interconnect

[1. interconnect类声明 2](#_Toc532325353)

[2. 定义interconnect的对象 5](#_Toc532325354)

[（1） 在定义interconnect之前需要指定的参数 5](#_Toc532325355)

[（2） Interconnect的输入参数中的固定值（相同） 5](#_Toc532325356)

[（3） Interconnect的输入参数之一data\_w 6](#_Toc532325357)

[（4） Interconnect的输入参数之一len 7](#_Toc532325358)

[（5） Len中用到的高度的具体值 8](#_Toc532325359)

[3. interconnect的构造函数 10](#_Toc532325360)

# interconnect类声明

|  |
| --- |
| //主要用于核内的bypass网络  class interconnect **:** public Component  **{**  public**:**  interconnect**(**  string name\_**,**  enum Device\_ty device\_ty\_**,**//Core\_device  double base\_w**,** double base\_h**,** //1,1 未用到  int data\_w**,** double len**,**//数据宽度和线长  const InputParameter **\***configure\_interface**,**//mcpat和cacti之间的参数  int start\_wiring\_level\_**,**//3，没有用到    //可流水化的wire：bus，不关心延迟，只关心吞吐量  //非流水化的wire：bypass，关心延迟  bool pipelinable\_ **=** **false,**//是否可以流水化    //覆盖率  double route\_over\_perc\_ **=**0.5**,**  //是否优化  bool opt\_local\_**=true,**  //核心的类型  enum Core\_type core\_ty\_**=**Inorder**,**  //线类型  // Global /\* gloabl wires with repeaters \*/,  // Global\_5 /\* 5% delay penalty \*/,  // Global\_10 /\* 10% delay penalty \*/,  // Global\_20 /\* 20% delay penalty \*/,  // Global\_30 /\* 30% delay penalty \*/,  // Low\_swing /\* differential low power wires with high area overhead \*/,  // Semi\_global /\* mid-level wires with repeaters\*/,  // Transmission /\* tranmission lines with high area overhead \*/,  // Optical /\* optical wires \*/,  // Invalid\_wtype  enum Wire\_type wire\_model**=**Global**,**  //都采用了默认值  double width\_s**=**1.0**,** double space\_s**=**1.0**,**  TechnologyParameter**::**DeviceType **\***dt **=** **&(**g\_tp**.**peri\_global**)**  **);**  **~**interconnect**()** **{};**  void compute**();**    string name**;**  enum Device\_ty device\_ty**;**  InputParameter l\_ip**;**  uca\_org\_t local\_result**;**  Area no\_device\_under\_wire\_area**;**    void set\_in\_rise\_time**(**double rt**)**  **{**  in\_rise\_time **=** rt**;**  **}**  //该函数未被使用  void leakage\_feedback**(**double temperature**);**    powerDef power\_bit**;**  //这些参数未被使用  double wire\_bw**;**  double init\_wire\_bw**;** // bus width at root  double base\_width**;**  double base\_height**;**  int start\_wiring\_level**;**  double max\_unpipelined\_link\_delay**;**  double in\_rise\_time**,** out\_rise\_time**;**    //数据宽度和线长  int data\_width**;**  double length**;**  enum Wire\_type wt**;**  double width\_scaling**,** space\_scaling**;**  double interconnect\_latency**;**  double interconnect\_throughput**;**    double min\_w\_nmos**;**  double min\_w\_pmos**;**  //latency = l\_ip.latency;  //throughput = l\_ip.throughput;  double latency**,** throughput**;**  bool latency\_overflow**;**  bool throughput\_overflow**;**    bool opt\_local**;**  enum Core\_type core\_ty**;**  bool pipelinable**;**  double route\_over\_perc**;**  //当可流水化的时候才有用  //num\_pipe\_stages = (int)ceil(delay/throughput);  int num\_pipe\_stages**;**  private**:**  TechnologyParameter**::**DeviceType **\***deviceType**;**  **};** |

# 定义interconnect的对象

* 该类定义了处理器中功能部件和寄存器堆，LSQ，ROB，指令窗口之间的bypass逻辑
* 在EXEU中定义了该类的六个对象，（数据，tag）\*（ALU，FPU，MUL）

