# Revision History

| Author | Description | Date |
|---|---|---|
| 刘恒雨 | Initialised and updated spec doc | 2025/04/10 |

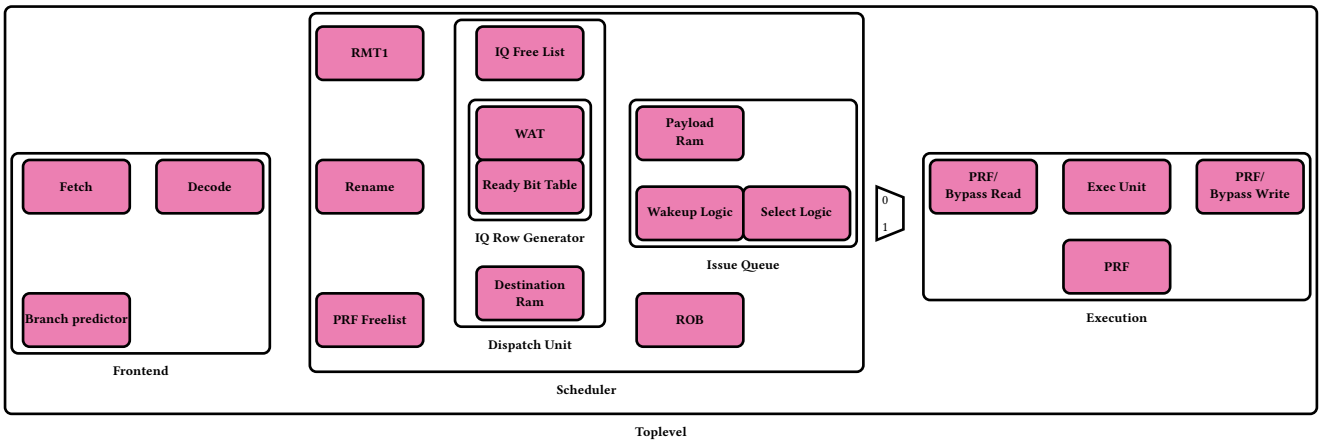
# Contents



# 1 Terminology

**PRF** Physical Register File
**WAT** Wakeup Allocation Table
**RMT** Register Map Table
**SDA** Store Data Array
**IRL** Instruction Replay Logic
**STQ** STore Queue
**LDQ** LoaD Queue

# 2 Overview




# 3 Parameters

# 4 Interface

## 4.1 BP_IFU_Interface

BP_IFU_Interface 的构造参数如下：

| Name | Type | Description |
| --- | --- | --- |
| **fetch_width** | `Int` | val GHR_width : Int |

IO 端口定义如下：

| Name | Type | Description |
| --- | --- | --- |
| **PC_curr** | `Input(UInt(32.W))` | 当前 IFU 的 PC 值 |
| **PC_target** | `Output(Vec(fetch_width, UInt(32.W)))` | 预测的目标地址 |
| **BTB_Hit** | `Output(Vec(fetch_width, Bool()))` | 是否命中 BTB |
| **BHT_Taken** | `Output(Vec(fetch_width, Bool()))` | 是否预测为 taken |
| **GHR** | `Output(UInt(GHR_width.W))` | 作出预测时的全局历史寄存器快照，使得更新 BHT 时能够生成正确的 index |

## 4.2 BP_ROB_Interface

BP_ROB_Interface 的构造参数如下：

| Name | Type | Description |
| --- | --- | --- |
| **GHR_width** | `Int` | |

IO 端口定义如下：

| Name | Type | Description |
| --- | --- | --- |
| **PC** | `Input(UInt(32.W))` | 当前 ROB 的 PC 值 |
| **instrType** | `Input(UInt(2.W))` | 当前指令类型,该模块需要区分条件分支和无条件分支 |
| **BTB_Hit** | `Input(Bool())` | 是否命中 BTB。根据条件分支最初是否命中 BTB，给出不同的更新 BHT 策略 |
| **predict_Taken** | `Input(Bool())` | 是否预测为 taken |
| **actual_Taken** | `Input(Bool())` | 实际是否 taken |
| **GHR** | `Input(UInt(GHR_width.W))` | 作出预测时的全局历史寄存器快照，使得更新 BHT 时能够生成正确的 index |
| **actualTargetPC** | `Input(UInt(32.W))` | 实际跳转的目标地址 |

