

目录

[快速预览简介 2](#_Toc16840)

[1 项目概述 3](#_Toc8539)

[1.1 项目背景 3](#_Toc5406)

[1.2 设计目标 3](#_Toc13204)

[1.3 设计平台说明 3](#_Toc5984)

[2 RISC-V CPU 架构设计 3](#_Toc26911)

[2.1 指令集支持 5](#_Toc3177)

[2.2 执行模块 5](#_Toc26922)

[2.2.1 取指模块(IFU) 5](#_Toc3712)

[2.2.2 译码模块(IDU) 6](#_Toc32099)

[2.2.3 算数逻辑模块（ALU） 6](#_Toc7927)

[2.2.4 地址生成模块（AGU） 7](#_Toc3762)

[2.2.5 访存模块（LSU） 7](#_Toc9359)

[2.2.6 写回模块（WBU） 7](#_Toc19581)

[2.3 数据通路 8](#_Toc3604)

[2.4 寄存器堆 9](#_Toc31579)

[2.5 流水线 10](#_Toc17109)

[2.5.1 流水线总线 10](#_Toc3622)

[2.5.2 流水线状态机 10](#_Toc6116)

[2.5.3 流水线控制器 12](#_Toc1222)

[2.6 总线信号表 13](#_Toc30782)

[2.6.1 各个总线的信号详述 13](#_Toc9774)

[2.6.2 各个枚举类型详述 16](#_Toc8010)

[3 性能优化 18](#_Toc5982)

[3.1 时序优化 18](#_Toc90)

[3.2 访存优化 18](#_Toc8046)

[4 特色功能 18](#_Toc2825)

[4.1 AXI 总线 18](#_Toc10472)

**快速预览简介**

本文面向全国大学生集成电路创新大赛——竞业达杯需求，设计并实现了一款基于 RISC-V 32I 指令集的五级流水线CPU架构。该处理器采用推测执行策略，包含取指、译码、执行（集成ALU/AGU）、访存及写回五大核心模块，通过解耦总线与翻转总线架构实现模块间低耦合通信。关键技术突破包括：（1）创新性提出分支状态机与多源仲裁机制，有效解决指令分发冲突与写回竞争问题；（2）构建动态流水线控制器，通过valid-ready握手协议实现stall/flush信号智能生成，提升异常处理效率；（3）设计异步寄存器堆架构（读组合逻辑+写时序逻辑），严格遵循RV32I标准同时支持CSR扩展。功能验证表明：RTL级仿真通过率100%，支持基础指令集全功能运行。本设计为后续扩展浮点单元、定制指令集及硬件加速模块奠定了可扩展架构基

设计报告 —

# 1.项目概述

## 1.1 项目背景

本文档介绍了**逆舟冰箱小队**的 RISCV 32I 五级流水线处理器设计

## 1.2 设计目标

本项目旨在设计一款基于 RISC-V 32I 指令集的五级流水线 CPU ，支持基本指令集的全功能运行。该处理器采用推测执行策略，包含取指、译码、执行（集成 ALU/

