目录

[1. SchedulerU类声明 2](#_Toc532913896)

[2. SchedulerU的构造函数 3](#_Toc532913897)

[3. computerEnergy函数 4](#_Toc532913898)

[4. DisplayEnergy函数 7](#_Toc532913899)

# SchedulerU类声明

* 该类是Core中的调度单元的功能实现。类中定义了三个ArrayST对象：int\_inst\_window, fp\_inst\_window, ROB和一个选择逻辑(selection\_logic)对象instruction\_selection
* 在按序硬件多线程处理器中，才会定义int\_inst\_window，此时的主要功能是从多个线程中选择合适的指令
* 在面积方面，三个ArrayST对象的面积都需要将流水线的个数考虑进去，对于选择逻辑，mcpat没有计算面积
* 由于bypass逻辑会连接到这三个ArrayST对象，因此需要计算它们的height(cacti会自动计算出cache\_ht)
* 声明代码

|  |
| --- |
| //调度器  class SchedulerU **:**public Component **{**  public**:**  ParseXML **\***XML**;**  int ithCore**;**  InputParameter interface\_ip**;**  CoreDynParam coredynp**;**  double clockRate**,**executionTime**;**  double scktRatio**,** chip\_PR\_overhead**,** macro\_PR\_overhead**;**  double Iw\_height**,** fp\_Iw\_height**,**ROB\_height**;**    //定点指令窗口，int IQ  ArrayST **\*** int\_inst\_window**;**  //浮点指令串口，float IQ  ArrayST **\*** fp\_inst\_window**;**  //ROB  ArrayST **\*** ROB**;**  //选择逻辑,logic.h 52  selection\_logic **\*** instruction\_selection**;**  bool exist**;**  SchedulerU**(**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);**  **~**SchedulerU**();**  **};** |

# SchedulerU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

|  |
| --- |
| SchedulerU**::**SchedulerU**(**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\_**),**int\_inst\_window**(**0**),**  fp\_inst\_window**(**0**),**ROB**(**0**),**instruction\_selection**(**0**),**  exist**(**exist\_**)**  **{**  **if** **(!**exist**)** **return;**  bool is\_default**=true;**  //按序处理器中(多线程的情况)只定义了整型的指令窗口  //主要是从多个线程中选择指令  **if** **((**coredynp**.**core\_ty**==**Inorder **&&** coredynp**.**multithreaded**))**  **{**  int\_inst\_window **=** **new** ArrayST**(**"InstFetchQueue"**);**  //area=int\_inst\_window->local\_result.area\*coredynp.num\_pipelines  //bypass逻辑会连到指令窗口，因此需要计算height  Iw\_height **=**int\_inst\_window**->**local\_result**.**cache\_ht**;**  //对于指令的选择逻辑，没有考虑面积影响，只计算了功耗  instruction\_selection **=** **new** selection\_logic**(**  coredynp**.**peak\_issueW**\***XML**->**sys**.**core**[**ithCore**].**number\_hardware\_threads**);**  **}**  //乱序的情况下，则定义了整型，浮点的指令窗口和ROB  **if** **(**coredynp**.**core\_ty**==**OOO**)**  **{**  //tmp\_name=issueQueue / Reservation Station  int\_inst\_window **=** **new** ArrayST**(**tmp\_name**);**  //area = int\_inst\_window->local\_result.area\*coredynp.num\_pipelines);    //计算height  Iw\_height **=**int\_inst\_window**->**local\_result**.**cache\_ht**;**  fp\_inst\_window **=** **new** ArrayST**(**tmp\_name**);**  fp\_Iw\_height **=**fp\_inst\_window**->**local\_result**.**cache\_ht**;**  **if** **(**XML**->**sys**.**core**[**ithCore**].**ROB\_size **>**0**)**  **{**  ROB **=** **new** ArrayST**(**"ReorderBuffer"**);**  //area=ROB->local\_result.area\*coredynp.num\_pipelines    //bypass也会连接到ROB中  ROB\_height **=**ROB**->**local\_result**.**cache\_ht**;**  **}**  //不计算选择逻辑的面积  instruction\_selection **=** **new** selection\_logic**(**coredynp**.**peak\_issueW**);**  **}**  **}** |

