目录

[1. RegFU类声明 2](#_Toc532893892)

[2. RegFU的构造函数 3](#_Toc532893893)

[3. computerEnergy函数 4](#_Toc532893894)

[4. DisplayEnergy函数 7](#_Toc532893895)

# RegFU类声明

* 该类是Core中用于定义寄存器堆的类，该类包括了IRF，FRF和RFWIN，三者均为ArrayST类型的对象。在按序流水线中，IRF和FRF的个数即为体系结构寄存器个数，在乱序流水线中，则为物理寄存器个数
* 处理器为每个线程提供分开的体系结构寄存器，因此在计算面积时需要将线程数考虑进去
* 在计算bypass逻辑的面积和功耗时，bypass的一端需要连接寄存器堆中，此时也需要考虑到处理器提供的线程独立的多个寄存器堆
* **RFWIN目前推测应该是为了保存异常发生时的寄存器堆或者是****函数调用时保存寄存器。**对于RFWIN，寄存器窗口，McPAT只在处理器为按序处理器，并且提供了相关参数的情况下，才会定义该对象。寄存器窗口RFWin不需要考虑线程的问题
* 除了考虑线程的问题，所有的对象都必须考虑处理器中流水线的个数。
* 声明代码

|  |
| --- |
| class RegFU **:**public Component **{**  public**:**  ParseXML **\***XML**;**  int ithCore**;**  InputParameter interface\_ip**;**  CoreDynParam coredynp**;**  double clockRate**,**executionTime**;**  double scktRatio**,** chip\_PR\_overhead**,** macro\_PR\_overhead**;**  double int\_regfile\_height**,** fp\_regfile\_height**;**    //int reg file  ArrayST **\*** IRF**;**  //float reg file  ArrayST **\*** FRF**;**  //reg window: physical register file  ArrayST **\*** RFWIN**;**  bool exist**;**  RegFU**(**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);**  **~**RegFU**();**  **};** |

# RegFU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

|  |
| --- |
| RegFU**::**RegFU**(**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\_**),**  IRF **(**0**),**FRF **(**0**),**RFWIN **(**0**),**exist**(**exist\_**)**  **{**  **if** **(!**exist**)** **return;**    //创建IRF，FRF对象  IRF **=** **new** ArrayST**(&**interface\_ip**,** "Integer Register File"**,**  Core\_device**,** coredynp**.**opt\_local**,** coredynp**.**core\_ty**);**  //计算IRF的面积=每个IRF/FRF的面积\*硬件支持的线程数\*流水线个数\*overhead  IRF**->**area**.**set\_area**(**IRF**->**area**.**get\_area**()+**  IRF**->**local\_result**.**area**\***XML**->**sys**.**core**[**ithCore**].**number\_hardware\_threads  **\***coredynp**.**num\_pipelines**\***cdb\_overhead**);**  //将面积的结果累加到RegFU的area对象中  area**.**set\_area**(**area**.**get\_area**()+** IRF**->**local\_result**.**area**\***  XML**->**sys**.**core**[**ithCore**].**number\_hardware\_threads  **\***coredynp**.**num\_pipelines**\***cdb\_overhead**);**    //用于计算bypass网络时使用  //此时需要考虑到bypass需要连接到所有寄存器，因此需要\*硬件支持的线程数  int\_regfile\_height**=** IRF**->**local\_result**.**cache\_ht  **\***XML**->**sys**.**core**[**ithCore**].**number\_hardware\_threads**\***sqrt**(**cdb\_overhead**);**  fp\_regfile\_height **=** FRF**->**local\_result**.**cache\_ht  **\***XML**->**sys**.**core**[**ithCore**].**number\_hardware\_threads**\***sqrt**(**cdb\_overhead**);**    //regWindowing=  // (register\_windows\_size>0&&coredynp.core\_ty==Inorder)?true:false;  //按序处理器并且指定了寄存器窗口的大小，此时为真，否则为假  **if** **(**coredynp**.**regWindowing**)**  **{**  //创建RFWIN对象，并且计算面积\*流水线条数  RFWIN **=** **new** ArrayST**(&**interface\_ip**,** "RegWindow"**,**  Core\_device**,** coredynp**.**opt\_local**,** coredynp**.**core\_ty**);**  RFWIN**->**area**.**set\_area**(**RFWIN**->**area**.**get\_area**()+**  RFWIN**->**local\_result**.**area**\***coredynp**.**num\_pipelines**);**  area**.**set\_area**(**area**.**get\_area**()+**  RFWIN**->**local\_result**.**area**\***coredynp**.**num\_pipelines**);**  **}**  **}** |