AGU）、访存及写回五个流水阶，通过解耦总线与翻转总线架构实现模块间低耦合通信。

## 1.3 设计平台说明

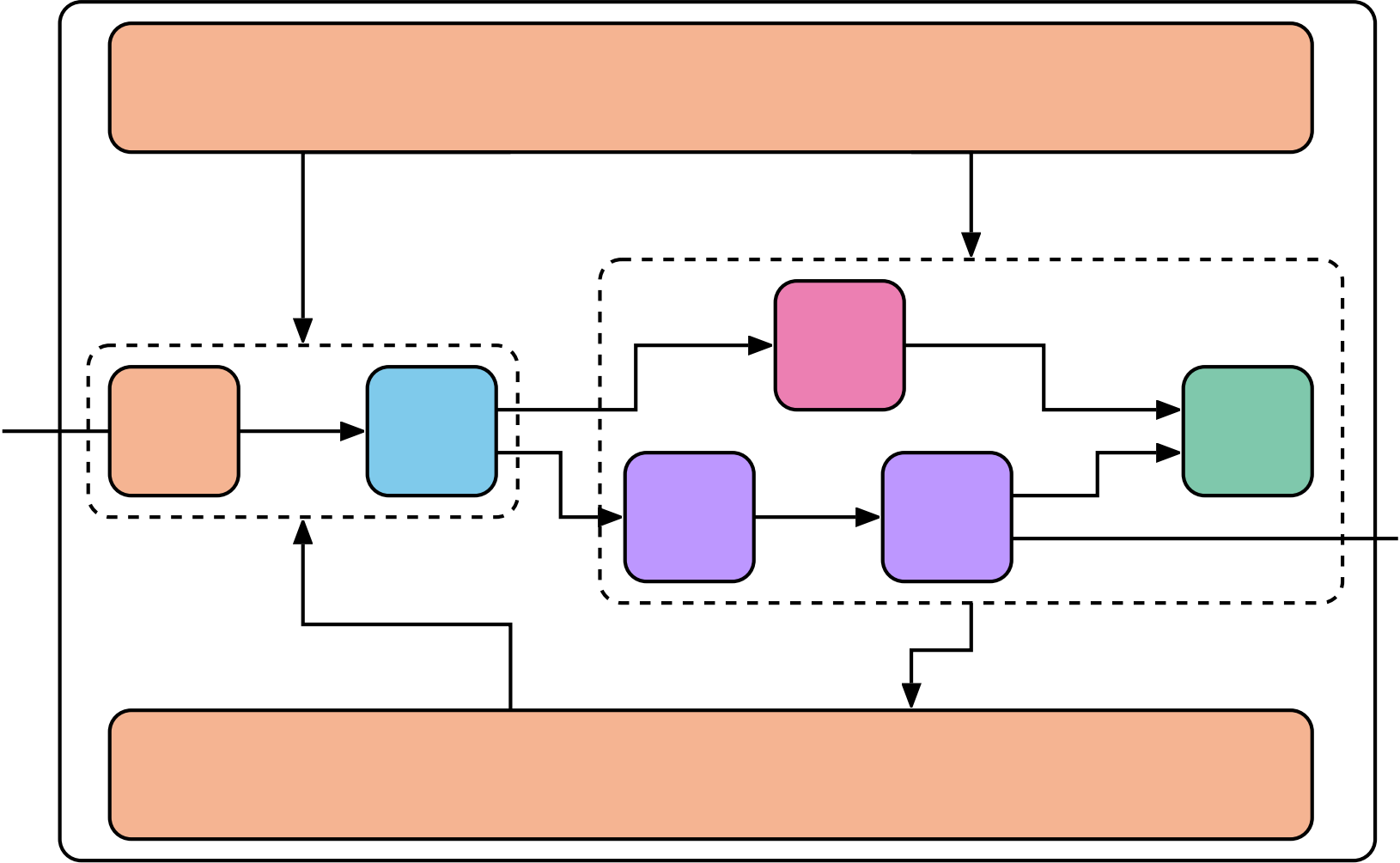
本项目基于 Vivado 2023.2进行设计，使用 Verilog HDL语言进行描述，采用