## 4.3 BP_IO

BP_IO 的构造参数如下：

| Name | Type | Description |
| --- | --- | --- |
| **fetch_width** | `Int` | val GHR_width : Int |

IO 端口定义如下：

| Name | Type | Description |
| --- | --- | --- |
| **ifu** | `new BP_IFU_Interface(fetch_width, GHR_width)` | |
| **rob** | `new BP_ROB_Interface(GHR_width)` | |

## 4.4 ALUIO

IO 端口定义如下：

| Name | Type | Description |
| --- | --- | --- |
| **in1** | `Input(UInt(p(XLen).W))` | |
| **in2** | `Input(UInt(p(XLen).W))` | |
| **fn** | `Input(UInt(4.W))` | |
| **out** | `Output(UInt(p(XLen).W))` | |
| **cmp_out** | `Output(Bool())` | |

### 4.5 BypassNetworkIO

IO 端口定义如下：

| Name | Type | Description |
| --- | --- | --- |
| **exec_units** | `Input(Vec(2, Valid(new BypassInfo)))` | |
| **preg_rd** | `Input(UInt(p(PhysRegIdxSz).W))` | |
| **data_out** | `Output(UInt(p(XLen).W))` | |

### 4.6 DCacheRequest

IO 端口定义如下：

| Name | Type | Description |
| --- | --- | --- |
| **addr** | `UInt(p.CoreMaxAddrbits.W)` | |
| **data** | `Bits(p.CoreDataBits.W)` | |

### 4.7 DCacheResponse

IO 端口定义如下：

| Name | Type | Description |
| --- | --- | --- |
| **addr** | `UInt(p.CoreMaxAddrbits.W)` | |
| **data** | `Bits(p.CoreDataBits.W)` | |

### 4.8 LSUMemIO

描述与数据内容交互的各种信号，用于管理 L/S 请求和缓存访问，包含一部分异常处理、顺序控制的功能

IO 端口定义如下：

| Name | Type | Description |
| --- | --- | --- |
| **req** | `new DecoupledIO(p.lsuWidth,Valid(new DCacheRequest))` | LSU 发出的数据缓存请求 |
| **resp** | `new Flipped(p.lsuWidth,Valid(new DCacheResponse))` | LSU 接收数据缓存响应 |
| **req_kill** | `Output(p.lsuWidth,Bool())` | 表示每个 req 是否被 kill |
| **req_nack_adv** | `Input(p.lsuWidth,Bool())` | 表示某个 req 是否被拒绝 |
| **store_ack** | `Flipped(Vec(p.lsuWidth,new ValidIO(new DCacheRequest)))` | 存储请求的确认 |

| | | |
|---|---|---|
| **nack** | `Flipped(Vec(p.lsuWidth,new ValidIO(new DCacheRequest)))` | 作为接口接受 neck |
| **load_rel_resp** | `Flipped(new DecoupledIO(new DCacheResponse))` | 作为接口接受缓存的 load/release 响应 |
| **bradate** | `Output(new bradateInfo)` | 报告分支更新信息 |
| **exception** | `Output(Bool())` | 输出异常 |
| **rob_pnr_idx** | `Output(UInt((p.robAddrSz).W))` | |
| **rob_head_idx** | `Output(UInt((p.robAddrSz).W))` | rob 中的后备和头部索引 |
| **release** | `Flipped(new DecoupledIO(new TLBundle(edge.bundle)))` | 处理缓存协议的释放操作 |
| **force_order** | `Output(Bool())` | 强制顺序控制，在保证 l/s 顺序的时候激活 |
| **order** | `Input(Bool())` | 顺序控制信号，表示当前是否满足顺序要求 |
| **perf** | `Input(new Bundle {` | |
| **acquire** | `Bool()` | |
| **release** | `Bool()` | |

## 4.9 LSUCoreIO

LSU 的核心 IO 接口，与 execution、dcache、rob 等进行交互

IO 端口定义如下：

| Name | Type | Description |
|---|---|---|
| **data** | `UInt(xLen.W)` | |
| **agen** | `Flipped(Vec(p.lsuWidth, Valid(new MemGen)))` | 地址生成器的输入 |
| **dgen** | `Flipped(Vec(p.memWidth + 1, Valid(new MemGen)))` | 数据生成器的输入,memWidth 为内存宽度 |
| **bypassable** | `Bool()` | |
| **speculative_mask** | `UInt(aluWidth.W)` | |
| **rebusy** | `Bool()` | |
| **iwakeups** | `Vec(p.lsuWidth, Valid(new Wakeup))` | LSU 的唤醒信号 |
| **iresp** | `Vec(p.lsuWidth, Valid(new ExeUnitResp(xLen)))` | |
| **fresp** | `Vec(p.lsuWidth, Valid(new ExeUnitResp(xLen)))` | |

