目录

[1. Pipeline类介绍 2](#_Toc532546351)

[2. Pipeline类声明 4](#_Toc532546352)

[3. 定义Pipeline的对象 5](#_Toc532546353)

[4. Pipeline的构造函数 6](#_Toc532546354)

[5. Computer函数 7](#_Toc532546355)

[6. compute\_stage\_vector函数 9](#_Toc532546356)

# Pipeline类介绍

* 该类定义了Core中由于使用流水线技术而带来的面积和功耗开销。开销的主要来源于流水级之间需要保存中间的状态
* 对于流水线中增加的存储使用一位的D触发器（DFF）模拟，可以理解为每一位都要可以单独的存取
* 通过估计流水级之间需要保存的额外的位数和D触发器的面积和功耗，得到整体的面积和功耗
* McPAT通过举例了一个按序6级流水线处理器需要增加的位数和一个乱序12级流水线处理器需要增加的位数，大致估算出在每一种结构下，平均每一级流水线需要增加的位数
* McPAT最后根据计算出的平均值，判断如果流水级大于6或者12，则使用级数\*平均值大致估算整体的增加的位数（不等于6/12应该也可以）

|  |
| --- |
| //假定需要50%额外的控制寄存器和中断寄存器（thumb中的设定）  //num\_piperegs为之前使用6级或者12级流水线估计出的数值  num\_piperegs **=** num\_piperegs **\*** 1.5**;**  tot\_stage\_vector**=**num\_piperegs**;**  //估算平均流水线中的每一级需要多少位来保存中间数据  per\_stage\_vector**=**tot\_stage\_vector**/**num\_stages**;**  //根据实际的处理器类型和流水级数，利用估算的结果进行估计  **if** **(**coredynp**.**core\_ty**==**Inorder**)**  **{**  **if** **(**coredynp**.**pipeline\_stages**>**6**)**  num\_piperegs**=** per\_stage\_vector**\***coredynp**.**pipeline\_stages**;**  **}**  **else**  **{**  **if** **(**coredynp**.**pipeline\_stages**>**12**)**  num\_piperegs**=** per\_stage\_vector**\***coredynp**.**pipeline\_stages**;**  **}** |

* 在Core中定义的corepipe最终结果将会分散到每个功能部件中，包括面积和功耗，不会单独显示流水级带来的面积和功耗

|  |
| --- |
| //乱序5个单元 unit，ifu, lsu, mmu, exu, rnu  //按序4个单元 unit，ifu, lsu, mmu, exu  //分散面积到每个单元  pipeline\_area\_per\_unit **=** **(**corepipe**->**area**.**get\_area**()\***coredynp**.**num\_pipelines**)/**num\_units**;**  rnu**->**area**.**set\_area**(**rnu**->**area**.**get\_area**()** **+** pipeline\_area\_per\_unit**);**  ifu**->**area**.**set\_area**(**ifu**->**area**.**get\_area**()** **+** pipeline\_area\_per\_unit**);**  lsu**->**area**.**set\_area**(**lsu**->**area**.**get\_area**()** **+** pipeline\_area\_per\_unit**);**  exu**->**area**.**set\_area**(**exu**->**area**.**get\_area**()** **+** pipeline\_area\_per\_unit**);**  mmu**->**area**.**set\_area**(**mmu**->**area**.**get\_area**()** **+** pipeline\_area\_per\_unit**);**  //分散功耗到每个单元  //按序中num\_units=4，乱序为5  temp**=**coredynp**.**num\_pipelines**/**num\_units**;**  pppm\_t**={**temp**,**temp**,**temp**,**temp**}**  rnu**->**power **=** rnu**->**power **+** corepipe**->**power**\***pppm\_t**;**  //增加占空比以提高计算的精度(rnu也可以增加该参数)  pppm\_t**={**temp**\***coredynp**.**IFU\_duty\_cycle**,**temp**,**temp**,**temp**}**  ifu**->**power **=** ifu**->**power **+** corepipe**->**power**\***pppm\_t**;**  pppm\_t**={**temp**\***coredynp**.**LSU\_duty\_cycle**,**temp**,**temp**,**temp**}**  lsu**->**power **=** lsu**->**power **+** corepipe**->**power**\***pppm\_t**;**  pppm\_t**={**temp**\***coredynp**.**ALU\_duty\_cycle**,**temp**,**temp**,**temp**}**  exu**->**power **=** exu**->**power **+** corepipe**->**power**\***pppm\_t**;**  pppm\_t**={**temp**\*(**0.5**+**0.5**\***coredynp**.**LSU\_duty\_cycle**),**temp**,**temp**,**temp**}**  mmu**->**power **=** mmu**->**power **+** corepipe**->**power**\***pppm\_t**;** |