Xilinx xc7k325tffg900-2 FPGA 作为目标平台。设计中 ip 核使用了 System Verilog和

VHDL描述。本项目同时维护一个实际上板版本，以及一个孪生平台版本。前者用于本地，后者则用于云端的 FPGA 测试。

# 2.RISC-V CPU 架构设计

BUS\_IROM

**Pipeline\_ctrl**

**ALU**

**IDU**

**WBU**

**IFU**

**AGU**

**LSU**

**Front**

**Back**

**Regfile**

BUS\_DRAM

**CPU**

图 1-CPU 设计框图

上述设计图没有具体地规定模块之间的相应接口，但清晰地刻画了各个模块之间 的关系和联系，表现了数据流在各个模块之间的大致方向。

根据以上设计图，我们的 CPU 分成以下主要模块：

• 顶层模块（CPU）

• 取值单元（[IFU](#bookmark9)）

• 指令译码单元（[IDU](#bookmark10)）

• 算数逻辑单元（[ALU](#bookmark11)）

• 地址生成单元（[AGU](#bookmark12)）

• 访存单元（[LSU](#bookmark13)）

• 写回单元（[WBU](#bookmark14)）

• 寄存器堆（[RegFile](#bookmark16)）

• 流水线控制器（[Pipeline\_Ctrl](#bookmark17)）

其中 IFU 和 IDU 为前端，ALU 、AGU 、LSU 和 WBU 为后端，共同组成了执行单元（[EXU](#bookmark8)）。

接下来我们将对各个模块的信号名称、方向和功能进行详细说明。

其中方向有五种类型

• Input输入信号，数据流向模块内部

• Output输出信号，数据从模块流出

• Bundle信号捆绑，包含多个子信号的复合类型

• Decoupled解耦总线，包含 valid 、ready 握手协议的数据传输总线

• Flipped翻转总线，Decoupled 的镜像，通常用于接收端

顶层模块是整个 CPU 的最外层封装，用于连接各个功能模块，协调数据流动和控制信号传递。以下是顶层模块的接口信号：

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** |  | **功能描述** |
| clock | Input |  | 系统时钟信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| reset | Input |  | 高电平有效的同步复位信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| IROM | Bundle |  | IROM 总线 |

|  |  |  |  |
| --- | --- | --- | --- |
| DRAM | Bundle |  | DRAM 总线 |

表 1-顶层模块的接口信号

## 2.1 指令集支持

RV32I ，或者可选的 RV32E

## 2.2 执行模块

下面将对各个执行器接口方向和定义分别进行详细说明。其中若无特殊说明，每 个模块均隐含一个输出总线 Pipeline\_ctrl ，用于传递给流水线控制器必要的数据

我们会在最后一个子项中集中描述各个总线的具体定义。

**2.2.1 取指模块(IFU)**

取指模块的任务是通过 IROM 总线从 IROM 中读取指令并传递给后续模块，我们的处理器采取推测执行策略，因此 PC 存放在 IFU 中并在每一次取指完成之后+4，然而推测有可能错误，因此当执行错误时，需要冲刷指令，并且通过 WBU 传递的总线 数据将 PC 更新为正确的值。

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** |  | **功能描述** |
| clock | Input |  | 系统时钟信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| reset | Input |  | 高电平有效的同步复位信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| IROM | Bundle |  | IROM 总线 |

|  |  |  |  |
| --- | --- | --- | --- |
| WBU\_2\_IFU | Flipped |  | WBU 到 IFU 的总线信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| IFU\_2\_IDU | Decoupled |  | IFU 到 IDU 的总线信号 |

表 2-取指模块的接口信号

### 2.2.2 译码模块(IDU)

译码模块的任务是将指令转译为各个模块可以直接使用的控制信号，并将指令根 据类型分发给 ALU 或者 AGU，同时，其还会根据指令需要的操作数从 GPR，CSR，IMM， PC 中选择两个作为实际需要的操作数。

为了获取这些数据，IDU 需要根据指令提取出立即数，并通过总线将需要读取的 GPR 和 CSR 地址传递给 RegFile，以读取其数据。

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** |  | **方向** | **功能描述** |
| clock |  | Input | 系统时钟信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| reset |  | Input | 高电平有效的同步复位信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| REG\_2\_IDU |  | Bundle | REG 到 IDU 的总线信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| IDU\_2\_REG |  | Bundle | IDU 到 REG 的总线信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| IFU\_2\_IDU |  | Flipped | IFU 到 IDU 的总线信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| IDU\_2\_EXU |  | Decoupled | IDU 到 EXU ，也即 AGU 或 ALU 的总线信号 |

表 3-译码模块的接口信号

### 2.2.3 算数逻辑模块（ALU）

算数逻辑模块的任务时根据需要执行加法，减法，逻辑与，逻辑或，异或等操作，并将结果通过总线传递给 WBU 模块。ALU 模块的输入信号包括来自 IDU 的指令和操作数。输出信号则是运算结果和状态标志位。

|  |  |  |
| --- | --- | --- |
| **信号名称** | **方向** | **功能描述** |
| clock | Input | 系统时钟信号 |

|  |  |  |
| --- | --- | --- |
| reset | Input | 高电平有效的同步复位信号 |

|  |  |  |
| --- | --- | --- |
| IDU\_2\_EXU | Flipped | IDU 到 ALU 的总线信号 |

|  |  |  |
| --- | --- | --- |
| EXU\_2\_WBU | Decoupled | ALU 到 WBU 的总线信号 |

表 4-算数逻辑模块的接口信号

### 2.2.4 地址生成模块（AGU）

地址生成模块的任务是根据指令计算出内存地址，并将地址通过总线传递WBU 模块。

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** |  | **功能描述** |
| clock | Input |  | 系统时钟信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| reset | Input |  | 高电平有效的同步复位信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| IDU\_2\_EXU | Flipped |  | IDU 到 AGU 的总线信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| AGU\_2\_LSU | Decoupled |  | AGU 到 LSU 的总线信号 |