## 在定义interconnect之前需要指定的参数

|  |
| --- |
| **if** **(**XML**->**sys**.**Embedded**)**  **{**  //Global\_30 /\* 30% delay penalty  interface\_ip**.**wt **=**Global\_30**;**  //wire\_inside\_mat\_type  //mat内部的连线类型：global(2), local(0), else(1)  interface\_ip**.**wire\_is\_mat\_type **=** 0**;**  //wire\_outside\_mat\_type  //mat外部的连线类型，global(2), else(1)  interface\_ip**.**wire\_os\_mat\_type **=** 0**;**  interface\_ip**.**throughput **=** 1.0**/**clockRate**;**  interface\_ip**.**latency **=** 1.0**/**clockRate**;**  **}**  **else**  **{**  //gloabl wires with repeaters  interface\_ip**.**wt **=**Global**;**  interface\_ip**.**wire\_is\_mat\_type **=** 2**;**  //start from semi-global since local wires are already used  interface\_ip**.**wire\_os\_mat\_type **=** 2**;**  interface\_ip**.**throughput **=** 10.0**/**clockRate**;** //Do not care  interface\_ip**.**latency **=** 10.0**/**clockRate**;**  **}** |

## Interconnect的输入参数中的固定值（相同）

|  |  |
| --- | --- |
| 参数名 | 值 |
| enum Device\_ty device\_ty\_ | Core\_device |
| double base\_w | 1（没有使用到） |
| double base\_h | 1（没有使用到） |
| InputParameter \*configure\_interface | &interface\_ip |
| int start\_wiring\_level | 3（没有使用到） |
| bool pipelinable\_ | False（不可流水化的bypass，一个周期）  可流水化的wire：bus，不关心延迟，只关心吞吐量  非流水化的wire：bypass，关心延迟 |
| double route\_over\_perc\_ | 1.0 |
| bool opt\_local\_ | coredynp.opt\_local |
| enum Core\_type core\_ty\_ | coredynp.core\_ty |
| 其余除了data\_w和len | 默认值 |

## Interconnect的输入参数之一data\_w

|  |  |
| --- | --- |
| 对象名 | Len的值 |
| 按序情况下的bypass网络定义 | |
| int\_bypass | int(ceil(XML->sys.machine\_bits/32.0)\*32)  **解释：机器宽度，32的倍数，bits单位** |
| intTagBypass | coredynp.perThreadState  **解释：定义为8 bit，每个线程的状态** |
| int\_mul\_bypass | int(ceil(XML->sys.machine\_bits/32.0)\*32\*1.5)  **解释：机器宽度\*1.5，bits（不太懂为什么\*1.5）** |
| intTag\_mul\_Bypass | coredynp.perThreadState  **解释：定义为8 bit，每个线程的状态** |
| fp\_bypass | int(ceil(XML->sys.machine\_bits/32.0)\*32\*1.5)  **解释：机器宽度\*1.5，bits（不太懂为什么\*1.5）** |
| fpTagBypass | coredynp.perThreadState  **解释：定义为8 bit，每个线程的状态** |
|  | |
| 乱序+PRF情况下的bypass定义 | |
| int\_bypass | int(ceil(coredynp.int\_data\_width))  **解释：整数数据宽度（机器宽度一致）** |
| intTagBypass | coredynp.phy\_ireg\_width  **解释：索引定点物理寄存器位宽 bits** |
| int\_mul\_bypass | int(ceil(coredynp.int\_data\_width))  **解释：整数数据宽度（机器宽度一致）** |
| intTag\_mul\_Bypass | coredynp.phy\_ireg\_width  **解释：索引定点物理寄存器位宽 bits** |
| fp\_bypass | int(ceil(coredynp.fp\_data\_width))  **解释：浮点数据宽度（机器宽度一致）** |
| fpTagBypass | coredynp.phy\_freg\_width  **解释：索引浮点物理寄存器位宽 bits** |
|  | |
| 乱序+PRF情况下的bypass定义 | |
| int\_bypass | int(ceil(coredynp.int\_data\_width))  **解释：整数数据宽度（机器宽度一致）** |
| intTagBypass | coredynp.phy\_ireg\_width  **解释：索引定点物理寄存器位宽 bits** |
| int\_mul\_bypass | int(ceil(coredynp.int\_data\_width))  **解释：整数数据宽度（机器宽度一致）** |
| intTag\_mul\_Bypass | coredynp.phy\_ireg\_width  **解释：索引定点物理寄存器位宽 bits** |
| fp\_bypass | int(ceil(coredynp.fp\_data\_width))  **解释：浮点数据宽度（机器宽度一致）** |
| fpTagBypass | coredynp.phy\_freg\_width  **解释：索引浮点物理寄存器位宽 bits** |

