**目录**

[1. Processor.hh 2](#_Toc531799787)

* [简介 2](#_Toc531799788)
* [processor的属性值 2](#_Toc531799789)
* [Processor的方法 3](#_Toc531799790)
* [其它 3](#_Toc531799791)

[2. Processor.cc 3](#_Toc531799792)

* [设置参数函数set\_proc\_param() 3](#_Toc531799793)
* [构造函数Processor(ParseXML \*XML\_interface) 4](#_Toc531799794)
* [显示信息函数displayEnergy(uint32\_t indent = 0,int plevel = 100, bool is\_tdp=true) 9](#_Toc531799795)
* [其它 13](#_Toc531799796)

# Processor.hh

## 简介

定义了Processor类，继承于component。Processor包括了xml文件中所有的组件和参数，只被声明了一次，在main函数中

## processor的属性值

|  |
| --- |
| //所有属性都是公共属性  public**:**  //用于通过XML，获取解析得到的属性值  ParseXML **\***XML**;**  //处理器中的核心，可以是多个，因此使用vector  vector**<**Core **\*>** cores**;**  //L2 cache，可以为多个  vector**<**SharedCache **\*>** l2array**;**  //L3 cache，可以为多个  vector**<**SharedCache **\*>** l3array**;**  //L1 cache的目录  vector**<**SharedCache **\*>** l1dirarray**;**  //L2 cache的目录  vector**<**SharedCache **\*>** l2dirarray**;**  //片上网络，NoC，可以是多个  vector**<**NoC **\*>** nocs**;**  //存储控制器，MC  MemoryController **\*** mc**;**  //网卡，NIU，一个  NIUController **\*** niu**;**  //总线，PCIe  PCIeController **\*** pcie**;**  //闪存控制器，Flashc  FlashController **\*** flashcontroller**;**  //将部分XML中的参数放入另一个类中。  //该类是cacti中的一个方法，因此需要传递参数过去  InputParameter interface\_ip**;**  //处理器的一些简单参数和名称，例如  //int numCore, numL2, numL3, numNOC, numL1Dir, numL2Dir,numMC, numMCChannel;  //bool homoCore, homoL2, homoL3, homoNOC, homoL1Dir, homoL2Dir;  //double vdd;double power\_gating\_vcc;  ProcParam procdynp**;**  //处理器中可能包括的所有组件  //不论他们的个数多少，定义一个组件用于记录数据，例如功耗，面积  //使用core来记录所有cores的参数  Component core**,** l2**,** l3**,** l1dir**,** l2dir**,** noc**,** mcs**,** cc**,** nius**,** pcies**,**flashcontrollers**;**  //处理器的组件的个数  int numCore**,** numL2**,** numL3**,** numNOC**,** numL1Dir**,** numL2Dir**;** |

## Processor的方法

|  |
| --- |
| //构造函数，会初始化所有的属性值，包括其中的对象  //同时会进一步调用每个对象的方法，计算功耗和面积  //最后将结果放入到componets中  Processor**(**ParseXML **\***XML\_interface**);**  //没有实现这个函数，应该是所有计算过程都放入到了构造函数中  void compute**();**  //根据XML中解析的结果设置属性中的procdynp和interface\_ip中的属性  void set\_proc\_param**();**  //按照一定的格式化显示功耗，面积等结果，会显示所有组件的结果  void displayEnergy**(**uint32\_t indent **=** 0**,**int plevel **=** 100**,** bool is\_tdp**=true);**  //使用字符串显示设备种类  //ITRS high performance,ITRS low standby power  //ITRS low operating power,LP-DRAM,COMM-DRAM  void displayDeviceType**(**int device\_type\_**,** uint32\_t indent **=** 0**);**  //使用字符串显示互连关系  //aggressive interconnect,conservative interconnect  void displayInterconnectType**(**int interconnect\_type\_**,** uint32\_t indent **=** 0**);**  **~**Processor**();** |