表 5-地址生成模块的接口信号

### 2.2.5 访存模块（LSU）

地址生成模块的任务是根据指令计算出内存地址，并将地址通过总线传递给 WBU 模块。

|  |  |  |
| --- | --- | --- |
| **信号名称** | **方向** | **功能描述** |
| clock | Input | 系统时钟信号 |

|  |  |  |
| --- | --- | --- |
| reset | Input | 高电平有效的同步复位信号 |

|  |  |  |
| --- | --- | --- |
| AGU\_2\_LSU | Flipped | AGU 到 LSU 的总线信号 |

|  |  |  |
| --- | --- | --- |
| EXU\_2\_WBU | Decoupled | LSU 到 WBU 的总线信号 |

|  |  |  |
| --- | --- | --- |
| DRAM | Bundle | DRAM 总线信号 |

表 6-访存生成模块的接口信号

### 2.2.6 写回模块（WBU）

写回模块的任务是将运算结果写回寄存器堆中。WBU 模块的输入信号包括来自ALU 和 LSU 的运算结果和状态标志位。输出信号则是写回寄存器堆的控制信号和数据。

|  |  |  |
| --- | --- | --- |
| **信号名称** | **方向** | **功能描述** |
| clock | Input | 系统时钟信号 |

|  |  |  |
| --- | --- | --- |
| reset | Input | 高电平有效的同步复位信号 |

|  |  |  |
| --- | --- | --- |
| EXU\_2\_WBU | Flipped | EXU ，即 ALU 或 LSU 到 WBU 的总线信号 |

|  |  |  |
| --- | --- | --- |
| WBU\_2\_IFU | Decoupled | WBU 到 IFU 的总线信号 |

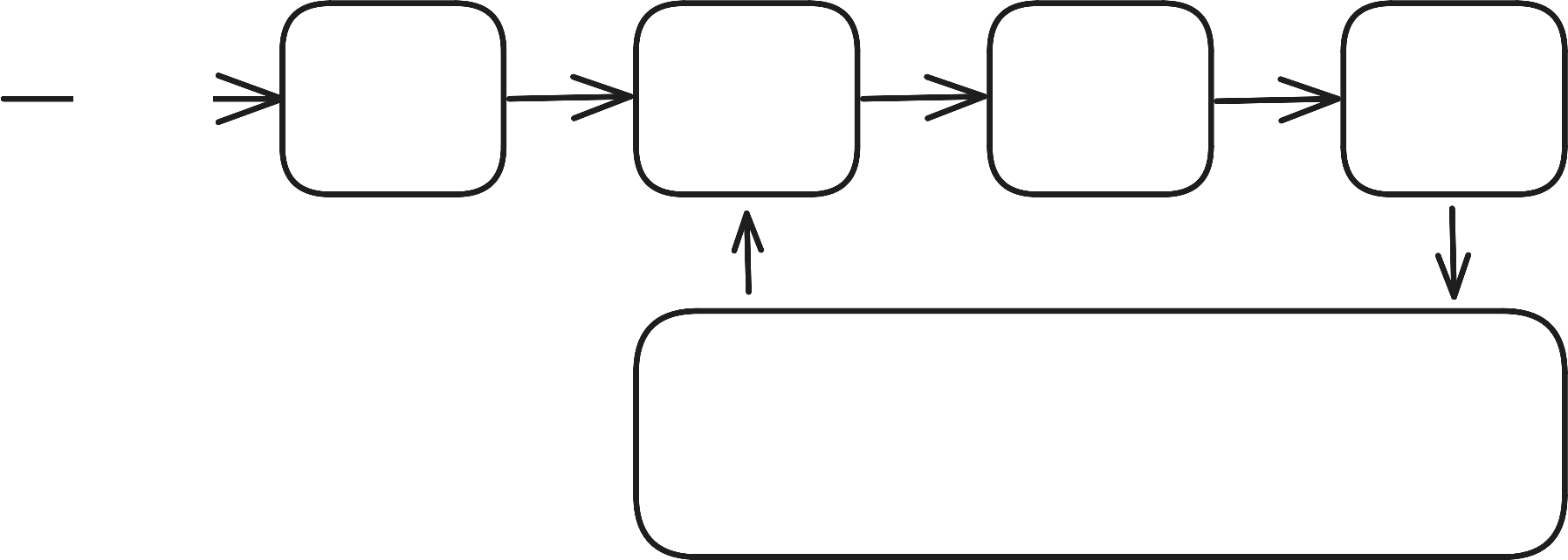
|  |  |  |
| --- | --- | --- |
| WBU\_2\_REG | Bundle | Regfile 写入总线信号 |