# 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 SchedulerUtion 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：（假定ROB每周期都在工作，ROB\_duty\_cycle = 1）
  + 对于乱序核，需要初始化三个对象的基本参数，此时指令窗口的查找操作也需要被记录下来
  + 对于ROB，当指令提交的时候一定会被读取一次，当指令插入时/当结果产生时一定会被写入一次，但是没有查找操作，因为是按序的访问

|  |
| --- |
| **if** **(**is\_tdp**)**  **{**  //如果是乱序核，则包括三个对象  **if** **(**coredynp**.**core\_ty**==**OOO**)**  **{**  //三个操作，查找，读取，写入  //用发射宽度确定  int\_inst\_window**->**stats\_t**.**readAc**.**access **=**  coredynp**.**issueW**\***coredynp**.**num\_pipelines**;**  int\_inst\_window**->**stats\_t**.**writeAc**.**access **=**  coredynp**.**issueW**\***coredynp**.**num\_pipelines**;**  int\_inst\_window**->**stats\_t**.**searchAc**.**access **=**  coredynp**.**issueW**\***coredynp**.**num\_pipelines**;**    //用端口数确定，假设每周期都在工作  fp\_inst\_window**->**stats\_t**.**readAc**.**access **=**  fp\_inst\_window**->**l\_ip**.**num\_rd\_ports**\***coredynp**.**num\_fp\_pipelines**;**  fp\_inst\_window**->**stats\_t**.**writeAc**.**access **=**  fp\_inst\_window**->**l\_ip**.**num\_wr\_ports**\***coredynp**.**num\_fp\_pipelines**;**  fp\_inst\_window**->**stats\_t**.**searchAc**.**access **=**  fp\_inst\_window**->**l\_ip**.**num\_search\_ports**\***coredynp**.**num\_fp\_pipelines**;**  **if** **(**XML**->**sys**.**core**[**ithCore**].**ROB\_size **>**0**)**  **{**  //当指令提交时，ROB一定会被读取一次  //PRF时读取物理寄存器tag，RS时读取数据  //没有查找操作，因为是顺序的访问  ROB**->**stats\_t**.**readAc**.**access **=**  coredynp**.**commitW**\***coredynp**.**num\_pipelines**\***ROB\_duty\_cycle**;**  ROB**->**stats\_t**.**writeAc**.**access **=**  coredynp**.**issueW**\***coredynp**.**num\_pipelines**\***ROB\_duty\_cycle**;**  **}**  **}**  **else** **if** **(**coredynp**.**multithreaded**)**//按序多线程的情况下  **{**  int\_inst\_window**->**stats\_t**.**readAc**.**access **=**  coredynp**.**issueW**\***coredynp**.**num\_pipelines**;**  int\_inst\_window**->**stats\_t**.**writeAc**.**access **=**  coredynp**.**issueW**\***coredynp**.**num\_pipelines**;**  int\_inst\_window**->**stats\_t**.**searchAc**.**access **=**  coredynp**.**issueW**\***coredynp**.**num\_pipelines**;**  **}**  **}**  **else**  **{**  **if** **(**coredynp**.**core\_ty**==**OOO**)**  **{**  int\_inst\_window**->**stats\_t**.**readAc**.**access **=**  XML**->**sys**.**core**[**ithCore**].**inst\_window\_reads**;**  int\_inst\_window**->**stats\_t**.**writeAc**.**access **=**  XML**->**sys**.**core**[**ithCore**].**inst\_window\_writes**;**  int\_inst\_window**->**stats\_t**.**searchAc**.**access **=**  XML**->**sys**.**core**[**ithCore**].**inst\_window\_wakeup\_accesses**;**  fp\_inst\_window**->**stats\_t**.**readAc**.**access **=**  XML**->**sys**.**core**[**ithCore**].**fp\_inst\_window\_reads**;**  fp\_inst\_window**->**stats\_t**.**writeAc**.**access **=**  XML**->**sys**.**core**[**ithCore**].**fp\_inst\_window\_writes**;**  fp\_inst\_window**->**stats\_t**.**searchAc**.**access **=**  XML**->**sys**.**core**[**ithCore**].**fp\_inst\_window\_wakeup\_accesses**;**  **if** **(**XML**->**sys**.**core**[**ithCore**].**ROB\_size **>**0**)**  **{**  //ROB的写操作：当指令插入时/当结果产生时  //ROB的读操作：当指令提交时  ROB**->**stats\_t**.**readAc**.**access **=**  XML**->**sys**.**core**[**ithCore**].**ROB\_reads**;**  ROB**->**stats\_t**.**writeAc**.**access **=**  XML**->**sys**.**core**[**ithCore**].**ROB\_writes**;**  **}**  **}**  **else** **if** **(**coredynp**.**multithreaded**)**  **{**  int\_inst\_window**->**stats\_t**.**readAc**.**access **=**  XML**->**sys**.**core**[**ithCore**].**int\_instructions  **+** XML**->**sys**.**core**[**ithCore**].**fp\_instructions**;**  int\_inst\_window**->**stats\_t**.**writeAc**.**access **=**  XML**->**sys**.**core**[**ithCore**].**int\_instructions  **+** XML**->**sys**.**core**[**ithCore**].**fp\_instructions**;**  //两个操作数都要查找指令窗口  int\_inst\_window**->**stats\_t**.**searchAc**.**access **=**  2**\*(**XML**->**sys**.**core**[**ithCore**].**int\_instructions  **+** XML**->**sys**.**core**[**ithCore**].**fp\_instructions**);**  **}**  **}** |

