目录

[1. EXECU类声明 2](#_Toc532847446)

[2. EXECU的构造函数 3](#_Toc532847447)

[3. computerEnergy函数 4](#_Toc532847448)

[4. DisplayEnergy函数 7](#_Toc532847449)

# EXECU类声明

* 该类是Core中的执行阶段的定义
* 该类定义的对象包括了功能单元，寄存器，调度单元以及功能部件和其它结构之间的bypass逻辑

|  |  |  |  |
| --- | --- | --- | --- |
| EXECU | rfu(RegFU)  fp\_u(FuctionalUnit)  int\_bypass  (interconnect)  intTagBypass  (interconnect) | scheu(SchedulerU)  exeu(FuctionalUnit)  int\_mul\_bypass  (interconnect)  intTag\_mul\_Bypass  (interconnect) | mul(FuctionalUnit)  fp\_bypass  (interconnect)  fpTagBypass  (interconnect) |

* 声明代码

|  |
| --- |
| //执行单元  class EXECU **:**public Component **{**  public**:**  //一样的参数  ParseXML **\***XML**;**  int ithCore**;**  InputParameter interface\_ip**;**  double clockRate**,**executionTime**;**  double scktRatio**,** chip\_PR\_overhead**,** macro\_PR\_overhead**;**    //线程数？  double lsq\_height**;**  CoreDynParam coredynp**;**    //寄存器单元  RegFU **\*** rfu**;**  //调度器 IQ  SchedulerU **\*** scheu**;**  //功能部件：浮点，定点，乘法  //logic.h 189  FunctionalUnit **\*** fp\_u**;**  FunctionalUnit **\*** exeu**;**  FunctionalUnit **\*** mul**;**  //互连结构，定点bypass，乘法bypass，浮点bypass和相应的tag比较  //interconnect.h 48  interconnect **\*** int\_bypass**;**  interconnect **\*** intTagBypass**;**  interconnect **\*** int\_mul\_bypass**;**  interconnect **\*** intTag\_mul\_Bypass**;**  interconnect **\*** fp\_bypass**;**  interconnect **\*** fpTagBypass**;**    //用于记录信息的  Component bypass**;**  bool exist**;**  EXECU**(**ParseXML **\***XML\_interface**,** int ithCore\_**,** InputParameter**\*** interface\_ip\_**,** double lsq\_height\_**,**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);**  **~**EXECU**();**  **};** |

# EXECU的构造函数

* 构造函数主要定义了类中的各个对象，然后计算每个对象的面积，累加得到该单元的面积
* 在乱序的处理器中，调度策略可能是RS或者是PRF，此时bypass需要传递的参数将会不同，因此需要考虑
* Bypass需要使用相关组件的height信息