表 7-写回模块的接口信号

## 2.3 数据通路

对于要求实现的 RV32I 中的 37 条指令，我们将其分为两类，分别走在不同的流水线上

• 逻辑处理指令(add, and, lui, sll 等)

Stage1 IFU

Stage2 I DU

Stage3

ALU

Stage4

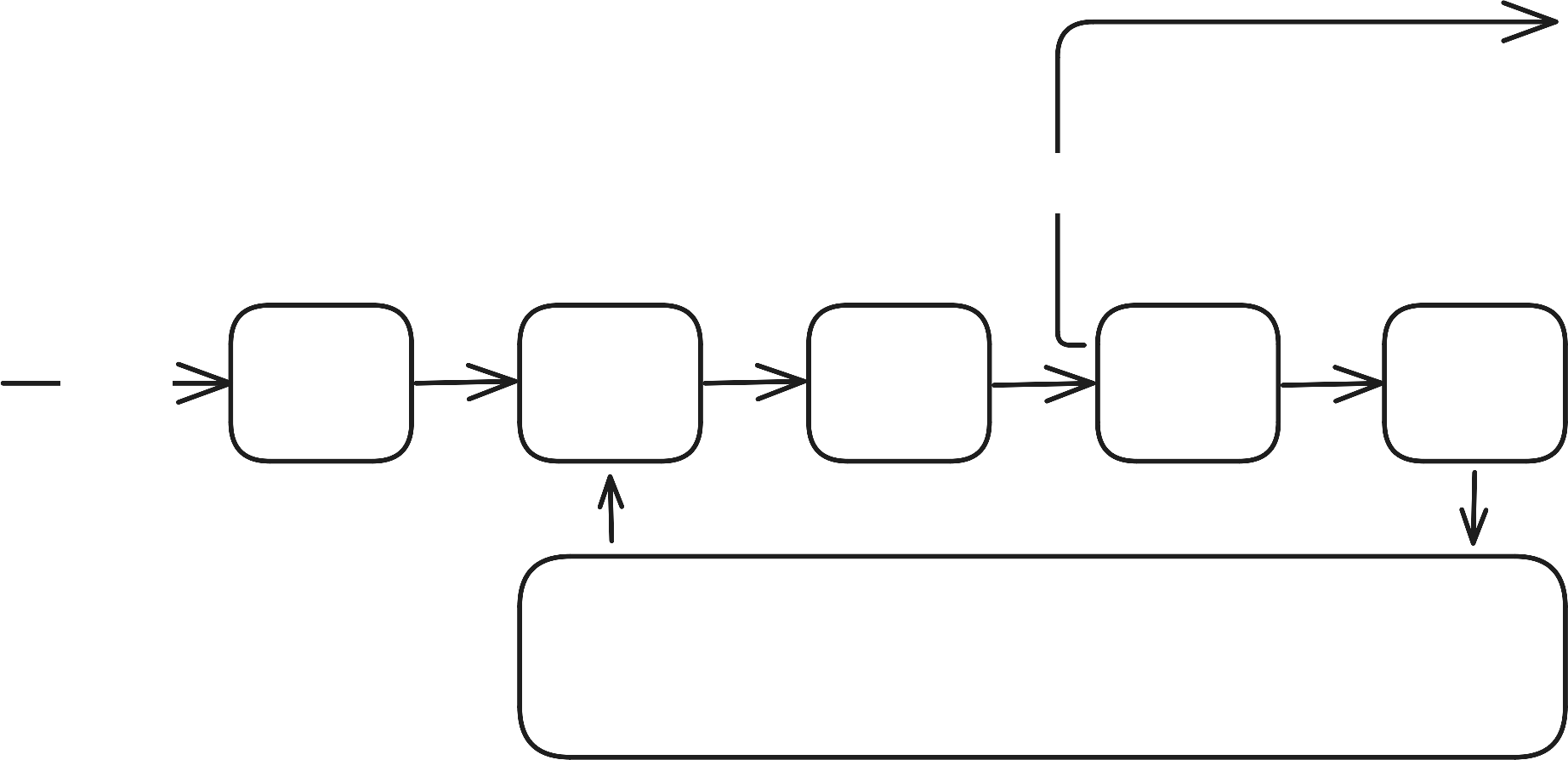
WBU

I ROM

REG\_File

图 2-逻辑处理指令数据通路

• 分支指令

DRAM

LSU

Stage1 IFU

Stage3

AGU

Stage5

WBU

I ROM

Stage2 I DU

REG\_File

图 3-访存指令数据通路

可以观察到，这与教科书上的五级流水线不同，将后端的逻辑处理流水和访存流水分离，从而避免了逻辑处理指令必须在访存阶段空转而浪费周期。

对于 riscv 的访存地址，我们只需要将两个源数据(两个寄存器或者寄存器与立即数，不过 AGU 不需要关心来源，这些工作会被 IDU 完成)相加，因此 AGU 占用的资源相较于一个完整 ALU 非常少。

## 2.4 寄存器堆

模块 RegFile 包含 rv32i 定义的标准寄存器堆，即 32 个通用寄存器(GPR)和部分系 统控制寄存器(CSR)。

所有读操作都是组合逻辑的，当前周期即可获取数值，而写操作则是时序逻辑的，当前周期的写操作会在下一个周期生效。

不过寄存器堆中并不包含程序计数器(PC) ，为了方便进行推测执行以及后续会添加的分支预测，其目前被存放在 IFU 中。

寄存器堆的输入输出端口定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** |  | **方向** | **功能描述** |
| clock |  | Input | 系统时钟信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| reset |  | Input | 高电平有效的同步复位信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| REG\_2\_IDU |  | Bundle | REG 到 IDU 的总线信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| WBU\_2\_REG |  | Bundle | WBU 到 REG 的总线信号 |