# Pipeline类声明

|  |
| --- |
| //主要用于计算在流水线处理器中  //由于增加流水线而带来的面积开销和功耗开销  class Pipeline **:** public Component**{**  public**:**  /\*输入参数：  l\_ip=configure\_interface：mcpat和cacti之间参数传递的接口  coredynp=dyn\_p：core的参数信息  device\_ty=device\_ty\_：设备类型，默认为Core\_device  is\_core\_pipeline=\_is\_core\_pipeline：是否支持流水线，默认为true，即支持  is\_default=\_is\_default：  \*/  Pipeline**(**const InputParameter **\***configure\_interface**,**  const CoreDynParam **&** dyn\_p\_**,** enum Device\_ty device\_ty\_**=**Core\_device**,**  bool \_is\_core\_pipeline**=true,** bool \_is\_default**=true);**    InputParameter l\_ip**;**  uca\_org\_t local\_result**;**  CoreDynParam coredynp**;**  enum Device\_ty device\_ty**;**  bool is\_core\_pipeline**,** is\_default**;**    //流水级之间需要的寄存器个数，初始化为0  double num\_piperegs**;**  //判断core是否为嵌入式设备  //true代表不是，false代表是  bool process\_ind**;**  double WNANDn **;**  double WNANDp**;**  double load\_per\_pipeline\_stage**;**  //计算流水线需要增加的位数，用于保存临时结果  //使用估算的方式，计算任意级数的流水线结构  void compute\_stage\_vector**();**  //使用一位的D触发器对象和位数计算功耗和面积  void compute**();**  **~**Pipeline**(){**  local\_result**.**cleanup**();**  **};**  **};** |

# 定义Pipeline的对象

* Pipeline在Core的构造函数中定义，只传入两个参数，其余均使用默认值

|  |
| --- |
| /\*输入参数：  l\_ip = &interface\_ip  coredynp = coredynp  device\_ty = Core\_device，默认值  is\_core\_pipeline= true，默认值  is\_default = true，默认值 \*/  corepipe **=** **new** Pipeline**(&**interface\_ip**,**coredynp**);** |

* Pipeline中函数使用到的外部参数，大多数在估算流水线需要增加的位数时使用

|  |  |
| --- | --- |
| Name | Explanation |
| l\_ip.pipeline\_stages | 默认为0，core中的参数没有传入 |
| l\_ip.per\_stage\_vector | 默认为0 |
| coredynp.Embedded | 是否为嵌入式设备，工艺不同 |
| coredynp.core\_ty | 核的类型，inorder/OOO，此时计算流水线开销不同 |
| coredynp.pipeline\_stages | 流水级的个数  XML->sys.core[ithCore].pipeline\_depth[0] |
| coredynp.x86 | 获取操作码宽度，区别x86和其它ISA  coredynp.micro\_opcode\_length或者coredynp.opcode\_length |
| coredynp.micro\_opcode\_length | X86指令的微指令的操作码宽度，bits |
| coredynp.opcode\_length | 正常指令的操作码宽度，bits |
| coredynp.multithreaded | Bool类型，根据线程数设置，大于1则为true |
| coredynp.num\_hthreads | 硬件支持的线程数  XML->sys.core[ithCore].number\_hardware\_threads |
| coredynp.perThreadState | 默认设置为8bits，并且没有更改 |
| coredynp.pc\_width | XML->sys.virtual\_address\_width |
| coredynp.fetchW | XML->sys.core[ithCore].fetch\_width |
| coredynp.decodeW | XML->sys.core[ithCore].decode\_width |
| coredynp.issueW | XML->sys.core[ithCore].issue\_width |
| coredynp.instruction\_length | XML->sys.core[ithCore].instruction\_length (bits) |
| coredynp.arch\_ireg\_width | int(ceil(log2  (XML->sys.core[ithCore].archi\_Regs\_IRF\_size)))  体系结构寄存器的索引宽度 |
| coredynp.phy\_ireg\_width | PRF：  int(ceil(log2(…sys.core[ithCore].phy\_Regs\_IRF\_size)))  ROB：  int(ceil(log2(XML->sys.core[ithCore].ROB\_size))) |
| coredynp.v\_address\_width | XML->sys.virtual\_address\_width  虚拟地址的宽度 (bits) |
| coredynp.int\_data\_width | 整数的宽度  int(ceil(XML->sys.machine\_bits/32.0))\*32 |