# 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 RegFUtion 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：
  + 在初始化每周期内的信息时，IRF的访问次数通过发射宽度和ALU部件的占空比计算，但是根据Thumb的统计判断，大约有10%的指令不会访问ALU，但是仍会访问寄存器，因此需要有一个1.1
  + 另一个问题，在初始化每周期的REGWIN信息时，McPAT没有考虑，直接设置为0，之后可能需要自己增加
  + 在初始化整体的组件信息时，当存在REGWIN时，则通过函数调用的次数和寄存器保存的个数来计算它的状态。因为REGWIN用于保存函数调用时的寄存器信息，因此使用这两个信息来反映其状态。
  + 考虑到函数调用时，不仅需要访问REGWIN，也需要访问IRF和FRF，因此需要在它们的状态中增加这些信息

|  |
| --- |
| **if** **(**is\_tdp**)**//初始化每周期的状态  **{**  //1.1的目的：大约有10%的指令不会访问ALU，因此ALU\_duty\_cycle\*1.1  //根据Thumb得到的结论  IRF**->**stats\_t**.**readAc**.**access **=** coredynp**.**issueW**\***2**\*(**coredynp**.**ALU\_duty\_cycle**\***1.1**+**  **(**coredynp**.**num\_muls**>**0**?**coredynp**.**MUL\_duty\_cycle**:**0**))\***coredynp**.**num\_pipelines**;**  IRF**->**stats\_t**.**writeAc**.**access **=** coredynp**.**issueW**\*(**coredynp**.**ALU\_duty\_cycle**\***1.1**+**  **(**coredynp**.**num\_muls**>**0**?**coredynp**.**MUL\_duty\_cycle**:**0**))\***coredynp**.**num\_pipelines**;**    //1.05的目的和1.1类似  FRF**->**stats\_t**.**readAc**.**access **=** FRF**->**l\_ip**.**num\_rd\_ports  **\***coredynp**.**FPU\_duty\_cycle**\***1.05**\***coredynp**.**num\_fp\_pipelines**;**  FRF**->**stats\_t**.**writeAc**.**access **=** FRF**->**l\_ip**.**num\_wr\_ports  **\***coredynp**.**FPU\_duty\_cycle**\***1.05**\***coredynp**.**num\_fp\_pipelines**;**  //似乎没有考虑这个东西，如果按序的情况下，需要自己更改  **if** **(**coredynp**.**regWindowing**)**  **{**  RFWIN**->**stats\_t**.**readAc**.**access **=** 0**;**//0.5\*RFWIN->l\_ip.num\_rw\_ports;  RFWIN**->**stats\_t**.**writeAc**.**access **=** 0**;**//0.5\*RFWIN->l\_ip.num\_rw\_ports;  **}**  **}**  **else**//初始化执行过程中的状态  **{**  //整个执行过程中的一些操作的次数  IRF**->**stats\_t**.**readAc**.**access **=** XML**->**sys**.**core**[**ithCore**].**int\_regfile\_reads**;**  IRF**->**stats\_t**.**writeAc**.**access **=** XML**->**sys**.**core**[**ithCore**].**int\_regfile\_writes**;**  FRF**->**stats\_t**.**readAc**.**access **=** XML**->**sys**.**core**[**ithCore**].**float\_regfile\_reads**;**  FRF**->**stats\_t**.**writeAc**.**access **=** XML**->**sys**.**core**[**ithCore**].**float\_regfile\_writes**;**    **if** **(**coredynp**.**regWindowing**)**  **{**  //在函数调用的时候(包括异常转移)时，保存寄存器  //函数调用次数\*需要保存的寄存器个数，16个  RFWIN**->**stats\_t**.**readAc**.**access **=** XML**->**sys**.**core**[**ithCore**].**function\_calls**\***16**;**  RFWIN**->**stats\_t**.**writeAc**.**access **=** XML**->**sys**.**core**[**ithCore**].**function\_calls**\***16**;**  //IRF和FPR也需要增加这个因素(访问时双向的)  IRF**->**stats\_t**.**readAc**.**access **=** XML**->**sys**.**core**[**ithCore**].**int\_regfile\_reads **+**  XML**->**sys**.**core**[**ithCore**].**function\_calls**\***16**;**  IRF**->**stats\_t**.**writeAc**.**access **=** XML**->**sys**.**core**[**ithCore**].**int\_regfile\_writes **+**  XML**->**sys**.**core**[**ithCore**].**function\_calls**\***16**;**  FRF**->**stats\_t**.**readAc**.**access **=** XML**->**sys**.**core**[**ithCore**].**float\_regfile\_reads **+**  XML**->**sys**.**core**[**ithCore**].**function\_calls**\***16**;;**  FRF**->**stats\_t**.**writeAc**.**access **=** XML**->**sys**.**core**[**ithCore**].**float\_regfile\_writes**+**  XML**->**sys**.**core**[**ithCore**].**function\_calls**\***16**;;**  **}**  **}** |