| | | |
|---|---|---|
| **sfence** | `Flipped(Valid(new rocket.SFenceReq))` | 传递 sfence 的请求，用于内容屏蔽 |
| **dis_uops** | `Vec(p.coreWidth, Valid(new MicroOp))` | 解码后的微操作 |
| **dis_ldq_idx** | `Output(Vec(p.coreWidth,UInt((1+ldpAddrSz).W)))` | |
| **dis_stq_idx** | `Output(Vec(p.coreWidth,UInt((1+stqAddrSz).W)))` | |
| **ldq_full** | `Output(Vec(p.coreWidth,Bool()))` | |
| **stq_full** | `Output(Vec(p.coreWidth,Bool()))` | |
| **commit** | `Input(new CommitInfo)` | 提交信号 CommitInfo 是一个 Bundle，包含了提交的指令信息 |
| **commitLoadAtRobHead** | `Input(Bool())` | 当前是否在 rob 的头部进行加载 |
| **clr_busy_bit** | `Output(Bool())` | 清除 busybit 的信号 |
| **clr_unsafe** | `Output(Bool())` | 清除不安全的加载状态 |
| **fence_dmem** | `Input(Bool())` | 控制内存屏障操作 |
| **bradate** | `Input(new bradateInfo)` | 分支更新信息 |
| **rob_head_idx** | `Input(UInt((p.robAddrSz).W))` | rob 的头部索引 |
| **rob_pnr_idx** | `Input(UInt((p.robAddrSz).W))` | rob 的后备索引 |
| **exception** | `Input(Bool())` | 异常信号 |
| **Fencei_ready** | `Output(Bool())` | 表示 FENCEI 是否准备好 |
| **load_excep** | `Output(Valid(new Exception))` | 表示加载异常的信号 |
| **tsc_reg** | `Input(UInt())` | 时间戳寄存器(计数器)的值 |
| **status** | `Input(new rocket.MStatus)` | 接收机器模式状态寄存器的值 |
| **bp** | `Input(Vec(breakpoint_num,new rocket.BP))` | 接收断点设置 |
| **mach_context** | `Input(UInt)` | 接收机器模式上下文的值 |
| **s_context** | `Input(UInt)` | 接收超线程模式上下文的值 |
| **perf** | `Output(new Bundle {` | |
| **acquire** | `Bool()` | |
| **release** | `Bool()` | |
| **tlbMiss** | `Bool()` | tlb 可能不需要，此处待定 |

## 4.10 LSUIO

提供 LSU 的核心、内存（TLB）的 IO 接口

IO 端口定义如下：

| Name | Type | Description |
|---|---|---|
| **ptw** | new rocket.TLBPTWIO | TLB 的输入输出接口,待定 |
| **core** | new LSUCoreIO | LSU 的核心接口 |
| **dcache_mem** | new LSUMemIO | LSU 的内存接口 |

# 5 Microarchitecture

本处理器为乱序执行多发射 RV32IM 架构，其设计主要借鉴于 RSD 以及 BOOM。

## 5.1 BranchPredictor

BranchPredictor 的构造参数如下：

| Name | Type | Description |
|---|---|---|
| **fetch_width** | Int | val GHR_width : Int |

IO 端口定义如下：

| Name | Type | Description |
|---|---|---|
| **io** | IO(new BP_IO(fetch_width, GHR_width)) | |

## 5.2 PayloadRAM

PayloadRAM 是一个 RAM 模块，用于存储指令的有效载荷。

IO 端口定义如下：

| Name | Type | Description |
|---|---|---|
| **io** | IO(new Bundle { | |
| **we** | Input(Vec(p.DISPATCH_WIDTH, Bool())) | 写使能信号 |
| **waddr** | Input(Vec(p.DISPATCH_WIDTH, UInt(log2Ceil(NUM_PayloadRAM).W))) | 写地址 |
| **wdata** | Input(Vec(p.DISPATCH_WIDTH, UInt(Insr_WIDTH.W))) | 写数据 |
| **raddr** | Input(Vec(p.ISSUE_WIDTH, UInt(log2Ceil(NUM_PayloadRAM).W))) | 读地址 |
| **rdata** | Output(Vec(p.ISSUE_WIDTH, UInt(Insr_WIDTH.W))) | 读数据 |