## 其它

* Interface\_ip属性的功能是为了向mcpat中使用的另一个模型cacti提供必要的参数，包括初始化一些必要的工艺信息等之类的
* displayEnergy函数主要将所有计算好的结果打印出来。但是在processor中只会将每一类对象的总的信息打印出来，例如所有cores总体的功耗，面积信息。具体的内部信息，需要在每个对象中自己实现

# Processor.cc

## 设置参数函数set\_proc\_param()

* 设置procdynp参数，包括bool类型的判断类参数和一些具体参数

|  |
| --- |
| int numCore**,** numL2**,** numL3**,** numNOC**,** numL1Dir**,** numL2Dir**,**numMC**,** numMCChannel**;**  bool homoCore**,** homoL2**,** homoL3**,** homoNOC**,** homoL1Dir**,** homoL2Dir**;** |

* 使用XML解析到的信息，设置interface\_ip中的某些参数

|  |
| --- |
| /\*  使用XML中的信息设置interface\_ip中的某些参数  data\_arr\_ram\_cell\_tech\_type  data\_arr\_peri\_global\_tech\_type  tag\_arr\_ram\_cell\_tech\_type  tag\_arr\_peri\_global\_tech\_type  ic\_proj\_type  temp，F\_sz\_nm，F\_sz\_um  ver\_htree\_wires\_over\_array  broadcast\_addr\_din\_over\_ver\_htrees  power\_gating  \*/ |

* 设置interface\_ip中某些参数，固定，尽量不改动

|  |
| --- |
| /\*  设置一些interface\_ip固定，一般不修改的参数  specific\_hp\_vdd，specific\_lop\_vdd，specific\_lstp\_vdd，specific\_vcc\_min  delay\_wt，area\_wt，dynamic\_power\_wt，leakage\_power\_wt，cycle\_time\_wt  delay\_dev，area\_dev，dynamic\_power\_dev，leakage\_power\_dev，cycle\_time\_dev  ed，burst\_len，int\_prefetch\_w，page\_sz\_bits，F\_sz\_nm，F\_sz\_um  \*/ |

* 初始化一些interface\_ip中的参数，防止运行时出现错误，之后某些会被重新赋值

|  |
| --- |
| /\*  初始化一些interface\_ip的参数，防止运行出错，之后某些会被覆盖  cache\_sz，line\_sz，assoc，nbanks，out\_w，specific\_tag，tag\_w，access\_mode  obj\_func\_dyn\_energy，obj\_func\_dyn\_power，obj\_func\_leak\_power  obj\_func\_cycle\_t  is\_main\_mem，rpters\_in\_htree，ver\_htree\_wires\_over\_array  broadcast\_addr\_din\_over\_ver\_htrees，num\_rw\_ports，num\_rd\_ports  num\_wr\_ports，num\_se\_rd\_ports，num\_search\_ports  nuca，nuca\_bank\_count，is\_cache，pure\_ram，pure\_cam  force\_cache\_config，power\_gating  wt，wire\_is\_mat\_type，wire\_os\_mat\_type  force\_wiretype，print\_detail，add\_ecc\_b\_  \*/ |

## 构造函数Processor(ParseXML \*XML\_interface)

1. 初始化参数

|  |
| --- |
| //设置参数  set\_proc\_param**();**    //设置processor对象的一些基本信息参数  //numCore，numL2，numL3，numNOC，numL1Dir，numL2Dir  //私有cache的个数必须和核数一致  //同构的时候这些参数为1，计算功耗和面积的时候乘以实际的数量即可  **if** **(**procdynp**.**homoCore**)**  numCore **=** procdynp**.**numCore**==**0**?** 0**:**1**;**  **else**  numCore **=** procdynp**.**numCore**;** |