# Pipeline的构造函数

|  |
| --- |
| //初始化一些传入的参数，同时给出一些更底层的电路参数  Pipeline**::**Pipeline**(**const InputParameter **\***configure\_interface**,**  const CoreDynParam **&** dyn\_p\_**,**enum Device\_ty device\_ty\_**,**  bool \_is\_core\_pipeline**,**bool \_is\_default**)**  **:** l\_ip**(\***configure\_interface**),**coredynp**(**dyn\_p\_**),**device\_ty**(**device\_ty\_**),**  is\_core\_pipeline**(**\_is\_core\_pipeline**)**  **,**is\_default**(**\_is\_default**),**num\_piperegs**(**0.0**)**  **{**  //生成uca\_org\_t对象  local\_result **=** init\_interface**(&**l\_ip**);**    //判断core是否是嵌入式设备，例如ARM  **if** **(!**coredynp**.**Embedded**)**  process\_ind **=** **true;**//不是嵌入式  **else**  process\_ind **=** **false;**//是嵌入式    //更底层的逻辑参数或者是工艺尺寸问题  WNANDn **=** **(**process\_ind**)?** 25 **\*** l\_ip**.**F\_sz\_um **:**  g\_tp**.**min\_w\_nmos\_ **;**  WNANDp **=** **(**process\_ind**)?** 37.5 **\*** l\_ip**.**F\_sz\_um **:**  g\_tp**.**min\_w\_nmos\_**\***pmos\_to\_nmos\_sz\_ratio**();**  //负载？  load\_per\_pipeline\_stage **=** 2**\***gate\_C**(**WNANDn **+** WNANDp**,** 0**,** **false);**  //计算功耗和面积  compute**();**  **}** |

# Computer函数

|  |
| --- |
| //计算功耗和面积  //使用一个一位的D触发器\*个数估算  void Pipeline**::**compute**()**  **{**  //计算流水线需要增加的位向量的长度  compute\_stage\_vector**();**    //定义一个一位触发器对象  DFFCell pipe\_reg**(false,** WNANDn**,**WNANDp**,** load\_per\_pipeline\_stage**,** **&**l\_ip**);**  //计算功耗  pipe\_reg**.**compute\_DFF\_cell**();**  //乘以整体的个数  double clock\_power\_pipereg **=** num\_piperegs **\*** pipe\_reg**.**e\_clock**.**readOp**.**dynamic**;**  double pipe\_reg\_power **=** num\_piperegs **\***  **(**pipe\_reg**.**e\_switch**.**readOp**.**dynamic**+**pipe\_reg**.**e\_keep\_0**.**readOp**.**dynamic  **+**pipe\_reg**.**e\_keep\_1**.**readOp**.**dynamic**)/**3**+**clock\_power\_pipereg**;**  double pipe\_reg\_leakage **=** num\_piperegs **\*** pipe\_reg**.**e\_switch**.**readOp**.**leakage**;**  double pipe\_reg\_gate\_leakage **=** num\_piperegs **\*** pipe\_reg**.**e\_switch**.**readOp**.**gate\_leakage**;**  power**.**readOp**.**dynamic **+=**pipe\_reg\_power**;**  power**.**readOp**.**leakage **+=**pipe\_reg\_leakage**;**  power**.**readOp**.**gate\_leakage **+=**pipe\_reg\_gate\_leakage**;**    //计算面积  area**.**set\_area**(**num\_piperegs **\*** pipe\_reg**.**area**.**get\_area**());**  double long\_channel\_device\_reduction **=**  longer\_channel\_device\_reduction**(**device\_ty**,** coredynp**.**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**;**  //乘以工艺系数  double sckRation **=** g\_tp**.**sckt\_co\_eff**;**  power**.**readOp**.**dynamic **\*=** sckRation**;**  power**.**writeOp**.**dynamic **\*=** sckRation**;**  power**.**searchOp**.**dynamic **\*=** sckRation**;**  double macro\_layout\_overhead **=** g\_tp**.**macro\_layout\_overhead**;**  //如果不是嵌入式，需要乘额外的工艺系数  **if** **(!**coredynp**.**Embedded**)**  area**.**set\_area**(**area**.**get\_area**()\***macro\_layout\_overhead**);**  **}** |

