|  |  |
| --- | --- |
| 文档编号 |  |
| 文档版本 |  |
| 文档管控 | 内部公开 |
| 存档日期 |  |

昆明湖项目FTQ模块AS

|  |  |
| --- | --- |
| 编 写： | 高泽宇 |
| 校 对： |  |
| 审 核： |  |
| 批 准： |  |

昆明湖V1项目

2024年XX月XX日

文档修订记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序号** | **版本编号** | **变化状态** | **变更说明** | **作者** | **日期** |
| 1 | V0.1 | C | 昆明湖版本初稿 | 高泽宇 | 2024/01/15 |
|  |  |  |  |  |  |
| 3 |  |  |  |  |  |
|  |  |  |  |  |  |

\*变化状态：C—创建，A—增加，M—修改，D—删除

文档审批信息

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本** | **审核** | **会签** | **批准** | **备注** |
| V1.0 |  |  |  |  |
|  |  |  |  |  |

目 录

[1 简介 1](#_Toc107389929)

[1.1 文档介绍 1](#_Toc107389930)

[1.2 参考文档 1](#_Toc107389931)

[1.3 术语说明 1](#_Toc107389932)

[1.4 技术背景 1](#_Toc107389933)

[2 设计规格 2](#_Toc107389934)

[3 功能描述 2](#_Toc107389935)

[4 总体设计 2](#_Toc107389936)

[4.1 整体框图 2](#_Toc107389937)

[4.2 接口列表 3](#_Toc107389938)

[4.3 接口时序 3](#_Toc107389939)

[4.4 时钟复位 4](#_Toc107389940)

[4.5 寄存器配置 4](#_Toc107389941)

[4.6 补充说明 5](#_Toc107389942)

[5 模块设计 5](#_Toc107389943)

[5.1 二级模块A 6](#_Toc107389944)

[5.1.1 功能 6](#_Toc107389945)

[5.1.2 整体框图 6](#_Toc107389946)

[5.1.3 接口列表 6](#_Toc107389947)

[5.1.4 接口时序 6](#_Toc107389948)

[5.1.5 关键电路 6](#_Toc107389949)

[5.1.6 三级模块设计 6](#_Toc107389950)

[5.2 二级模块B 6](#_Toc107389951)

[5.3 关键电路 6](#_Toc107389952)

[5.3.1 时钟切换电路 6](#_Toc107389953)

[6 PPA优化设计 7](#_Toc107389954)

[7 验证关注点 7](#_Toc107389955)

[8 Floorplan 建议 7](#_Toc107389956)

[9 遗留问题 8](#_Toc107389957)

1. 简介
   1. 文档介绍

本文档是取指目标队列（Fetch Target Queue）的文档，描述昆明湖架构取指目标队列单元设计。

本文档主要用于指导芯片模块的详细设计及验证。

* 1. 参考文档

无。

* 1. 术语说明

表1.1 术语说明

|  |  |  |
| --- | --- | --- |
| **缩写** | **全称** | **描述** |
| CRU | Clock Reset Unit | 时钟复位单元 |
| FTQ | Fetch Target Queue | 取指目标队列 |
| FTB | Fetch Target Buffer | 取指目标缓冲 |
|  |  |  |

* 1. 技术背景

无。

1. 设计规格

* 支持暂存 BPU 预测的取指目标，并向 IFU 发送取指请求
* 支持暂存 BPU 的预测信息，并送回 BPU 训练
* 支持后端读取指令 PC
* 支持重定向恢复
* 支持向 ICache发送预取请求

1. 功能描述
   1. 功能概述

FTQ 是分支预测和取指单元之间的缓冲队列，它的主要职能是暂存 BPU 预测的取指目标，并根据这些取指目标给 IFU 发送取指请求。它的另一重要职能是暂存 BPU 各个预测器的预测信息，在指令提交后把这些信息送回 BPU 用作预测器的训练，因此它需要维护指令从预测到提交的完整的生命周期。