1. 计算cores的功耗和面积

|  |
| --- |
| /\*  inline void set\_pppm(double \* pppv, double a=1, double b=1, double c=1, double d=1)  {  pppv[0]= a;pppv[1]= b;pppv[2]= c;pppv[3]= d;  }  \*/  //遍历所有核，为每一个核生成一个Core对象，压入cores的vector中  //计算每一个核的功耗，面积，将结果放入core组件中  **for** **(**i **=** 0**;**i **<** numCore**;** i**++)**  **{**  //实例化一个Core对象，并放入cores的容器中  cores**.**push\_back**(new** Core**(**XML**,**i**,** **&**interface\_ip**));**  //计算新加入核的能耗  //computeEnergy(bool is\_tdp=true)默认tdp为true  //power和rt\_power都是cacti的component基类的属性  //这两个对象中包括着更细致的功耗分类  cores**[**i**]->**computeEnergy**();**//计算TDP, power  cores**[**i**]->**computeEnergy**(false);**//计算dyn\_power，rt\_power    //如果是同构核的面积计算方法  //同构核的情况下循环只会进行一遍  **if** **(**procdynp**.**homoCore**){**    //面积为核数乘以每个核的面积  core**.**area**.**set\_area**(**core**.**area**.**get\_area**()** **+** cores**[**i**]->**area**.**get\_area**()\***procdynp**.**numCore**);**  //将数据记录在processor的area属性中  area**.**set\_area**(**area**.**get\_area**()** **+** core**.**area**.**get\_area**());**    set\_pppm**(**pppm\_t**,**cores**[**i**]->**clockRate**\***procdynp**.**numCore**,** procdynp**.**numCore**,**procdynp**.**numCore**,**procdynp**.**numCore**);**  //power\*pppm\_t的含义：power对象中的前四个属性值分别和数组中的前四个数相乘  //power中的dynamic\*clockRate\*numCore  //power中计算的dynamic为时钟每一次反转而产生的功耗  core**.**power **=** core**.**power **+** cores**[**i**]->**power**\***pppm\_t**;**  //将数据记录在processor的属性中  power **=** power **+** core**.**power**;**    set\_pppm**(**pppm\_t**,**1**/**cores**[**i**]->**executionTime**,** procdynp**.**numCore**,**procdynp**.**numCore**,**procdynp**.**numCore**);**  //运行时动态功耗，之前的rt\_power中的dynamic应该记录的是整个运行时的能耗  core**.**rt\_power **=** core**.**rt\_power **+** cores**[**i**]->**rt\_power**\***pppm\_t**;**  rt\_power **=** rt\_power **+** core**.**rt\_power**;**  **}**  **else{**//异构核的处理  //计算面积  core**.**area**.**set\_area**(**core**.**area**.**get\_area**()** **+** cores**[**i**]->**area**.**get\_area**());**  area**.**set\_area**(**area**.**get\_area**()** **+** cores**[**i**]->**area**.**get\_area**());**    //基本一致，但是会为每个核都计算一遍  set\_pppm**(**pppm\_t**,**cores**[**i**]->**clockRate**,** 1**,** 1**,** 1**);**  core**.**power **=** core**.**power **+** cores**[**i**]->**power**\***pppm\_t**;**  power **=** power **+** cores**[**i**]->**power**\***pppm\_t**;**  set\_pppm**(**pppm\_t**,**1**/**cores**[**i**]->**executionTime**,** 1**,** 1**,** 1**);**  core**.**rt\_power **=** core**.**rt\_power **+** cores**[**i**]->**rt\_power**\***pppm\_t**;**  rt\_power **=** rt\_power **+** cores**[**i**]->**rt\_power**\***pppm\_t**;**  **}**  **}** |

