# Selection\_logic类声明

|  |
| --- |
| //选择逻辑，主要在指令发射阶段，从指令队列中选择合适的指令发射  class selection\_logic **:** public Component**{**  public**:**  //构造函数  //参数：  /\*  is\_default = \_is\_default， 没有用到  win\_entries = win\_entries\_，指令窗口的表项数  issue\_width = issue\_width\_，发射宽度，每次发射几条指令  l\_ip = \*configure\_interface，用于传递到cacti的参数  device\_ty = device\_ty\_，Core\_device，核心类型  core\_ty = core\_ty\_，inorder/OOO  \*/  selection\_logic**(**bool \_is\_default**,** int win\_entries\_**,**  int issue\_width\_**,** const InputParameter **\***configure\_interface**,**  enum Device\_ty device\_ty\_**=**Core\_device**,**  enum Core\_type core\_ty\_**=**Inorder**);**  //没有被使用到  bool is\_default**;**  //传递给cacti使用  InputParameter l\_ip**;**  //  uca\_org\_t local\_result**;**  //没有使用到，也没有被初始化  const ParseXML **\***XML\_interface**;**  //指令窗口的表项数  int win\_entries**;**  //发射宽度，每次发射几条指令  int issue\_width**;**  //线程数  int num\_threads**;**  //设备类型，core  enum Device\_ty device\_ty**;**  //核的类型，inorder/ooo  enum Core\_type core\_ty**;**  //计算功率  void selection\_power**();**  void leakage\_feedback**(**double temperature**);** // TODO  **};** |

# 定义selection\_logic的对象

* 该类主要用于计算由于在发射阶段从指令窗口中选择就绪指令所带来的功率
* 按序和乱序会有不同的选择逻辑，因此在SechdulerU中会定义两次（非同时）
* 在按序单发射多线程处理器中，发射宽度为1\*线程数，因为按序的情况不需要选择
* 按序情况下并且保证是多线程的情况下的定义（如果不是多线程，则根本不需要选择）

|  |
| --- |
| //重新初始化，以防止出现其它额外的问题  interface\_ip**.**assoc **=** 1**;**  /\*  is\_default\_=true  numIQEntries表项数  win\_entries\_=XML->sys.core[ithCore].instruction\_window\_size  按序情况下，发射宽度为1  issue\_width\_=coredynp.peak\_issueW\*XML->sys.core[ithCore].number\_hardware\_threads  configure\_interface=&interface\_ip  device\_ty\_=Core\_device  core\_ty\_=coredynp.core\_ty  \*/  instruction\_selection **=** **new** selection\_logic**(**is\_default**,**  XML**->**sys**.**core**[**ithCore**].**instruction\_window\_size**,**  coredynp**.**peak\_issueW**\***XML**->**sys**.**core**[**ithCore**].**number\_hardware\_threads**,**  **&**interface\_ip**,** Core\_device**,** coredynp**.**core\_ty**);** |

* 乱序情况下的定义

|  |
| --- |
| /\*  is\_default\_=true  numIQEntries表项数  win\_entries\_=XML->sys.core[ithCore].instruction\_window\_size  按序情况下，发射宽度为1  issue\_width\_=coredynp.peak\_issueW  configure\_interface=&interface\_ip  device\_ty\_=Core\_device  core\_ty\_=coredynp.core\_ty  \*/  instruction\_selection **=** **new** selection\_logic**(**is\_default**,**  XML**->**sys**.**core**[**ithCore**].**instruction\_window\_size**,**  coredynp**.**peak\_issueW**,** **&**interface\_ip**,** Core\_device**,** coredynp**.**core\_ty**);** |

* 两者的不同：

1. 按序的情况下，代码中给的前提条件是按序并且是多线程的情况下才会使用到选择逻辑，并且此时的选择的指令数为发射宽度\*线程数
2. 乱序的情况下，不需要一定是多线程。其次此时的选择的指令数仅为发射宽度，而没有线程数参与，其余参数一致

* 输入参数的统计

|  |  |
| --- | --- |
| 参数 | 说明 |
| bool is\_default | True, selection\_logic未用到 |
| int win\_entries | 指令窗口的表项数，设置为stats中的numIQEntries  XML->sys.core[ithCore].instruction\_window\_size |
| int issue\_width | 发射宽度（按序单线程的情况下不需要选择逻辑）  按序情况下：coredynp.peak\_issueW\* XML->sys.core[ithCore].number\_hardware\_threads  乱序情况下：coredynp.peak\_issueW |
| InputParameter \*  configure\_interface | 用于初始化uca\_org\_t对象  &interface\_ip |
| enum Device\_ty  device\_ty | Core\_device，设备类型为core |
| enum Core\_type  core\_ty | Coredynp.core\_ty，按序或者乱序 |

# Selection\_logic的构造函数

|  |
| --- |
| //selection\_logic  selection\_logic**::**selection\_logic**(**bool \_is\_default**,**  int win\_entries\_**,**int issue\_width\_**,**  const InputParameter **\***configure\_interface**,**  enum Device\_ty device\_ty\_**,**enum Core\_type core\_ty\_**)**  **:**is\_default**(**\_is\_default**),**win\_entries**(**win\_entries\_**),**  issue\_width**(**issue\_width\_**),**device\_ty**(**device\_ty\_**),**  core\_ty**(**core\_ty\_**)**  **{**  //参数初始化  l\_ip**=\***configure\_interface**;**  local\_result **=** init\_interface**(&**l\_ip**);**    //计算功耗  selection\_power**();**    //和工艺尺寸相关，可以看作比例系数  double sckRation **=** g\_tp**.**sckt\_co\_eff**;**  power**.**readOp**.**dynamic **\*=** sckRation**;**  power**.**writeOp**.**dynamic **\*=** sckRation**;**  power**.**searchOp**.**dynamic **\*=** sckRation**;**  double long\_channel\_device\_reduction **=**  longer\_channel\_device\_reduction**(**device\_ty**,**core\_ty**);**  power**.**readOp**.**longer\_channel\_leakage **=**  power**.**readOp**.**leakage**\***long\_channel\_device\_reduction**;**  double pg\_reduction **=** power\_gating\_leakage\_reduction**(false);**  power**.**readOp**.**power\_gated\_leakage **=**  power**.**readOp**.**leakage**\***pg\_reduction**;**  power**.**readOp**.**power\_gated\_with\_long\_channel\_leakage **=**  power**.**readOp**.**power\_gated\_leakage **\*** long\_channel\_device\_reduction**;**  **}** |

# selection\_power()函数

* 已经到电路级别的计算，不介绍