* 初始化状态之后，需要计算能耗(操作个数\*操作消耗的能量)

|  |
| --- |
| //初始化power参数  IRF**->**power\_t**.**reset**();**  FRF**->**power\_t**.**reset**();**  //计算能耗  //操作次数\*操作的能量消耗  IRF**->**power\_t**.**readOp**.**dynamic **+=** **(**  IRF**->**stats\_t**.**readAc**.**access**\***IRF**->**local\_result**.**power**.**readOp**.**dynamic  **+**IRF**->**stats\_t**.**writeAc**.**access**\***IRF**->**local\_result**.**power**.**writeOp**.**dynamic**);**  FRF**->**power\_t**.**readOp**.**dynamic **+=** **(**  FRF**->**stats\_t**.**readAc**.**access**\***FRF**->**local\_result**.**power**.**readOp**.**dynamic  **+**FRF**->**stats\_t**.**writeAc**.**access**\***FRF**->**local\_result**.**power**.**writeOp**.**dynamic**);**  **if** **(**coredynp**.**regWindowing**)**  **{**  RFWIN**->**power\_t**.**reset**();**  RFWIN**->**power\_t**.**readOp**.**dynamic **+=** **(**  RFWIN**->**stats\_t**.**readAc**.**access**\***RFWIN**->**local\_result**.**power**.**readOp**.**dynamic **+**  RFWIN**->**stats\_t**.**writeAc**.**access**\***RFWIN**->**local\_result**.**power**.**writeOp**.**dynamic**);**  **}** |

* 最后计算泄露功耗(需要考虑线程数)，将结果保存在RegFU的power/rt\_power中

|  |
| --- |
| //两者基本一样，保存结果的参数不同，power/rt\_power  **if** **(**is\_tdp**)**  **{**  //pppm\_lkg\_multhread={0, coredynp.num\_hthreads, coredynp.num\_hthreads, 0}  //泄露功耗需要考虑到有多少个硬件线程，因为每一个硬件线程都需要有寄存器  IRF**->**power**=**IRF**->**power\_t**+**IRF**->**local\_result**.**power**\***coredynp**.**pppm\_lkg\_multhread**;**  FRF**->**power**=**FRF**->**power\_t**+**FRF**->**local\_result**.**power**\***coredynp**.**pppm\_lkg\_multhread**;**  power **=**power **+** **(**IRF**->**power **+** FRF**->**power**);**    **if** **(**coredynp**.**regWindowing**)**  **{**  //pppm\_lkg={0,1,1,0}  RFWIN**->**power **=** RFWIN**->**power\_t **+** RFWIN**->**local\_result**.**power **\***pppm\_lkg**;**  power **=** power **+** RFWIN**->**power**;**  **}**  **}** |

# DisplayEnergy函数

* 显示RegFU的详细结果，包括面积，峰值功耗，漏电功耗和动态功耗

|  |
| --- |
| void RegFU**::**displayEnergy**(**uint32\_t indent**,**int plevel**,**bool is\_tdp**)**  **{**  **if** **(!**exist**)** **return;**  **if** **(**is\_tdp**)**  **{**  /\*  Integer RF::  Floating Point RF:    Area = IRF->area.get\_area()\*1e-6  Peak Dynamic = IRF->power.readOp.dynamic\*clockRate  Subthreshold Leakage = (long\_channel?  IRF->power.readOp.longer\_channel\_leakage:IRF->power.readOp.leakage)  Subthreshold Leakage with power gating = (long\_channel?  IRF->power.readOp.power\_gated\_with\_long\_channel\_leakage :  IRF->power.readOp.power\_gated\_leakage)  Runtime Dynamic = IRF->rt\_power.readOp.dynamic/executionTime  \*/  **if** **(**coredynp**.**regWindowing**)**  **{**  //Register Windows:RFWIN  **}**  **}**  **else**  **{**  //Peak Dynamic = IRF->rt\_power.readOp.dynamic\*clockRate  //Subthreshold Leakage = IRF->rt\_power.readOp.leakage  //Gate Leakage = IRF->rt\_power.readOp.gate\_leakage  **}**  **}** |