## 5.3 WakeupLogic

WakeupLogic 是一个唤醒逻辑模块，用于处理指令的唤醒信号。它根据指令的依赖关系和状态，决定哪些指令可以被唤醒。

IO 端口定义如下：

| Name | Type | Description |
|---|---|---|

| | | |
|---|---|---|
| **io** | `IO(new Bundle {` | |
| **stall** | `Input(Bool())` | 停顿信号 |
| **write** | `Input(Vec(p.DISPATCH_WIDTH, Bool()))` | 写使能 |
| **writePtr** | `Input(Vec(p.DISPATCH_WIDTH, UInt(p.ISSUE_QUEUE_INDEX_WIDTH.W)))` | 写指针 |
| **writeSrcTag** | `Input(Vec(p.DISPATCH_WIDTH, new SrcTag()))` | 写源标签,SrcTag 可能包含寄存器标签与寄存器地址 |
| **writeDstTag** | `Input(Vec(p.DISPATCH_WIDTH, new DstTag()))` | 写目标标签,DstTag 可能包含寄存器标签 |
| **wakeup** | `Input(Vec(p.WAKEUP_WIDTH, Bool()))` | 唤醒信号 |
| **wakeupDstTag** | `Input(Vec(p.WAKEUP_WIDTH, new DstTag()))` | 唤醒目标标签 |
| **wakeupVector** | `Input(Vec(p.WAKEUP_WIDTH + p.ISSUE_STORE_WIDTH, Vec(p.ISSUE_QUEUE_ENTRY_NUM, Bool())))` | 唤醒向量 |
| **notIssued** | `Input(Vec(p.ISSUE_QUEUE_ENTRY_NUM, Bool()))` | 未发射标志 |
| **dispatchStore** | `Input(Vec(p.DISPATCH_WIDTH, Bool()))` | 是否是 Store |
| **dispatchLoad** | `Input(Vec(p.DISPATCH_WIDTH, Bool()))` | 是否是 Load |
| **memDependencyPred** | `Input(Vec(p.DISPATCH_WIDTH, Bool()))` | 内存依赖预测 |
| **opReady** | `Output(Vec(p.ISSUE_QUEUE_ENTRY_NUM, Bool()))` | 操作就绪标志 |

## 5.4 RenameUnit

重命名单元将逻辑寄存器地址映射成实际寄存器。逻辑寄存器指的是 ISA 定义的 x0-x31，而实际寄存器数量多于 32 个，一般可达 128 个。主要解决 WAW，WAR 等问题。

IO 端口定义如下：

| Name | Type | Description |
|---|---|---|
| **io** | `IO(new Bundle {` | |
| **in** | `Flipped(Vec(p.RENAME_WIDTH, DecodedInstr))` | 输入 |
| **out** | `Vec(p.RENAME_WIDTH, RenamedInsr)` | soaeuaoeu |

## 5.5 ROB

重命名缓冲区，主要用于存储指令的执行结果。它是一个 FIFO 结构，先进先出。指令在执行完成后，将结果写入 ROB 中。ROB 中的数据可以被其他指令读取，从而实现数据的共享和重用。

ROB 的构造参数如下：

| Name | Type | Description |
| --- | --- | --- |
| **num_WakeupPorts** | `Int` | |

