inst\_decoder **\*** ID\_operand**;**  inst\_decoder **\*** ID\_misc**;**  //用于判断有没有生成该类的对象，析构函数中判断  bool exist**;**    //构造函数，设置参数，计算面积  InstFetchU**(**ParseXML **\***XML\_interface**,** int ithCore\_**,**  InputParameter**\*** interface\_ip\_**,**const CoreDynParam **&** dyn\_p\_**,**  bool exsit**=true);**  //计算功率，power，rt\_power  void computeEnergy**(**bool is\_tdp**=true);**  //显示  void displayEnergy**(**uint32\_t indent **=** 0**,**int plevel **=** 100**,** bool is\_tdp**=true);**  **~**InstFetchU**();**  **};** |

# InstFetchU的构造函数

* 构造函数主要定义了类中的各个对象，以及icache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。对于译码器，由于该类中进一步定义了cacti中的Deocde对象，因此获取它的面积使用get\_area函数
* 具体的代码

|  |
| --- |
| InstFetchU**::**InstFetchU**(**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\_**),**IB **(**0**),**BTB **(**0**),**  ID\_inst **(**0**),**ID\_operand **(**0**),**ID\_misc **(**0**),**exist**(**exist\_**)**  **{**  **if** **(!**exist**)** **return;**  //cacti/parameter.cc:45:TechnologyParameter g\_tp  //一些工艺参数  scktRatio **=** g\_tp**.**sckt\_co\_eff**;**  chip\_PR\_overhead **=** g\_tp**.**chip\_layout\_overhead**;**  macro\_PR\_overhead **=** g\_tp**.**macro\_layout\_overhead**;**  //创建icache中的一些对象，caches, missb, ifb, prefetchb  //将这些组件的面积大小累加到icache的area和InstFetchU的area中  icache**.**caches **=** **new** ArrayST**(&**interface\_ip**,** "icache"**);**  //area=icache.missb->local\_result.area    //创建指令buffer  IB **=** **new** ArrayST**(&**interface\_ip**,** "InstBuffer"**);**  //IB->local\_result.area    //如果预测宽度大于零，意味着有预测器  **if** **(**coredynp**.**predictionW**>**0**)**  **{**  //创建BTB和BPT对象  BTB **=** **new** ArrayST**(&**interface\_ip**,** "Branch Target Buffer"**);**  //BTB->local\_result.area  BPT **=** **new** BranchPredictor**(**XML**,** ithCore**,** **&**interface\_ip**,**coredynp**);**  //BPT->area.get\_area()  **}**    //暂时不关心  //译码指令类型  ID\_inst **=** **new** inst\_decoder**(**is\_default**,** **&**interface\_ip**,**  coredynp**.**opcode\_length**,** 1**,** coredynp**.**x86**,**  Core\_device**,** coredynp**.**core\_ty**);**  //译码指令中的寄存器  ID\_operand **=** **new** inst\_decoder**(**is\_default**,** **&**interface\_ip**,**  coredynp**.**arch\_ireg\_width**,** 1**,**coredynp**.**x86**,**  Core\_device**,** coredynp**.**core\_ty**);**  //译码x86指令？  ID\_misc **=** **new** inst\_decoder**(**is\_default**,** **&**interface\_ip**,**  8**,** 1**,**coredynp**.**x86**,**  Core\_device**,** coredynp**.**core\_ty**);**    //根据译码宽度判断译码器的个数，然后将计算的面积放入InstFetchU的area中  area**.**set\_area**(**area**.**get\_area**()+** coredynp**.**decodeW**\*(**  ID\_inst**->**area**.**get\_area**()**  **+**ID\_operand**->**area**.**get\_area**()**  **+**ID\_misc**->**area**.**get\_area**()));**  **}** |