* 1. 功能详述
     1. 暂存BPU预测的取指目标，并向IFU发送取指请求
        1. 暂存BPU预测的取指目标
           1. 存储PC的结构

BPU 的一次预测会经历三个流水级，每一个流水级都会产生新的预测内容。FTQ 接收来自 BPU 每个流水级的预测结果，并且后面的流水级的结果会覆盖前面流水级的结果。

指令以预测块为单位，从 BPU 发出，进入 FTQ，同时 bpuPtr 指针加一，初始化对应 FTQ 项的各种状态，把各种预测信息写入存储结构；如果预测块来自 BPU 覆盖预测逻辑，则恢复 bpuPtr 和 ifuPtr。

BPU 预测的取值目标被 FTQ 暂存于 ftq\_pc\_mem 中：

* ftq\_pc\_mem: 寄存器堆实现，为存储与指令地址相关的信息，包括如下的域：
  + startAddr 预测块起始地址。
  + nextLineAddr 预测块下一个缓存行的起始地址。
  + isNextMask 预测块每一条可能的指令起始位置是否在按预测宽度对齐的下一个区域内。isNextMask有16bit，每个bit表示相对起始地址的2byte\*n位置是否跨cacheline，表示的是每个位置的性质。
  + fallThruError 预测出的下一个顺序取指地址是否存在错误。

每一个域都各自存在自己的寄存器（例如data\_0\_startAddr）里，并没有拼接后存进同一个Reg里。

* + - * 1. 计算PC的方式

每次从ICache取指都会取一个或两个CacheLineSize（64 Bytes）长度的缓存行指令数据，是否取两个由预测块是否跨缓存行决定。

而每个预测块的长度为PredictWidth（16）个压缩指令的长度（32 Bytes）。每个缓存行的长度为每个预测块长度的两倍，所以每个预测块的startAddr要么在当前缓存行的前半部分（startAddr[5] = 0），要么在当前缓存行的后半部分（startAddr[5] = 1）。

如果startAddr[5] = 0，那么当前预测块必然不会跨缓存行，那么此时预测指令pc = {startAddr[38, 6], startAddr[5, 1] + offset, 1’b0}。

如果startAddr[5] = 1，那么当前预测块可能会出现跨缓存行的情况。此时：

* 如果isNextMask(offset) = 0，表示当前预测指令pc未跨缓存行，那么此时预测指令pc = {startAddr[38, 6], startAddr[5, 1] + offset, 1’b0}。
* 如果isNextMask(offset) = 1，表示当前预测指令pc跨越了缓存行，那么此时预测指令pc = {nextLineAddr[38,6], startAddr[5, 1] + offset, 1’b0}。
  + - 1. 向IFU发送取指请求

FTQ 向 IFU 发出取指请求，ifuPtr 指针加一，等待预译码信息写回。

IFU 写回的预译码信息被 FTQ 暂存于 ftq\_pd\_mem 中：

* + ftq\_pd\_mem: 寄存器堆实现，存储取指单元返回的预测块内的各条指令的译码信息，包括如下的域：
    - brMask 每条指令是否是条件分支指令。
    - jmpInfo 预测块末尾无条件跳转指令的信息，包括它是否存在、是 jal 还是 jalr、是否是 call 或 ret 指令。
    - jmpOffset 预测块末尾无条件跳转指令的位置。
    - jalTarget 预测块末尾 jal 的跳转地址。
    - rvcMask 每条指令是否是压缩指令。
    1. 暂存BPU的预测信息，并送回BPU训练
       1. 暂存BPU的预测信息

BPU 传给 FTQ 的预测信息除了会暂存到上文提到的 ftq\_pc\_mem 中，还有部分信息会存储到 ftq\_redirect\_sram、ftq\_pc\_mem 和 ftb\_entry\_mem 中。