## Interconnect的输入参数之一len

|  |  |
| --- | --- |
| 对象名 | Len的值 |
| 按序情况下的bypass网络定义 | |
| int\_bypass | rfu->int\_regfile\_height + exeu->FU\_height + lsq\_height  **（数据传递：IRF, ALU, LSQ） LSQ的地址计算** |
| intTagBypass | rfu->int\_regfile\_height + exeu->FU\_height + lsq\_height + scheu->Iw\_height  **（TAG传递：IRF, ALU, LSQ, IW）** |
| int\_mul\_bypass | rfu->fp\_regfile\_height + exeu->FU\_height + mul->FU\_height + lsq\_height  **（数据传递：FRF, ALU, MUL, LSQ）** |
| intTag\_mul\_Bypass | rfu->fp\_regfile\_height + exeu->FU\_height + mul->FU\_height + lsq\_height + scheu->Iw\_height  **（TAG传递：FRF, ALU, MUL, LSQ, IW）** |
| fp\_bypass | rfu->fp\_regfile\_height + fp\_u->FU\_height  **（数据传递：FRF, FPU）** |
| fpTagBypass | rfu->fp\_regfile\_height + fp\_u->FU\_height + lsq\_height + scheu->Iw\_height  **（TAG传递：FRF, MUL, LSQ, IW）** |
|  | |
| 乱序+PRF情况下的bypass定义 | |
| int\_bypass | rfu->int\_regfile\_height + exeu->FU\_height + lsq\_height  **（数据传递：IRF, ALU, LSQ）** |
| intTagBypass | rfu->int\_regfile\_height + exeu->FU\_height + lsq\_height + scheu->Iw\_height + scheu->ROB\_height  **（TAG传递：IRF, ALU, LSQ, IW, ROB）** |
| int\_mul\_bypass | rfu->int\_regfile\_height + exeu->FU\_height + mul->FU\_height + lsq\_height  **（数据传递：IRF, ALU, MUL, LSQ）** |
| intTag\_mul\_Bypass | rfu->int\_regfile\_height + exeu->FU\_height + mul->FU\_height + lsq\_height + scheu->Iw\_height + scheu->ROB\_height  **（TAG传递：IRF, ALU, MUL, LSQ, IW, ROB）** |
| fp\_bypass | rfu->fp\_regfile\_height + fp\_u->FU\_height  **（数据传递：FRF, FPU）** |
| fpTagBypass | rfu->fp\_regfile\_height + fp\_u->FU\_height + lsq\_height + scheu->fp\_Iw\_height + scheu->ROB\_height  **（TAG传递：FRF,FPU, LSQ, IW, ROB）** |
|  | |
| 乱序+PRF情况下的bypass定义 | |
| int\_bypass | rfu->int\_regfile\_height + exeu->FU\_height + lsq\_height + scheu->Iw\_height + scheu->ROB\_height  **（数据传递：IRF, ALU, LSQ, IW, ROB）** |
| intTagBypass | rfu->int\_regfile\_height + exeu->FU\_height + lsq\_height + scheu->Iw\_height + scheu->ROB\_height  **（TAG传递：IRF, ALU, LSQ, IW, ROB）** |
| int\_mul\_bypass | rfu->int\_regfile\_height + exeu->FU\_height + mul->FU\_height + lsq\_height + scheu->Iw\_height + scheu->ROB\_height  **（数据传递：IRF, ALU, MUL, LSQ, IW, ROB）** |
| intTag\_mul\_Bypass | rfu->int\_regfile\_height + exeu->FU\_height + mul->FU\_height + lsq\_height + scheu->Iw\_height + scheu->ROB\_height  **（TAG传递：IRF, ALU, MUL, LSQ, IW, ROB）** |
| fp\_bypass | rfu->fp\_regfile\_height + fp\_u->FU\_height + lsq\_height + scheu->fp\_Iw\_height + scheu->ROB\_height  **（数据传递：FRF, FPU, LSQ, IW, ROB）** |
| fpTagBypass | rfu->fp\_regfile\_height + fp\_u->FU\_height + lsq\_height + scheu->fp\_Iw\_height + scheu->ROB\_height  **（TAG传递：FRF, FPU, LSQ, IW, ROB）** |