|  |
| --- |
| EXECU**::**EXECU**(**ParseXML**\*** XML\_interface**,** int ithCore\_**,**  InputParameter**\*** interface\_ip\_**,** double lsq\_height\_**,**  const CoreDynParam **&** dyn\_p\_**,** bool exist\_**)**  **:**XML**(**XML\_interface**),**ithCore**(**ithCore\_**),**  interface\_ip**(\***interface\_ip\_**),**lsq\_height**(**lsq\_height\_**),**  coredynp**(**dyn\_p\_**),**rfu**(**0**),**scheu**(**0**),**fp\_u**(**0**),**  exeu**(**0**),**mul**(**0**),**int\_bypass**(**0**),**intTagBypass**(**0**),**  int\_mul\_bypass**(**0**),**intTag\_mul\_Bypass**(**0**),**fp\_bypass**(**0**),**  fpTagBypass**(**0**),**exist**(**exist\_**)**  **{**  **if** **(!**exist**)** **return;**  //没有用处  double fu\_height **=** 0.0**;**  //创建寄存器堆对象，调度单元对象，定点执行单元对象  //并将这些组件的面积放入EXECU的area对象中  rfu **=** **new** RegFU**(**XML**,** ithCore**,** **&**interface\_ip**,**coredynp**);**  scheu **=** **new** SchedulerU**(**XML**,** ithCore**,** **&**interface\_ip**,**coredynp**);**  exeu **=** **new** FunctionalUnit**(**XML**,** ithCore**,&**interface\_ip**,** coredynp**,** ALU**);**  area**.**set\_area**(**area**.**get\_area**()+** exeu**->**area**.**get\_area**()** **+**  rfu**->**area**.**get\_area**()** **+**scheu**->**area**.**get\_area**()** **);**    //如果有fpu，则创建fpu对象  //对象返回的参数已经考虑到了fpu的个数问题  **if** **(**coredynp**.**num\_fpus **>**0**)**  **{**  fp\_u **=** **new** FunctionalUnit**(**XML**,** ithCore**,&**interface\_ip**,** coredynp**,** FPU**);**  area**.**set\_area**(**area**.**get\_area**()+** fp\_u**->**area**.**get\_area**());**  **}**  //如果有乘法器，则创建mul对象  //该对象似乎是定点乘法  **if** **(**coredynp**.**num\_muls **>**0**)**  **{**  mul **=** **new** FunctionalUnit**(**XML**,** ithCore**,&**interface\_ip**,** coredynp**,** MUL**);**  area**.**set\_area**(**area**.**get\_area**()+** mul**->**area**.**get\_area**());**  **}**  //如果是按序处理器的情况下，创建6个bypass逻辑对象(data/tag\*mul/alu/fpu)  **if** **(**coredynp**.**core\_ty**==**Inorder**)**  **{**  int\_bypass **=** **new** interconnect**(**"Int Bypass Data"**);**  bypass**.**area**.**set\_area**(**bypass**.**area**.**get\_area**()** **+** int\_bypass**->**area**.**get\_area**());**  intTagBypass **=** **new** interconnect**(**"Int Bypass tag" **);**  bypass**.**area**.**set\_area**(**bypass**.**area**.**get\_area**()**  **+**intTagBypass**->**area**.**get\_area**());**  **if** **(**coredynp**.**num\_muls**>**0**)**  **{**  int\_mul\_bypass **=** **new** interconnect**(**"Mul Bypass Data"**);**  bypass**.**area**.**set\_area**(**bypass**.**area**.**get\_area**()**  **+**int\_mul\_bypass**->**area**.**get\_area**());**  intTag\_mul\_Bypass **=** **new** interconnect**(**"Mul Bypass tag"**);**  bypass**.**area**.**set\_area**(**bypass**.**area**.**get\_area**()**  **+**intTag\_mul\_Bypass**->**area**.**get\_area**());**  **}**  **if** **(**coredynp**.**num\_fpus**>**0**)**  **{**  fp\_bypass **=** **new** interconnect**(**"FP Bypass Data"**);**  bypass**.**area**.**set\_area**(**bypass**.**area**.**get\_area**()**  **+**fp\_bypass**->**area**.**get\_area**());**  fpTagBypass **=** **new** interconnect**(**"FP Bypass tag"**);**  bypass**.**area**.**set\_area**(**bypass**.**area**.**get\_area**()**  **+**fpTagBypass**->**area**.**get\_area**());**  **}**  **}**  **else**//在乱序的情况下，当调度策略不同时，创建bypass需要传递的参数也不同  **{**  **if** **(**coredynp**.**scheu\_ty**==**PhysicalRegFile**)**  **{**  //int\_bypass,intTagBypass  //int\_mul\_bypass,intTag\_mul\_Bypass  //fp\_bypass,fpTagBypass  **}**  **else**  **{**  //int\_bypass,intTagBypass  //int\_mul\_bypass,intTag\_mul\_Bypass  //fp\_bypass,fpTagBypass  **}**  **}**  //加上bypass的面积  area**.**set\_area**(**area**.**get\_area**()+** bypass**.**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
* 需要注意的是在计算bypass逻辑的时候，因为一条指令可能会传递2个操作数，因此需要两个bypass逻辑（浮点需要三个）