* ftq\_redirect\_sram: SRAM 实现，存储那些在重定向时需要恢复的预测信息，主要包括和 RAS 和分支历史相关的信息。分为3个bank，每个bank的深度×宽度为64×236。
* ftq\_meta\_1r\_sram: SRAM 实现，存储其余的 BPU 预测信息。SRAM的深度×宽度为64×256。
* ftb\_entry\_mem: 寄存器堆实现，存储预测时 FTB 项的必要信息，用于提交后训练新的 FTB 项。为什么要存ftb\_entry呢？因为更新的时候ftb\_entry需要在原来的基础上继续修改，为了不重新读一遍ftb，所以这里将ftb\_entry存在ftb\_entry\_mem中。

FTQ中的各个sram/mem的具体实现机制见下表：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 写入时机（正向写入） | 更新时机（反向更新，比如重定向等 ） | 读出时机 | 写入的数据内容 | 更新的数据内容 |
| ftq\_pc\_mem | BPU流水级的S1阶段，创建新的预测entry时写入 | 不存在（目前的设计是FTQ汇总重定向发到BPU和IFU，等bpu再把重定向到新地址的预测块重新入队的时候在ftq\_pc\_mem写入新的块，ftq\_pc\_mem的项是表示当前预测块的地址，而不包括target，所以不需要更新预测出错的那个块） | 读数据每个时钟周期都会存进Reg。如果IFU不需要从bypass中读取数据，Reg数据直连给Icache和IFU | startAddr：预测块起始地址  nextLineAddr：预测块下一个缓存行的起始地址  isNextMask：预测块每一条可能的指令起始位置是否在按预测宽度对齐的下一个区域内（①如果isNextMask(offset) = 0，表示当前预测指令pc未跨缓存行，那么此时预测指令pc = {startAddr[38, 6], startAddr[5, 1] + offset, 1’b0}。②如果isNextMask(offset) = 1，表示当前预测指令pc跨越了缓存行，那么此时预测指令pc = {nextLineAddr[38,6], startAddr[5, 1] + offset, 1’b0}。）  fallThruError：预测出的下一个顺序取指地址是否存在错误 | 无 |
| ftq\_meta\_1r\_sram | BPU流水级的S3阶段 | FTQ项中的指令能够commit的时候，将meta数据读出，发送给bpu训练 | 写入的数据包含了4个预测器的预测信息 |
| ftb\_entry\_mem | BPU流水级的S3阶段 | 1.backend重定向  2.ifu写回预译码信息  3.ifu预译码检测出错误发送重定向 | BrSlot: brSlot\_offset/lower/tarStat/sharing/valid  TailSlot: tailSlot\_offset/lower/tarStat/sharing/valid  pftAddr,carry,isCall,isRet,isJalr… |
| ftq\_pd\_mem | IFU阶段F3流水的下一拍 | 一直在读commPtr作为地址对应的数据，赋值给ftbEntryGen | rvcMask  brMask  jmpInfo  jmpOffset  jalTarget |

* + - 1. 送回BPU训练

指令在后端提交时会通知 FTQ 此指令已经提交。当 FTQ 项中所有的有效指令都已在后端提交，commPtr 指针加一，从存储结构中读出相应的信息，送给 BPU 进行训练。

* + 1. 后端读取指令PC

由于后端存储 PC 的开销较大，当后端需要指令 PC 的时候，会到 FTQ 读取。

* + 1. 重定向恢复

每次预测后，RAS 的栈顶项和栈指针都会存入 FTQ 的 ftq\_redirect\_sram，同时使用的 BPU 全局历史会存入 FTQ ，用于误预测恢复。

* + - 1. 预译码检测出预测错误

FTQ 向 IFU 发出取指请求后，IFU会向FTQ写回预译码信息，ifuWbPtr指针加一。如果预译码检测出了预测错误，则向BPU发送相应的重定向请求。FTQ根据重定向信号中的ftqIdx恢复bpuPtr和ifuPtr。

* + - 1. 后端检测出误预测

如果指令在后端执行时检测出误预测，则通知FTQ，FTQ给IFU和BPU发送对应的重定向请求，同时FTQ根据重定向信号中的ftqIdx恢复bpuPtr、ifuPtr和ifuWbPtr。