表 8-寄存器堆的接口信号

至于具体的各个寄存器定义可参照 RV32I 的 ISA 文档

## 2.5 流水线

我们将在这里描述流水线的状态转移方程和模块 PipelineCtrl 的接口与逻辑。

### 2.5.1 流水线总线

本处理器流水线采用 valid-ready 握手机制，流水线控制器对每一个流水线寄存器 都传入一个 PipeCtrl 总线

Pipectrl 的定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** |  | **功能描述** |
| stall | Output |  | 流水线阻塞信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| flush | Output |  | 流水线冲刷信号 |

表 9-寄存器堆的接口信号

### 2.5.2 流水线状态机

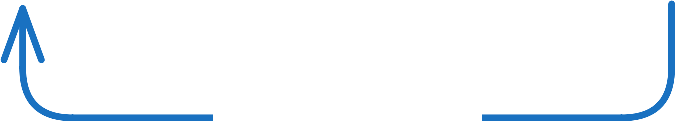
考虑两个模块之间的流水线状态机，分别将其命名为 prev 和 this ，数据方向从 prev 流向 this。

定义 prev 的数据输出端口为 prevOut ，this 的数据输入端口为 thisIn ，数据输出端 口为 thisOut

**1 对** **1 流水线**

那么 thisIn.valid 的状态转移方程如下：

flush



true

false

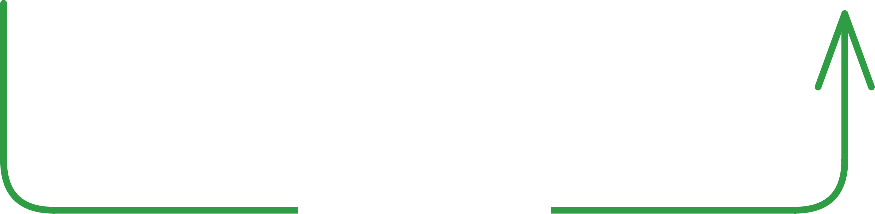
thisOut.fire

prevOut.fire



flush

thisOut.fire



prevOut.fire

图 4-thisIn.valid 的状态转移方程

其中状态转移条件优先级为  >  >  > 

阶段之间的通信使用握手机制，通常涉及 valid 和 ready 信号。当前阶段（this）的 ready 信号表示其接收新数据的能力。其状态由两个因素决定：

1.**下游就绪:**紧随其后的阶段（next ）必须准备好接收数据（即 next.ready必须为高）。这实现了反压，防止数据溢出。

2.**流水线控制:**流水线控制逻辑不得为当前阶段置位 stall 信号。stall信号通常会暂停流水线阶段的操作。

因此，当且仅当下游阶段就绪且当前阶段未被暂停时，当前阶段的 ready 信号才 会被置位： this.ready = next.ready and not stall