# compute\_stage\_vector函数

* 用于计算流水线需要增加多少存储位来保存中间结果
* 使用按序6级流水线和乱序12级流水线估计平均每级需要的位数（感觉计算的过程不是很清晰明白，必要时可以自己更改）

|  |  |
| --- | --- |
| 按序六级流水线 | |
| 0/IF | coredynp.pc\_width\*2\*coredynp.num\_hthreads  每个线程都需要保存两个PC值:PC, NPC |
| IF/ID | coredynp.fetchW\*(coredynp.instruction\_length + coredynp.pc\_width)\*coredynp.num\_hthreads  每个线程\*取指宽度\*(指令长度+每条指令的PC长度) |
| IF/ThreadSEL | 只在多线程的情况下存在  coredynp.num\_hthreads\*coredynp.perThreadState |
| ID/EXE | coredynp.decodeW\*coredynp.num\_hthreads \* (coredynp.instruction\_length + coredynp.pc\_width + pow(2.0,opcode\_length)+ 2\*coredynp.int\_data\_width)  译码宽度\*(每条指令的长度+指令PC+两个源操作的宽度+译码之后的总操作码长度)\*线程数  pow(2.0,opcode\_length)代表总操作码的解码信号 |
| EXE/MEM | coredynp.issueW\*(3 \* coredynp.arch\_ireg\_width + pow(2.0,opcode\_length) + **8\*2\*coredynp.int\_data\_width**)  发射宽度\*(3\*索引定点寄存器宽度+pow(2.0,opcode\_length)+ **8\*2\*整型数据宽度**) |
| MEM/WB | coredynp.issueW\*(2\*coredynp.int\_data\_width + pow(2.0,opcode\_length) + 8\*2\*coredynp.int\_data\_width)  发射宽度\*(2\*两个整型宽度+总的操作码解码信号+**8\*2\*整型数据宽度**) |

|  |  |
| --- | --- |
| 乱序12级流水线  Fetch, decode, rename, IssueQ, dispatch, regread, EXE, MEM, WB, CM | |
| 0/1F | coredynp.pc\_width\*2\*coredynp.num\_hthreads |
| IF/ID | coredynp.fetchW\*coredynp.num\_hthreads \* (coredynp.instruction\_length + coredynp.pc\_width)  此时的PC用于ID级的分支预测 |
| ID/Renaming | coredynp.decodeW\*coredynp.num\_hthreads \* (coredynp.instruction\_length + coredynp.pc\_width)  此时的PC用于之后分支计算使用 |
| Renaming/  Wire\_drive | coredynp.decodeW\*  (coredynp.instruction\_length + coredynp.pc\_width) |
| Renaming/  IssueQ | coredynp.issueW\*coredynp.num\_hthreads \* (coredynp.instruction\_length + coredynp.pc\_width + 3\*coredynp.phy\_ireg\_width)  两个源操作数+1个目的操作数 |
| IssueQ/  Dispatch | coredynp.issueW\*  (coredynp.instruction\_length + 3 \* coredynp.phy\_ireg\_width) |
| Dispatch/  Exe | coredynp.issueW\*(3 \* coredynp.phy\_ireg\_width + coredynp.pc\_width + pow(2.0,opcode\_length)) |
| Exe/Exe | coredynp.issueW\*  (2\*coredynp.int\_data\_width + pow(2.0,opcode\_length)) |
| Exe/Exe | coredynp.issueW\*  (2\*coredynp.int\_data\_width + pow(2.0,opcode\_length)) |
| Exe/Mem | coredynp.issueW\*(coredynp.int\_data\_width + coredynp.v\_address\_width + pow(2.0,opcode\_length)) |
| Mem/WB | coredynp.issueW\*  (coredynp.int\_data\_width + coredynp.phy\_ireg\_width) |
| WB/CM | coredynp.commitW\*(coredynp.int\_data\_width + coredynp.v\_address\_width + coredynp.phy\_ireg\_width)\* coredynp.num\_hthreads |