* 初始化状态之后，需要计算能耗(操作个数\*操作消耗的能量)。对于指令窗口而言，包括读操作，写操作，搜索操作和在该指令窗口(int/fp)发生的选择逻辑带来的能耗。对于选择逻辑的参与次数设置为指令窗口的读次数
* 最后计算泄露功耗(需要考虑线程数)，将结果保存在SchedulerU的power/rt\_power中

|  |
| --- |
| //赋值，基本一致，目的不同：power/rt\_power  **if** **(**is\_tdp**)**  **{**  **if** **(**coredynp**.**core\_ty**==**OOO**)**  **{**  //pppm\_lkg={0,1,1,0}  //计算泄露功耗  int\_inst\_window**->**power **=** int\_inst\_window**->**power\_t **+** pppm\_lkg **\***  **(**int\_inst\_window**->**local\_result**.**power**+**instruction\_selection**->**power**);**  fp\_inst\_window**->**power **=** fp\_inst\_window**->**power\_t **+** pppm\_lkg **\***  **(**fp\_inst\_window**->**local\_result**.**power **+**instruction\_selection**->**power**);**  power **=** power **+** int\_inst\_window**->**power **+** fp\_inst\_window**->**power**;**  **if** **(**XML**->**sys**.**core**[**ithCore**].**ROB\_size **>**0**)**  **{**  ROB**->**power **=** ROB**->**power\_t **+** ROB**->**local\_result**.**power**\***pppm\_lkg**;**  power **=** power **+** ROB**->**power**;**  **}**  **}**  **else** **if** **(**coredynp**.**multithreaded**)**  **{**  int\_inst\_window**->**power **=** int\_inst\_window**->**power\_t **+** pppm\_lkg **\***  **(**int\_inst\_window**->**local\_result**.**power **+**instruction\_selection**->**power**);**  power **=** power **+** int\_inst\_window**->**power**;**  **}**  **}** |

# DisplayEnergy函数

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

|  |
| --- |
| void SchedulerU**::**displayEnergy**(**uint32\_t indent**,**int plevel**,**bool is\_tdp**)**  **{**  **if** **(!**exist**)** **return;**  bool long\_channel **=** XML**->**sys**.**longer\_channel\_device**;**  bool power\_gating **=** XML**->**sys**.**power\_gating**;**  **if** **(**is\_tdp**)**  **{**  **if** **(**coredynp**.**core\_ty**==**OOO**)**  **{**  /\*  Instruction Window:  FP Instruction Window:  ROB:      Area = int\_inst\_window->area.get\_area()\*1e-6  Peak Dynamic = int\_inst\_window->power.readOp.dynamic\*clockRate  Subthreshold Leakage = (long\_channel?  int\_inst\_window->power.readOp.longer\_channel\_leakage  :int\_inst\_window->power.readOp.leakage)  Subthreshold Leakage with power gating = long\_channel?  int\_inst\_window->power.readOp.power\_gated\_with\_long\_channel\_leakage :  int\_inst\_window->power.readOp.power\_gated\_leakage  Gate Leakage = int\_inst\_window->power.readOp.gate\_leakage  Runtime Dynamic = int\_inst\_window->rt\_power.readOp.dynamic/executionTime  \*/  **}**  **else** **if** **(**coredynp**.**multithreaded**)**  **{**  //Instruction Window:  **}**  **}**  **else**  **{**  **if** **(**coredynp**.**core\_ty**==**OOO**)**  **{**  //Peak Dynamic = int\_inst\_window->rt\_power.readOp.dynamic\*clockRate  //Subthreshold Leakage = int\_inst\_window->rt\_power.readOp.leakage  //Gate Leakage = int\_inst\_window->rt\_power.readOp.gate\_leakage  **}**  **else** **if** **(**coredynp**.**multithreaded**)**  **{**  //Instruction Window:  **}**  **}**  **}** |