## Len中用到的高度的具体值

|  |  |
| --- | --- |
| 对象名 | 值 |
| local\_result.cache\_ht对于ArrayST对象而言会被自动计算出来  cdb\_overhead常量为1.1 | |
| rfu->int\_regfile\_height | IRF->local\_result.cache\_ht\* XML->sys.core[ithCore].number\_hardware\_threads\* sqrt(cdb\_overhead)  **相关因素：cache\_ht，cdb\_overheard，硬件线程数** |
| rfu->fp\_regfile\_height | FRF->local\_result.cache\_ht\* XML->sys.core[ithCore].number\_hardware\_threads\* sqrt(cdb\_overhead)  **相关因素**：cache\_ht，cdb\_overheard，硬件线程数 |
| exeu->FU\_height  mul->FU\_height  fp\_u->FU\_height | FU\_height=(18667\*num\_fu)\*interface\_ip.F\_sz\_um  **相关因素**：**功能单元个数** |
| lsq\_height | 按序情况下，没有LQ  LSQ->local\_result.cache\_ht\*sqrt(cdb\_overhead)  乱序情况下，有LQ  (LSQ->local\_result.cache\_ht + LoadQ->local\_result.cache\_ht)\* sqrt(cdb\_overhead)  **相关因素**：**LSQ和LQ的cache\_ht，cdb\_overheard** |
| scheu->Iw\_height | int\_inst\_window->local\_result.cache\_ht  **相关因素**：**cache\_ht** |
| scheu->fp\_Iw\_height | fp\_inst\_window->local\_result.cache\_ht  **相关因素**：**cache\_ht** |
| scheu->ROB\_height | ROB->local\_result.cache\_ht  **相关因素**：**cache\_ht** |