1. 计算NoC的功耗和面积

|  |
| --- |
| //如果有NOC，NOC分为两种：一种是bus，另一种是routers  **if** **(**numNOC **>**0**)**  **{** //将所有的NOC都计算一遍，如果是同构的也只有一个  **for** **(**i **=** 0**;**i **<** numNOC**;** i**++)**  **{**  **if** **(**XML**->**sys**.**NoC**[**i**].**type**)**//如果NoC是router的结构  **{**  nocs**.**push\_back**(new** NoC**(**XML**,**i**,** **&**interface\_ip**,** 1**));**  **if** **(**procdynp**.**homoNOC**)**//同构的乘以个数即可  **{**  noc**.**area**.**set\_area**(**noc**.**area**.**get\_area**()** **+** nocs**[**i**]->**area**.**get\_area**()\***procdynp**.**numNOC**);**  area**.**set\_area**(**area**.**get\_area**()** **+** noc**.**area**.**get\_area**());**  **}**  **else**//异构的需要累加  **{**  noc**.**area**.**set\_area**(**noc**.**area**.**get\_area**()** **+** nocs**[**i**]->**area**.**get\_area**());**  area**.**set\_area**(**area**.**get\_area**()** **+** nocs**[**i**]->**area**.**get\_area**());**  **}**  **}**  **else**//NoC是bus的结构  **{**  //link\_len\_ = sqrt(area.get\_area()\*XML->sys.NoC[i].chip\_coverage)  //初始化NoC对象的时候，需要给出连接长度  //使用面积和参数中NoC覆盖的范围，估算连接长度  nocs**.**push\_back**(new** NoC**(**XML**,**i**,** **&**interface\_ip**,** 1**,** sqrt**(**area**.**get\_area**()\***XML**->**sys**.**NoC**[**i**].**chip\_coverage**)));**  **if** **(**procdynp**.**homoNOC**){**  //同构乘以个数  noc**.**area**.**set\_area**(**noc**.**area**.**get\_area**()** **+** nocs**[**i**]->**area**.**get\_area**()\***procdynp**.**numNOC**);**  area**.**set\_area**(**area**.**get\_area**()** **+** noc**.**area**.**get\_area**());**  **}**  **else**  **{** //异构的需要累加  noc**.**area**.**set\_area**(**noc**.**area**.**get\_area**()** **+** nocs**[**i**]->**area**.**get\_area**());**  area**.**set\_area**(**area**.**get\_area**()** **+** nocs**[**i**]->**area**.**get\_area**());**  **}**  **}**  **}**  //计算全局的每个NoC之间的连接关系，但是在此之前，整个的芯片面积必须已知  **for** **(**i **=** 0**;**i **<** numNOC**;** i**++)**  **{**  //如果有全局的连接关系，并且是router的NoC结构  **if** **(**nocs**[**i**]->**nocdynp**.**has\_global\_link **&&** XML**->**sys**.**NoC**[**i**].**type**)**  **{**  //使用芯片面积和每个router的覆盖面积估算全局的bus长度  //total\_nodes = nocdynp.horizontal\_nodes\*nocdynp.vertical\_nodes  //指的是NoC内部的节点个数  nocs**[**i**]->**init\_link\_bus**(**sqrt**(**area**.**get\_area**()\***XML**->**sys**.**NoC**[**i**].**chip\_coverage**));**  **if** **(**procdynp**.**homoNOC**)**//同构乘以个数  **{**  noc**.**area**.**set\_area**(**noc**.**area**.**get\_area**()** **+** nocs**[**i**]->**link\_bus\_tot\_per\_Router**.**area**.**get\_area**()**  **\*** nocs**[**i**]->**nocdynp**.**total\_nodes**\*** procdynp**.**numNOC**);**  area**.**set\_area**(**area**.**get\_area**()** **+** nocs**[**i**]->**link\_bus\_tot\_per\_Router**.**area**.**get\_area**()**  **\*** nocs**[**i**]->**nocdynp**.**total\_nodes**\*** procdynp**.**numNOC**);**  **}**  **else**  **{**  noc**.**area**.**set\_area**(**noc**.**area**.**get\_area**()** **+** nocs**[**i**]->**link\_bus\_tot\_per\_Router**.**area**.**get\_area**()**  **\*** nocs**[**i**]->**nocdynp**.**total\_nodes**);**  area**.**set\_area**(**area**.**get\_area**()** **+** nocs**[**i**]->**link\_bus\_tot\_per\_Router**.**area**.**get\_area**()**  **\*** nocs**[**i**]->**nocdynp**.**total\_nodes**);**  **}**  **}**  **}**  //计算功耗  **for** **(**i **=** 0**;**i **<** numNOC**;** i**++)**  **{**  nocs**[**i**]->**computeEnergy**();**  nocs**[**i**]->**computeEnergy**(false);**  **if** **(**procdynp**.**homoNOC**){**//同构的成倍数  set\_pppm**(**pppm\_t**,**procdynp**.**numNOC**\***nocs**[**i**]->**nocdynp**.**clockRate**,** procdynp**.**numNOC**,**procdynp**.**numNOC**,**procdynp**.**numNOC**);**  noc**.**power **=** noc**.**power **+** nocs**[**i**]->**power**\***pppm\_t**;**    set\_pppm**(**pppm\_t**,**1**/**nocs**[**i**]->**nocdynp**.**executionTime**,** procdynp**.**numNOC**,**procdynp**.**numNOC**,**procdynp**.**numNOC**);**  noc**.**rt\_power **=** noc**.**rt\_power **+** nocs**[**i**]->**rt\_power**\***pppm\_t**;**    power **=** power **+** noc**.**power**;**  rt\_power **=** rt\_power **+** noc**.**rt\_power**;**  **}**  **else**//异构累加  **{**  set\_pppm**(**pppm\_t**,**nocs**[**i**]->**nocdynp**.**clockRate**,** 1**,** 1**,** 1**);**  noc**.**power **=** noc**.**power **+** nocs**[**i**]->**power**\***pppm\_t**;**  power **=** power **+** nocs**[**i**]->**power**\***pppm\_t**;**    set\_pppm**(**pppm\_t**,**1**/**nocs**[**i**]->**nocdynp**.**executionTime**,** 1**,** 1**,** 1**);**  noc**.**rt\_power **=** noc**.**rt\_power **+** nocs**[**i**]->**rt\_power**\***pppm\_t**;**  rt\_power **=** rt\_power **+** nocs**[**i**]->**rt\_power**\***pppm\_t**;**  **}**  **}**  **}** |