IO 端口定义如下：

| Name | Type | Description |
| --- | --- | --- |
| **io** | `IO(new Bundle {` | |
| **dispatch_in** | `Flipped(Vec(p.DISPATCH_WIDTH, DispatchedInsr))` | 来自 Dispatch Unit 的输入 |
| **branch_in** | `Flipped(Vec(p.BRANCH_WIDTH, BranchInfo))` | 来自 Branch Predictor 的输入 |
| **exception_fetch_pc** | `Input(UInt(p.ADDR_WIDTH.W))` | 异常发生时的 PC 值 |
| **ROB_head_idx** | `Output(UInt(p.ROB_ADDR_WIDTH.W))` | ROB 头指针 |
| **ROB_tail_idx** | `Output(UInt(p.ROB_ADDR_WIDTH.W))` | ROB 尾指针 |
| **ROB_pnr_idx** | `Output(UInt(p.ROB_ADDR_WIDTH.W))` | ROB 安全上限指针 |
| **ROB_empty** | `Output(Bool())` | ROB 空标志 |
| **ROB_ready** | `Output(Bool())` | ROB 准备好标志(不仅仅是非满) |
| **WB_resps** | `Flipped(Vec(num_WakeupPorts, Valid(?待确认?)))` | 来自执行单元的写回响应 |
| **LSU_clr_busy** | `Input(Vec(p.DISPATCH_WIDTH), Valid(UInt(p.ROB_ADDR_WIDTH.W)))` | 来自 LSU 的清除忙标志 |
| **LSU_clr_unsafe** | `Input(Vec(p.LSU_WIDTH), Valid(UInt(p.ROB_ADDR_WIDTH.W)))` | 来自 LSU 的清除不安全标志 |
| **LSU_exception** | `Input(Valid(new Exception()))` | 来自 LSU 的异常标志(Exception 类待定义) |
| **CSR_replay** | `Input(Valid(new Exception()))` | 来自 CSR 的重放标志(Exception 类待定义) |
| **CSR_stall** | `Input(Bool())` | 来自 CSR 的停顿标志 |
| **commit** | `Output(new Commit_Info())` | 提交信息(Commit_Info 类待定义) |
| **rollback** | `Output(Bool())` | 回滚标志 |

| | | |
|---|---|---|
| **commit_exception** | `Output(Valid(new Commit_exception_Info()))` | 提交异常信息（To CSR）(Commit_exception_Info 类待定义) |
| **flush** | `Output(Valid(new Commit_exception_Info()))` | 刷新标志(Commit_exception_Info 类待定义) |
| **flush_frontend** | `Output(Bool())` | 前端刷新标志 |

## 5.6 DestinationRAM

IO 端口定义如下：

| Name | Type | Description |
|---|---|---|
| **io** | `IO(new Bundle {` | |
| **wrEn** | `Input(Bool())` | |
| **wrReg** | `Input(UInt(params.phyRegBits.W))` | |
| **wrIQIdx** | `Input(UInt(params.iqIdxBits.W))` | |
| **rdReg1** | `Input(UInt(params.phyRegBits.W))` | |
| **rdIQIdx1** | `Output(UInt(params.iqIdxBits.W))` | |
| **ram** | `SyncReadMem(params.phyRegNum, UInt(params.iqIdxBits.W))` | |

## 5.7 ReadyBitTable

IO 端口定义如下：

| Name | Type | Description |
|---|---|---|
| **io** | `IO(new Bundle {` | |
| **init** | `Flipped(Valid(new Bundle {` | ... })) |
| **setReady** | `Flipped(Valid(new Bundle {` | ... })) |
| **readyBits** | `Output(Vec(params.iqEntries, Vec(2, Bool())))` | |
| **readyBits** | `RegInit(VecInit(Seq.fill(params.iqEntries)(VecInit(false.B, false.B))))` | |

## 5.8 ReadyBitTable

IO 端口定义如下：

| Name | Type | Description |
|---|---|---|
| **io** | `IO(new Bundle {` | |
| **init** | `Flipped(Valid(new Bundle {` | ... })) |

| | | |
|---|---|---|
| **setReady** | `Flipped(Valid(new Bundle {` | ... `}))` |
| **readyBits** | `Output(Vec(params.iqEntries, Vec(2, Bool())))` | |
| **readyBits** | `RegInit(VecInit(Seq.fill(params.iqEntries)(VecInit(false.B, false.B))))` | |

## 5.9 WAT

IO 端口定义如下：

| Name | Type | Description |
|---|---|---|
| **io** | `IO(new Bundle {` | |
| **addEntry** | `Flipped(Valid(new Bundle {` | ... `}))` |
| **wakeup** | `Flipped(Valid(UInt(params.phyRegBits.W)))` | |
| **setReady** | `Valid(new Bundle {` | ... `})` |
| **entries** | `Reg(Vec(params.watEntries, new Bundle {` | ... `}))` |

## 5.10 ALU

IO 端口定义如下：

| Name | Type | Description |
|---|---|---|
| **io** | `IO(new ALUIO)` | |

## 5.11 BypassNetwork

IO 端口定义如下：

| Name | Type | Description |
|---|---|---|
| **io** | `IO(new BypassNetworkIO)` | |

## 5.12 LSU

LSU 的模块定义，目前只完成了 IO 接口的定义，内部逻辑还未完成

IO 端口定义如下：

| Name | Type | Description |
|---|---|---|
| **io** | `IO(new LSUIO)` | 定义 LSU 的 IO 接口 |