# 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
* 初始化组件的状态信息

|  |
| --- |
| **if** **(**is\_tdp**)**//初始化组件在每周期内的状态参数  **{**  //此时计算的是caches每个周期中的操作数目，  //为了之后计算峰值功耗，因此有些数据会较高  //对于caches会有三个操作，访问，miss，hit  //access=读写端口数\*取指部件的占空比(一个周期内的工作时间)  icache**.**caches**->**stats\_t**.**readAc**.**access **=** 、  icache**.**caches**->**l\_ip**.**num\_rw\_ports**\***coredynp**.**IFU\_duty\_cycle**;**  //miss数被设置为0  icache**.**caches**->**stats\_t**.**readAc**.**miss **=** 0**;**  //mcpat假设都命中  icache**.**caches**->**stats\_t**.**readAc**.**hit **=**  icache**.**caches**->**stats\_t**.**readAc**.**access **-** icache**.**caches**->**stats\_t**.**readAc**.**miss**;**  **}**  **else**//初始化整个运行过程中的组件的状态参数  **{**  //此时计算的是在整个运行过程中，组件的操作数目  //这是为了之后计算组件的运行时的动态功耗  //access=xml中给定的icache的读取次数  icache**.**caches**->**stats\_t**.**readAc**.**access **=**  XML**->**sys**.**core**[**ithCore**].**icache**.**read\_accesses**;**  //miss也是xml中获取的信息  icache**.**caches**->**stats\_t**.**readAc**.**miss **=**  XML**->**sys**.**core**[**ithCore**].**icache**.**read\_misses**;**  //hit为两者之差  icache**.**caches**->**stats\_t**.**readAc**.**hit **=**  icache**.**caches**->**stats\_t**.**readAc**.**access **-** icache**.**caches**->**stats\_t**.**readAc**.**miss**;**  **}** |

* 初始化状态之后，需要计算能耗(操作个数\*操作消耗的能量)。

|  |
| --- |
| //将power中的参数初始化为0  IB**->**power\_t**.**reset**();**  //总的能耗=操作数\*操作的能耗  //energy=readOp.dynamic\*readAc.access+writeAc.access\*writeOp.dynamic  IB**->**power\_t**.**readOp**.**dynamic **+=**  IB**->**local\_result**.**power**.**readOp**.**dynamic**\***IB**->**stats\_t**.**readAc**.**access **+**  IB**->**stats\_t**.**writeAc**.**access**\***IB**->**local\_result**.**power**.**writeOp**.**dynamic**;** |

* 最后计算泄露功耗，将结果保存在InstFetchU的power/rt\_power中

|  |
| --- |
| **if(**tdp**)**  **{**  //pppm\_lkg={0,1,1,0}  //power的四个参数是：dynamic, leakage, gate\_leakage, short\_circuit  //因此结果是将local\_result.power的leakage和gate\_leakage  //直接放入到IB->power的这两个参数中  //IB->power的dynamic和short\_circuit为IB->power\_t的两个参数  IB**->**power **=** IB**->**power\_t **+** IB**->**local\_result**.**power**\***pppm\_lkg**;**  //在tdp的情况下，将结果放入到InstFetchU的power中，否则放入rt\_powet中  power **=** power **+** icache**.**power **+** IB**->**power**;**  **}** |

# DisplayEnergy函数

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

|  |
| --- |
| void InstFetchU**::**displayEnergy**(**uint32\_t indent**,**int plevel**,**bool is\_tdp**)**  **{**  **if** **(!**exist**)** **return;**  **if** **(**is\_tdp**)**  **{**  //icache  //Area = icache.area.get\_area()\*1e-6  //Peak Dynamic = icache.power.readOp.dynamic\*clockRate  //Subthreshold Leakage = (long\_channel?  // icache.power.readOp.longer\_channel\_leakage:icache.power.readOp.leakage)  //Subthreshold Leakage with power gating = (long\_channel?  // icache.power.readOp.power\_gated\_with\_long\_channel\_leakage  // : icache.power.readOp.power\_gated\_leakage)  //Gate Leakage = icache.power.readOp.gate\_leakage  //Runtime Dynamic = icache.rt\_power.readOp.dynamic/executionTime    **if** **(**coredynp**.**predictionW**>**0**)**  **{**  //BTB  **if** **(**BPT**->**exist**)**  **{**  //BPT  **if** **(**plevel**>**3**)**  **{**  BPT**->**displayEnergy**(**indent**+**4**,** plevel**,** is\_tdp**);**  **}**  **}**  **}**  //IB, Instruction Decoder:(ID\_inst+ID\_operand+ID\_misc)  **}**  **}** |