1. 计算其它组件的功耗和面积

|  |
| --- |
| //如果不是私有的L2 cache  //如果是私有cache，则会包含在core中  **if** **(!**XML**->**sys**.**Private\_L2**)**  **{**  //如果有多个L2 cache。同构的情况下，numL2=1  **if** **(**numL2 **>**0**)**  **{**  **}**  **}**  //如果有L3 cache，一定是在核外，则计算  **if** **(**numL3 **>**0**)**  **{** //如果是异构的，则需要将每一个都计算一遍  **for** **(**i **=** 0**;**i **<** numL3**;** i**++)**  **{**  **}**  **}**  //如果有L1 directory，则计算  **if** **(**numL1Dir **>**0**)**  **{**  //计算所有的  **for** **(**i **=** 0**;**i **<** numL1Dir**;** i**++)**  **{**  **}**  **}**  **if** **(**numL2Dir **>**0**)**  **{**  **for** **(**i **=** 0**;**i **<** numL2Dir**;** i**++)**  **{**  **}**  **}**  //如果youMC，则计算  **if** **(**XML**->**sys**.**mc**.**number\_mcs **>**0 **&&** XML**->**sys**.**mc**.**memory\_channels\_per\_mc**>**0**)**  **{**  **}**  //如果有flashc  **if** **(**XML**->**sys**.**flashc**.**number\_mcs **>**0 **)**//flash controller  **{**  **}**  //如果有NIU  **if** **(**XML**->**sys**.**niu**.**number\_units **>**0**)**  **{**  **}**  //如果有PICe  **if** **(**XML**->**sys**.**pcie**.**number\_units **>**0 **&&** XML**->**sys**.**pcie**.**num\_channels **>**0**)**  **{**  **}** |