* 代码

|  |
| --- |
| //计算流水线中需要增加多少的存储位数，保存中间结果  void Pipeline**::**compute\_stage\_vector**()**  **{**  //级数，整体需要的位数，每级的位数  double num\_stages**,** tot\_stage\_vector**,** per\_stage\_vector**;**  //指令的opcode(操作码)的长度  int opcode\_length **=** coredynp**.**x86**?** coredynp**.**micro\_opcode\_length**:**coredynp**.**opcode\_length**;**      //默认情况下，is\_core\_pipeline=true，即一般不会执行到此处  **if** **(!**is\_core\_pipeline**)**//非流水线core  **{**  //l\_ip.pipeline\_stages=0，l\_ip.per\_stage\_vector=0  num\_piperegs**=**l\_ip**.**pipeline\_stages**\***l\_ip**.**per\_stage\_vector**;**  **}**  **else**  **{**  //如果是按序流水线的情况  //假设是六级流水线，尽可能的估计出每一级需要的位数  **if** **(**coredynp**.**core\_ty**==**Inorder**)**  **{**  // 取指之前的流水级  //每个线程都需要保存两个PC值:PC, NPC  num\_piperegs **+=** coredynp**.**pc\_width**\***2**\***coredynp**.**num\_hthreads**;**    // IF/ID之间的流水级  //每个线程\*取指宽度\*(指令长度+每条指令的PC长度)  num\_piperegs **+=** coredynp**.**fetchW**\*(**coredynp**.**instruction\_length **+** coredynp**.**pc\_width**)\***coredynp**.**num\_hthreads**;**    // 流水级 IF/ThreadSEL，只在多线程的情况下存在  **if** **(**coredynp**.**multithreaded**)**  //每个线程\*每个线程的状态(默认情况是8bits)  num\_piperegs **+=** coredynp**.**num\_hthreads**\***coredynp**.**perThreadState**;**    // 流水级 ID/EXE  // 译码宽度\*(每条指令的长度+指令PC+两个源操作的宽度+pow(2.0,opcode\_length))\*线程数  // pow(2.0,opcode\_length)代表总操作码的解码信号  num\_piperegs **+=** coredynp**.**decodeW**\*(**coredynp**.**instruction\_length **+** coredynp**.**pc\_width **+** pow**(**2.0**,**opcode\_length**)+** 2**\***coredynp**.**int\_data\_width**)\***coredynp**.**num\_hthreads**;**    // 流水级 EXE/MEM  // 发射宽度\*(3\*索引定点寄存器宽度+pow(2.0,opcode\_length)+ 8\*2\*整型数据宽度)  num\_piperegs **+=** coredynp**.**issueW**\*(**3 **\*** coredynp**.**arch\_ireg\_width **+** pow**(**2.0**,**opcode\_length**)** **+** 8**\***2**\***coredynp**.**int\_data\_width**);**    // 流水级 MEM/WB  //发射宽度\*(2\*两个整型宽度+总的操作码解码信号+)  num\_piperegs **+=** coredynp**.**issueW**\*(**2**\***coredynp**.**int\_data\_width **+** pow**(**2.0**,**opcode\_length**)** **+** 8**\***2**\***coredynp**.**int\_data\_width**);**  num\_stages**=**6**;**  **}**  **else**  **{**  //乱序流水线：假设12级  /\*OOO: Fetch, decode, rename, IssueQ, dispatch, regread, EXE, MEM, WB, CM \*/  /\* pipe stage 0/1F\*/  // PC宽度\*2\*线程数，PC，NPC    num\_piperegs **+=** coredynp**.**pc\_width**\***2**\***coredynp**.**num\_hthreads **;**  /\* pipe stage IF/ID \*/  //每个线程\*取指宽度\*(指令长度+每条指令的PC长度)  // 此时的PC用于ID级的分支预测  num\_piperegs **+=** coredynp**.**fetchW**\*(**coredynp**.**instruction\_length **+** coredynp**.**pc\_width**)\***coredynp**.**num\_hthreads**;**    /\* pipe stage 1D/Renaming\*/  //译码宽度\*(指令长度+PC)\*线程数  //此时的PC用于之后分支指令使用  num\_piperegs **+=** coredynp**.**decodeW**\*(**coredynp**.**instruction\_length **+** coredynp**.**pc\_width**)\***coredynp**.