为了实现提前一拍读出在ftq中存储的重定向数据，减少redirect损失，后端会向ftq提前一拍（相对正式的后端redirect信号）传送ftqIdxAhead信号和ftqIdxSelOH信号。但是提前一拍后端无法及时得到准确的ftqIdx，需要在4个Alu通路中进行仲裁，但是仲裁结果在正式的后端redirect信号有效时才能得到，所以FTQ得到的提前一拍redirect的ftqIdx信号需要四个通路都读。

* io.fromBackend.ftqIdxAhead：7个FtqIdx。表示需要重定向的预测块在ftq中存储的索引。有7个是因为后端在最终仲裁前有7个可能产生redirect信号的通路，分别是Jump\*1、Alu\*4、LdReplay\*1、Exception\*1，但是其中只有Alu\*4产生的redirect信号我们会提前读，所以 ftqIdxAhead 实际用到的只有4个 FtqIdx。
* Io.fromBackend.ftqIdxSelOH：4位独热码+valid，表示4条通路的 ftqIdxAhead 有效与否，高有效。
  + 1. 向ICache发送预取请求

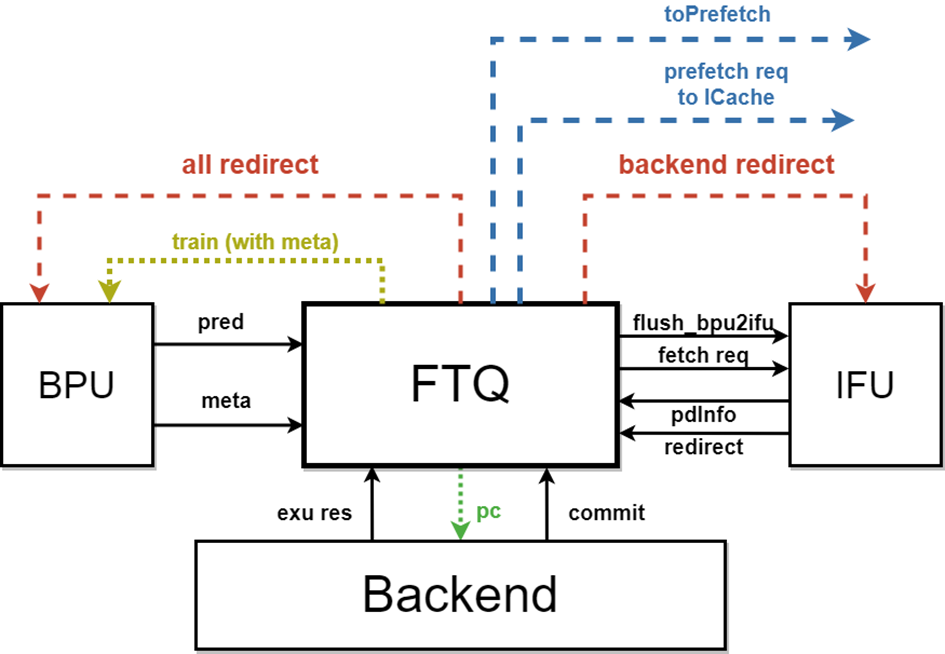
由于 BPU 基本无阻塞，它经常能走到 IFU 的前面，于是 FTQ 中实现了将 BPU 提供的还没发到 IFU 的取指请求用作指令预取，直接向指令缓存发送预取请求。