## 显示信息函数displayEnergy(uint32\_t indent = 0,int plevel = 100, bool is\_tdp=true)

|  |
| --- |
| void Processor**::**displayEnergy**(**uint32\_t indent**,** int plevel**,** bool is\_tdp**=true)**  **{**  bool long\_channel **=** XML**->**sys**.**longer\_channel\_device**;**//一般为false  bool power\_gating **=** XML**->**sys**.**power\_gating**;**//一般为true  **if** **(**is\_tdp**)**  **{**  //输出信息：  //Technology  //whether Using Long Channel Devices When Appropriate  //Core clock Rate(MHz)    //输出Processor的整体信息  //对于是否为long\_channel会影响是leakage还是longer\_channel\_leakage      //Area =  //面积的计算：area.get\_area()\*1e-6 （m^2 -> mm^2^2）    //Peak Power =  //峰值功率的计算：power.readOp.dynamic+power.readOp.gate\_leakage  // +power.readOp.leakage/power.readOp.longer\_channel\_leakage    //Total Leakage =  //整体泄露功率的计算：power.readOp.gate\_leakage  // +power.readOp.leakage/power.readOp.longer\_channel\_leakage      //Peak Dynamic =  //峰值动态功耗：power.readOp.dynamic    //Subthreshold Leakage =  //亚阈值泄漏功率：power.readOp.leakage/power.readOp.longer\_channel\_leakage    //Subthreshold Leakage with power gating = (power\_gating？)  //有功率门控的亚阈值泄漏功率：power.readOp.power\_gated\_with\_long\_channel\_leakage  // /power.readOp.power\_gated\_leakage      //Gate Leakage =  //栅漏功率：power.readOp.gate\_leakage  //Runtime Dynamic =  //运行时动态功率：rt\_power.readOp.dynamic    //有核的情况，显示所有的在一起的信息  **if** **(**numCore **>**0**){**  //core.power  //Total Cores:  //...  **}**  //非私有的L2 cache  **if** **(!**XML**->**sys**.**Private\_L2**)**  **{**  **if** **(**numL2 **>**0**){**  //l2.power  //Total L2s:  //...  **}**  **}**  **if** **(**numL3 **>**0**){**  //l3.power  //Total L3s:  //...  **}**  **if** **(**numL1Dir **>**0**){**  //l1dir.power  //Total First Level Directory:  //device type  //...  **}**  **if** **(**numL2Dir **>**0**){**  //l2dir.power  //Total Second Level Directory:  //device type  //...  **}**  **if** **(**numNOC **>**0**){**  //noc.power  //Total NoCs (Network/Bus):  //device type  //...  **}**  **if** **(**XML**->**sys**.**mc**.**number\_mcs **>**0 **&&** XML**->**sys**.**mc**.**memory\_channels\_per\_mc**>**0**)**  **{**  //mcs.power  //Total MCs:  //device type  //...  **}**  **if** **(**XML**->**sys**.**flashc**.**number\_mcs **>**0**)**  **{**  //flashcontrollers.power  //Total Flash/SSD Controllers:  //device type  //...  **}**  **if** **(**XML**->**sys**.**niu**.**number\_units **>**0 **)**  **{**  //nius.power  //Total NIUs:  //device type  //...  **}**  **if** **(**XML**->**sys**.**pcie**.**number\_units **>**0 **&&** XML**->**sys**.**pcie**.**num\_channels**>**0**)**  **{**  //pcies.power  //Total PCIes:  //device type  //...  **}**  cout **<<**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"**<<**endl**;**  **if** **(**plevel **>**1**)**//输出每个组件内部的具体信息  **{**  **for** **(**i **=** 0**;**i **<** numCore**;** i**++)**  cores**[**i**]->**displayEnergy**(**indent**+**4**,**plevel**,**is\_tdp**);**    **if** **(!**XML**->**sys**.**Private\_L2**)**  **for** **(**i **=** 0**;**i **<** numL2**;** i**++)**  l2array**[**i**]->**displayEnergy**(**indent**+**4**,**is\_tdp**);**    **for** **(**i **=** 0**;**i **<** numL3**;** i**++)**  l3array**[**i**]->**displayEnergy**(**indent**+**4**,**is\_tdp**);**  **for** **(**i **=** 0**;**i **<** numL1Dir**;** i**++)**  l1dirarray**[**i**]->**displayEnergy**(**indent**+**4**,**is\_tdp**);**  **for** **(**i **=** 0**;**i **<** numL2Dir**;** i**++)**  l2dirarray**[**i**]->**displayEnergy**(**indent**+**4**,**is\_tdp**);**  **if** **(**XML**->**sys**.**mc**.**number\_mcs **>**0 **&&** XML**->**sys**.**mc**.**memory\_channels\_per\_mc**>**0**)**  mc**->**displayEnergy**(**indent**+**4**,**is\_tdp**);**  **if** **(**XML**->**sys**.**flashc**.**number\_mcs **>**0 **&&** XML**->**sys**.**flashc**.**memory\_channels\_per\_mc**>**0**)**  flashcontroller**->**displayEnergy**(**indent**+**4**,**is\_tdp**);**  **if** **(**XML**->**sys**.**niu**.**number\_units **>**0 **)**  niu**->**displayEnergy**(**indent**+**4**,**is\_tdp**);**    **if** **(**XML**->**sys**.**pcie**.**number\_units **>**0 **&&** XML**->**sys**.**pcie**.**num\_channels**>**0**)**  pcie**->**displayEnergy**(**indent**+**4**,**is\_tdp**);**  **for** **(**i **=** 0**;**i **<** numNOC**;** i**++)**  nocs**[**i**]->**displayEnergy**(**indent**+**4**,**plevel**,**is\_tdp**);**  **}**  **}**  **}** |