**num\_hthreads**;**    /\* pipe stage Renaming/wire\_drive 重命名/线驱动\*/  //译码宽度\*(指令长度+PC)  num\_piperegs **+=** coredynp**.**decodeW**\*(**coredynp**.**instruction\_length **+** coredynp**.**pc\_width**);**      /\* pipe stage Renaming/IssueQ \*/  // 发射宽度\*(指令长度+PC+3\*索引定点物理寄存器的长度)\*线程数  //两个源操作数+1个目的操作数  num\_piperegs **+=** coredynp**.**issueW**\*(**coredynp**.**instruction\_length **+** coredynp**.**pc\_width **+** 3**\***coredynp**.**phy\_ireg\_width**)\***coredynp**.**num\_hthreads**;**    /\* pipe stage IssueQ/Dispatch \*/  //发射宽度\*(指令长度+3\*索引定点物理寄存器的长度)  //此时不再需要记录线程信息  num\_piperegs **+=** coredynp**.**issueW**\*(**coredynp**.**instruction\_length **+** 3 **\*** coredynp**.**phy\_ireg\_width**);**    /\* pipe stage Dispatch/EXE \*/  //发射宽度\*(指令长度+3\*索引定点物理寄存器的长度)  num\_piperegs **+=** coredynp**.**issueW**\*(**3 **\*** coredynp**.**phy\_ireg\_width **+** coredynp**.**pc\_width **+** pow**(**2.0**,**opcode\_length**));**  /\* 2^opcode\_length means the total decoded signal for the opcode\*/    num\_piperegs **+=** coredynp**.**issueW**\*(**2**\***coredynp**.**int\_data\_width **+** pow**(**2.0**,**opcode\_length**));**    /\*2 source operands in EXE; Assume 2EXE stages\* since we do not really distinguish OP\*/  num\_piperegs **+=** coredynp**.**issueW**\*(**2**\***coredynp**.**int\_data\_width **+** pow**(**2.0**,**opcode\_length**));**    /\* pipe stage EXE/MEM, data need to be read/write, address\*/  num\_piperegs **+=** coredynp**.**issueW**\*(**coredynp**.**int\_data\_width **+** coredynp**.**v\_address\_width **+** pow**(**2.0**,**opcode\_length**));**//memory Opcode still need to be passed    /\* pipe stage MEM/WB; result data, writeback regs \*/  num\_piperegs **+=** coredynp**.**issueW**\*(**coredynp**.**int\_data\_width **+** coredynp**.**phy\_ireg\_width**);**    /\* pipe stage WB/CM ; result data, regs need to be updated, address for resolve memory ops in ROB's top\*/  num\_piperegs **+=** coredynp**.**commitW**\*(**coredynp**.**int\_data\_width **+** coredynp**.**v\_address\_width **+** coredynp**.**phy\_ireg\_width**)\***coredynp**.**num\_hthreads**;**  num\_stages**=**12**;**  **}**  //假定需要50%额外的控制寄存器和中断寄存器（thumb中的设定）  num\_piperegs **=** num\_piperegs **\*** 1.5**;**  tot\_stage\_vector**=**num\_piperegs**;**    //估算平均流水线中的每一级需要多少位来保存中间数据  per\_stage\_vector**=**tot\_stage\_vector**/**num\_stages**;**  //根据实际的处理器类型和流水级数，利用估算的结果进行估计  **if** **(**coredynp**.**core\_ty**==**Inorder**)**  **{**  **if** **(**coredynp**.**pipeline\_stages**>**6**)**  num\_piperegs**=** per\_stage\_vector**\***coredynp**.**pipeline\_stages**;**  **}**  **else**  **{**  **if** **(**coredynp**.**pipeline\_stages**>**12**)**  num\_piperegs**=** per\_stage\_vector**\***coredynp**.**pipeline\_stages**;**  **}**  **}**  **}** |