1. 总体设计

*总体设计的标准：对设计进行分解，完成子模块划分、顶层接口定义、接口时序、数据流、控制流的设计。*

*总体设计面向的对象：顶层集成人员、验证人员、软件人员、设计人员*

* 1. 整体框图

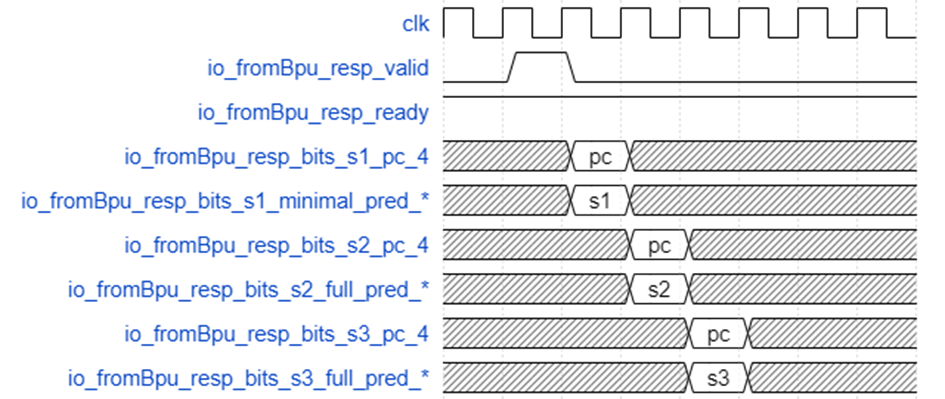


* 1. 接口列表

见excel。

* 1. 接口时序

1. BPU到FTQ接口时序



上图示意了BPU到FTQ的预测结果接口时序。当对应的握手信号io\_fromBpu\_resp\_valid和io\_fromBpu\_resp\_ready同时为高时，BPU三个流水级的预测结果在流水线内1、2、3阶段分别输入至FTQ。

若BPU后面流水级的预测结果与之前流水级不一致，则对应的redirect信号io\_fromBpu\_resp\_bits\_s2\_hasRedirect\_4或io\_fromBpu\_resp\_bits\_s3\_hasRedirect\_4会被拉高，表明需要刷新预测流水线。

* 1. 时钟复位

|  |  |  |
| --- | --- | --- |
| Module | Clock | Reset |
| BPU | clock | reset |

* 1. 寄存器配置

*如果模块涉及寄存器配置（包括状态信息、统计信息通过寄存器上报软件）需要简要描述相关的寄存器，描述可使用表格，格式如下。*

*总体AS可忽略此节*

表4.2 XXXX寄存器说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **寄存器** | **地址** | **复位值** | **属性** | **描述** |
| cfg\_fetch\_en | 0X0 | 32’d0 | RW | bit31-1: 保留  bit0: fetch\_en寄存器配置信号 |
| cfg\_clk\_sel | 0X 4 | 32’d0 | RW | bit31-1: 保留  bit0: 时钟动态切换信号  0：选择晶振时钟  1：选择PLL时钟 |
| pll\_lock | 0X 8 | 32’d0 | RO | bit31-1: 保留  bit0: PLL锁定信号  0：PLL未锁定  1：PLL锁定 |
|  |  |  |  |  |

*注：RO——只读寄存器；RW——可读可写寄存器*

* 1. 补充说明

*可选项。按照模块特点，根据4.1整体框图的划分，补充部分核心模块、关键电路、关键信号信号的说明。*

1. 模块设计

*模块设计的标准：能用于指导RTL代码的编写。理想情况下，RTL代码是对设计方案的翻译。*

*模块设计面向的对象：模块设计人员、模块验证人员*

*本模块下面各级子模块的详细设计说明。包括模块功能概述、模块IO、模块的设计框图、关键设计（流水线、memory（ram、fifo、寄存器组等）、主控制电路（包含不限于状态机、仲裁、关键握手时序等））。*

*对于关键设计描述要求：*

1. *Memory（ram、寄存器组、fifo等）：宽度、深度、接口含义、读写时序、data的详细描述、data在memory中存放的格式等信息*
2. *流水：有流水线框图、每一级流水线描述*
3. *仲裁：仲裁策略、优先级处理等*
4. *状态机：有状态机设计图，需要有每个状态描述、状态之间的跳转条件、复位状态等。*