# interconnect的构造函数

|  |
| --- |
| //额外使用的参数  //l\_ip.latency， l\_ip.throughput  interconnect**::**interconnect**(**  string name\_**,**  enum Device\_ty device\_ty\_**,**  double base\_w**,** double base\_h**,**  int data\_w**,** double len**,**  const InputParameter **\***configure\_interface**,**  int start\_wiring\_level\_**,**  bool pipelinable\_ **,**  double route\_over\_perc\_ **,**  bool opt\_local\_**,**  enum Core\_type core\_ty\_**,**  enum Wire\_type wire\_model**,**  double width\_s**,** double space\_s**,**  TechnologyParameter**::**DeviceType **\***dt  **)**  **:**name**(**name\_**),**  device\_ty**(**device\_ty\_**),**  in\_rise\_time**(**0**),**//未用到  out\_rise\_time**(**0**),**//未用到  base\_width**(**base\_w**),**//未用到  base\_height**(**base\_h**),**//未用到  data\_width**(**data\_w**),**//数据宽度  wt**(**wire\_model**),**//默认值  width\_scaling**(**width\_s**),**//默认值  space\_scaling**(**space\_s**),**//默认值  start\_wiring\_level**(**start\_wiring\_level\_**),**//未用到  length**(**len**),**//线长  opt\_local**(**opt\_local\_**),**  core\_ty**(**core\_ty\_**),**  pipelinable**(**pipelinable\_**),**//是否可以流水化  route\_over\_perc**(**route\_over\_perc\_**),**//覆盖率  deviceType**(**dt**)**//默认值  **{**  wt **=** Global**;**  l\_ip**=\***configure\_interface**;**  local\_result **=** init\_interface**(&**l\_ip**);**  max\_unpipelined\_link\_delay **=** 0**;** //TODO  min\_w\_nmos **=** g\_tp**.**min\_w\_nmos\_**;**  min\_w\_pmos **=** deviceType**->**n\_to\_p\_eff\_curr\_drv\_ratio **\*** min\_w\_nmos**;**  //给出延迟和吞吐量的要求，之后判断是否符合标准，  //或者是否应该增加流水级  latency **=** l\_ip**.**latency**;**  throughput **=** l\_ip**.**throughput**;**  //延迟是否未满足  latency\_overflow**=false;**  //吞吐量是否未满足，未用到  throughput\_overflow**=false;**  //不可流水化，为了关心延迟，类似于bypass逻辑  **if** **(**pipelinable **==** **false)**  **{**  //计算功耗  compute**();**  //判断是否优化时钟并且优化时序  **if** **(**opt\_for\_clk **&&** opt\_local**)**  **{**  //如果实际的延迟大于要求的延迟，  //则需要将线延迟扩宽，此时面积也会增加  **while** **(**delay **>** latency **&&** width\_scaling**<**3.0**)**  **{**  width\_scaling **\*=** 2**;**  space\_scaling **\*=** 2**;**  Wire winit**(**width\_scaling**,** space\_scaling**);**  compute**();**  **}**  **if** **(**delay **>** latency**)**  **{**  latency\_overflow**=true;**  **}**  **}**  **}**  //可流水化，例如bus，主要关心吞吐量而不是延迟  **else**  **{**  compute**();**  **if** **(**opt\_for\_clk **&&** opt\_local**)**  **{**  **while** **(**delay **>** throughput **&&** width\_scaling**<**3.0**)**  **{**  width\_scaling **\*=** 2**;**  space\_scaling **\*=** 2**;**  Wire winit**(**width\_scaling**,** space\_scaling**);**  compute**();**  **}**  //如果延迟太大，则需要插入流水级，以保证吞吐量  **if** **(**delay **>** throughput**)**  **{**  num\_pipe\_stages **=** **(**int**)**ceil**(**delay**/**throughput**);**  assert**(**num\_pipe\_stages**>**0**);**  delay **=** delay**/**num\_pipe\_stages **+** num\_pipe\_stages**\***0.05**\***delay**;**  **}**  **}**  **}**  //单根线的功耗，需要乘以线宽  power\_bit **=** power**;**  power**.**readOp**.**dynamic **\*=** data\_width**;**  power**.**readOp**.**leakage **\*=** data\_width**;**  power**.**readOp**.**gate\_leakage **\*=** data\_width**;**  area**.**set\_area**(**area**.**get\_area**()\***data\_width**);**  no\_device\_under\_wire\_area**.**h **\*=** data\_width**;**  **if** **(**latency\_overflow**==true)**  cout**<<** "Warning: "**<<** name **<<**" wire structure cannot satisfy latency constraint." **<<** endl**;**  assert**(**power**.**readOp**.**dynamic **>** 0**);**  assert**(**power**.**readOp**.**leakage **>** 0**);**  assert**(**power**.**readOp**.**gate\_leakage **>** 0**);**  double long\_channel\_device\_reduction **=** longer\_channel\_device\_reduction**(**device\_ty**,**core\_ty**);**  double pg\_reduction **=** power\_gating\_leakage\_reduction**(false);**//  double sckRation **=** g\_tp**.**sckt\_co\_eff**;**  power**.**readOp**.**dynamic **\*=** sckRation**;**  power**.**writeOp**.**dynamic **\*=** sckRation**;**  power**.**searchOp**.**dynamic **\*=** sckRation**;**  power**.**readOp**.**longer\_channel\_leakage **=**  power**.**readOp**.**leakage**\***long\_channel\_device\_reduction**;**  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**;**  //只有全局线路可以选择是否路由  **if** **(**pipelinable**)**  area**.**set\_area**(**area**.**get\_area**()\***route\_over\_perc **+** no\_device\_under\_wire\_area**.**get\_area**()\*(**1**-**route\_over\_perc**));**  Wire wreset**();**  **}**  void interconnect**::**compute**()**  **{**  Wire **\***wtemp1 **=** 0**;**    // Wire(enum Wire\_type wire\_model, double len /\* in u\*/,  // int nsense = 1/\* no. of sense amps connected to the low-swing wire \*/,  // double width\_scaling = 1,  // double spacing\_scaling = 1,  // enum Wire\_placement wire\_placement = outside\_mat,  // double resistivity = CU\_RESISTIVITY,  // TechnologyParameter::DeviceType \*dt = &(g\_tp.peri\_global));    wtemp1 **=** **new** Wire**(**wt**,** length**,** 1**,** width\_scaling**,** space\_scaling**);**  delay **=** wtemp1**->**delay**;**  power**.**readOp**.**dynamic **=** wtemp1**->**power**.**readOp**.**dynamic**;**  power**.**readOp**.**leakage **=** wtemp1**->**power**.**readOp**.**leakage**;**  power**.**readOp**.**gate\_leakage **=** wtemp1**->**power**.**readOp**.**gate\_leakage**;**  area**.**set\_area**(**wtemp1**->**area**.**get\_area**());**  no\_device\_under\_wire\_area**.**h **=** **(**wtemp1**->**wire\_width **+** wtemp1**->**wire\_spacing**);**  no\_device\_under\_wire\_area**.**w **=** length**;**  **if** **(**wtemp1**)**  **delete** wtemp1**;**  **}** |