|  |
| --- |
| void EXECU**::**computeEnergy**(**bool is\_tdp**)**  **{**  **if** **(!**exist**)** **return;**  double pppm\_t**[**4**]** **=** **{**1**,**1**,**1**,**1**};**  //计算rfu,scheu,exeu的功耗，结果保存在power和rt\_power中  rfu**->**computeEnergy**(**is\_tdp**);**  scheu**->**computeEnergy**(**is\_tdp**);**  exeu**->**computeEnergy**(**is\_tdp**);**    //判断fpu和mul是否存在，如果存在计算功耗  **if** **(**coredynp**.**num\_fpus **>**0**)**  **{**  fp\_u**->**computeEnergy**(**is\_tdp**);**  **}**  **if** **(**coredynp**.**num\_muls **>**0**)**  **{**  mul**->**computeEnergy**(**is\_tdp**);**  **}**  //保存结果的目的对象不同  **if** **(**is\_tdp**)**  **{**  /\* power的前四个属性值  double dynamic;  double leakage;  double gate\_leakage;  double short\_circuit;  \*/  //2是指每个定点指令需要两个通过bypass传递2个操作数  //不太清楚是两个bypass还是两次，应该是前者  set\_pppm**(**pppm\_t**,** 2**\***coredynp**.**ALU\_cdb\_duty\_cycle**,**  2**,** 2**,** 2**\***coredynp**.**ALU\_cdb\_duty\_cycle**);**  bypass**.**power **=** bypass**.**power **+**  intTagBypass**->**power**\***pppm\_t **+** int\_bypass**->**power**\***pppm\_t**;**  **if** **(**coredynp**.**num\_muls **>**0**)**  **{**  set\_pppm**(**pppm\_t**,** 2**\***coredynp**.**MUL\_cdb\_duty\_cycle**,**  2**,** 2**,** 2**\***coredynp**.**MUL\_cdb\_duty\_cycle**);**  bypass**.**power **=** bypass**.**power **+**  intTag\_mul\_Bypass**->**power**\***pppm\_t **+** int\_mul\_bypass**->**power**\***pppm\_t**;**  power **=** power **+** mul**->**power**;**  **}**  **if** **(**coredynp**.**num\_fpus**>**0**)**  **{**  //浮点需要三个bypass路径  set\_pppm**(**pppm\_t**,** 3**\***coredynp**.**FPU\_cdb\_duty\_cycle**,**  3**,** 3**,** 3**\***coredynp**.**FPU\_cdb\_duty\_cycle**);**  bypass**.**power **=** bypass**.**power **+**  fp\_bypass**->**power**\***pppm\_t **+** fpTagBypass**->**power**\***pppm\_t **;**  power **=** power **+** fp\_u**->**power**;**  **}**  power **=** power **+** rfu**->**power **+** exeu**->**power  **+** bypass**.**power **+** scheu**->**power**;**  **}**  **else**  **{**  set\_pppm**(**pppm\_t**,** XML**->**sys**.**core**[**ithCore**].**cdb\_alu\_accesses**,**  2**,** 2**,** XML**->**sys**.**core**[**ithCore**].**cdb\_alu\_accesses**);**  bypass**.**rt\_power **=** bypass**.**rt\_power **+**  intTagBypass**->**power**\***pppm\_t **+** int\_bypass**->**power**\***pppm\_t**;**  **if** **(**coredynp**.**num\_muls **>**0**)**  **{**  set\_pppm**(**pppm\_t**,** XML**->**sys**.**core**[**ithCore**].**cdb\_mul\_accesses**,**  2**,** 2**,** XML**->**sys**.**core**[**ithCore**].**cdb\_mul\_accesses**);**  bypass**.**rt\_power **=** bypass**.**rt\_power **+**  intTag\_mul\_Bypass**->**power**\***pppm\_t **+** int\_mul\_bypass**->**power**\***pppm\_t**;**  rt\_power **=** rt\_power **+** mul**->**rt\_power**;**  **}**  **if** **(**coredynp**.**num\_fpus**>**0**)**  **{**  set\_pppm**(**pppm\_t**,** XML**->**sys**.**core**[**ithCore**].**cdb\_fpu\_accesses**,**  3**,** 3**,** XML**->**sys**.**core**[**ithCore**].**cdb\_fpu\_accesses**);**  bypass**.**rt\_power **=** bypass**.**rt\_power **+** fp\_bypass**->**power**\***pppm\_t**;**  bypass**.**rt\_power **=** bypass**.**rt\_power **+** fpTagBypass**->**power**\***pppm\_t**;**  rt\_power **=** rt\_power **+** fp\_u**->**rt\_power**;**  **}**  rt\_power **=** rt\_power **+** rfu**->**rt\_power **+** exeu**->**rt\_power  **+** bypass**.**rt\_power **+** scheu**->**rt\_power**;**  **}**  **}** |

# DisplayEnergy函数

* 显示EXECU的详细结果，包括面积，峰值功耗，漏电功耗和动态功耗

|  |
| --- |
| void EXECU**::**displayEnergy**(**uint32\_t indent**,**int plevel**,**bool is\_tdp**)**  **{**  **if** **(!**exist**)** **return;**  **if** **(**is\_tdp**)**  **{**  /\*  Register Files:rfu  Instruction Scheduler:scheu  Results Broadcast Bus:bypass  Area = rfu->area.get\_area()\*1e-6  Peak Dynamic = rfu->power.readOp.dynamic\*clockRate  Subthreshold Leakage = (long\_channel?  rfu->power.readOp.longer\_channel\_leakage:rfu->power.readOp.leakage)  Subthreshold Leakage with power gating = (long\_channel?  rfu->power.readOp.power\_gated\_with\_long\_channel\_leakage :  rfu->power.readOp.power\_gated\_leakage)  Runtime Dynamic = rfu->rt\_power.readOp.dynamic/executionTime  \*/    **if** **(**plevel**>**3**){**  scheu**->**displayEnergy**(**indent**+**4**,**is\_tdp**);**  **}**  exeu**->**displayEnergy**(**indent**,**is\_tdp**);**    **if** **(**coredynp**.**num\_fpus**>**0**)**  **{**  fp\_u**->**displayEnergy**(**indent**,**is\_tdp**);**  **}**    **if** **(**coredynp**.**num\_muls **>**0**)**  **{**  mul**->**displayEnergy**(**indent**,**is\_tdp**);**  **}**    **}**  **else**  **{**  **}**  **}** |