*例如：*

* 1. 二级模块A
     1. 功能概述
     2. 整体框图

*要求如4.1*

* + 1. 接口列表

*只需要写IO信号*

* + 1. 接口时序
    2. 关键电路

*如果该层级只有例化，没有其他电路可不写。*

*主要包括Memory、流水线设计、仲裁、状态机等。多画图，配以文字说明*



图表 1 xx存储结构图

* + 1. 三级模块设计

*内容和要求同二级模块设计，如果还有4级模块需要一级级往下写。*

* 1. 二级模块B
  2. 关键电路

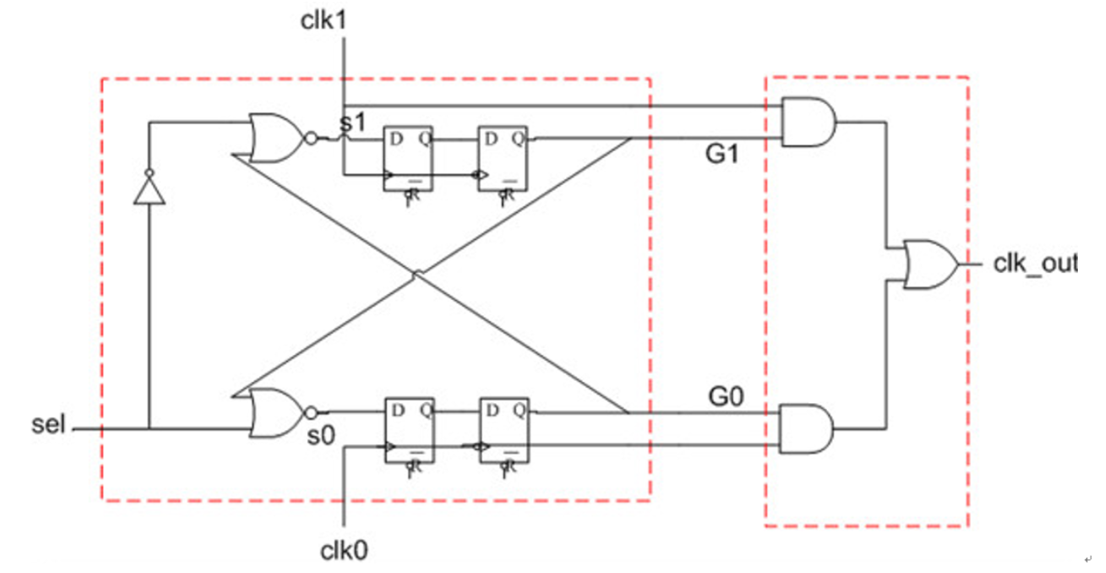
*例如：*

* + 1. 时钟切换电路

上述CRG框图中，紫色粗框内的clock\_mux是动态时钟切换模块，在用户程序配置PLL之后，通过配置寄存器cfg\_clk\_sel=1将系统时钟从晶振动态切换到PLL时钟。

为了保证时钟动态切换不会导致系统出错，需要使用无毛刺时钟切换电路，电路图如下：

图 2 无毛刺时钟切换电路图



上图中，下面两个寄存器的复位值为1，上面两个寄存器的复位值为0。复位时，clk\_out默认选择clk0时钟。

对于两级同步寄存器，同步器的第一级采用时钟上升沿触发，第二级采用时钟下降沿触发。

1. PPA

*内容包含:*

1. *Power*

*描述功耗设计目标*

*详细描述设计的功耗预估情况*

1. *Performance*

*详细描述设计的性能目标数据*

*详细描述性能的预估情况*

1. *Area*

*详细描述设计的面积目标数据*

*详细描述面积的预估情况*

*4.为优化PPA做的一些关键设计点（例如为了时序收敛做的一些面积/功耗/性能上的折中）*

1. 验证关注点

*从设计角度列举需要验证人员特别关注的测试点。*

*不涉及填“NA”*

1. Floorplan 建议

*芯片的floorplan考虑，依据数据流向，IO排布，模块大小进行芯片布局摆放设计*

*不涉及填“NA”*

1. 遗留问题

*需要跟踪的遗留问题*

*不涉及填“NA”*