## 其它

1. 对于同构核，在计算面积和功耗的过程中，只需要计算一个组件的面积和功耗，然后乘以组件的个数即可
2. set\_pppm(double \* pppv, double a=1, double b=1, double c=1, double d=1)。该函数仅仅只是将参数中的后四个放入第一个数组参数中
3. cores[i]->power\*pppm\_t。对于这个语句，含义是将power对象中的前四个属性值，分别乘以pppm\_t数组中的前四个元素。Power的前四个属性值为

|  |
| --- |
| double dynamic**;**  double leakage**;**  double gate\_leakage**;**  double short\_circuit**;** |

1. 对于异构核或者组件的面积和功耗的计算，需要遍历每一个组件，分别计算得到结果，然后累加放在processor中定义的组件中，以便之后的显示
2. ComputerEnergy(bool is\_tdp=true)。该函数会被调用两次，第一个tdp=true，计算的是组件的power；第二次则是tdp设置为false，计算的是组件的rt\_power
3. 通过computerEnergy计算的power中的dynamic属性值代表的是组件在时钟翻转一次时消耗的能量，因此需要乘上频率，才能得到功耗。峰值动态功耗即代表每个时钟沿都翻转了一次
4. 通过computerEnergy计算的rt\_power中的dynamic属性代表着组件在整个运行过程中消耗的能量，因此需要除以执行时间，才能够得到运行过程中的功耗
5. L2 cache如果被设置为私有的，则会被包含在core对象中，面积和功耗的计算也都在core中进行。如果L2是共享的，则会在processor中计算
6. 对于NoC而言，有两种类型，一种是Bus，另一种则是router的形式。两种的面积计算方式不同，但是功耗计算相同
7. 对于Bus的面积计算，需要通过芯片面积和NoC的覆盖范围估计出连接线路的长度（在初始化NoC对象的时候指令），然后进行计算
8. 对于Router的面积计算，一方面需要计算每个router的面积（不需要估计连接线路长度），另一方面，还需要使用芯片面积和每个router的覆盖范围，以及router中节点的个数来估计router内部节点之间的连接线路的长度和面积，两者合并之后才是真正的面积
9. 对于Router的节点个数total\_nodes = nocdynp.horizontal\_nodes\*nocdynp.vertical\_nodes