**数据锁存:**

数据从前一个阶段（prev）传输到当前阶段（this）发生在握手成功时。当上一个 阶段有有效数据（prev.valid 为高，隐含）且当前阶段准备好接收数据（this.ready为高）时，握手被认为是成功的。当这个条件（prev.valid and this.ready）在一个时钟周期内满足时，prev 阶段输出的数据被捕获并存储在与 this 阶段相关的流水线寄存器中。这确保了数据同步进行，并且仅在条件允许时进行。

**1 对** **n 流水线**

在 1 对 n 流水线场景中，数据从单个前驱阶段（prev）根据特定条件分发到 n 个可能的后继阶段（this\_ 1 到 this\_n）中的一个。一个典型的例子是指令译码单元（IDU）

根据指令类型将数据发送到算术逻辑单元（ALU）或地址生成单元（AGU）。

这种分发逻辑的关键在于：

1.**条件路由:** 每个目标分支都有一个关联的布尔条件（cond）。

2.**数据传输:** 数据（prevOut.bits）仅在源阶段有效（prevOut.valid）、源阶段就绪

（prevOut.ready）且目标分支的条件（cond）为真时，才锁存到该目标分支的输入寄

存器（branchIn.bits）。这对应于 prevOut.fire && cond。

3.**源就绪信号** ( prevOut.ready ): 源阶段的 ready 信号取决于下游。它仅在流水线未暂停（ctrl.stall 为假）且至少有一个满足其条件（cond 为真）的目标分支准备好接收数据 （branchIn.ready 为真）时，才置为高。这确保了反压机制能正确传递回源阶段。

4.**目标有效信号** ( branchIn.valid ): 每个目标分支输入的 valid 信号由一个状态机管理。

•当数据成功从源传输到该分支时（prevOut.fire && cond），valid置为真。

•当流水线被冲刷（ctrl.flush）或该分支的数据被其后续阶段消耗掉 （branchOut.fire）时，valid置为假。

•在其他情况下，valid 保持其先前状态。

•复位时，valid 默认为假。

**n 对** **1 流水线**

在设计多对一（n-to-1）流水线数据通路时，例如当多个功能单元（如 LSU和 ALU）的输出需要汇聚到同一个目标单元或流水线阶段时，必须使用仲裁器

（Arbiter）。仲裁器的作用是根据预设的优先级或调度策略，从多个请求源中选择一个，授予其访问共享资源的权限，确保数据在每个周期内有序且无冲突地传输。

例如，若 LSU 和 ALU 都需要将结果传递给写回（Write Back）阶段或寄存器堆（Register File）， 则需要在 LSU 输出、ALU输出与目标输入之间实例化一个仲裁器。

### 2.5.3 流水线控制器

流水线控制器的任务是根据各个执行单元的输入输出数据，判断出流水线是否应当阻塞或者冲刷。

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** |  | **方向** | **功能描述** |
| GPR\_read |  | Flipped | GPR 读信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| IFU\_out |  | Flipped | IFU 输出信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| IDU\_in |  | Flipped | IDU 输入信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| ALU\_in |  | Flipped | ALU 输入信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| AGU\_in |  | Flipped | AGU 输入信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| LSU\_in |  | Flipped | LSU 输入信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| WBU\_in |  | Flipped | WBU 输入信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| WBU\_out |  | Flipped | WBU 输出信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| IFUCtrl |  | Bundle | IFU 流水线控制信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| IDUCtrl |  | Bundle | IDU 流水线控制信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| AGUCtrl |  | Bundle | AGU 流水线控制信号 |

|  |  |  |  |
| --- | --- | --- | --- |
| EXUCtrl |  | Bundle | EXU 流水线控制信号 |

表 10-流水线控制器的接口信号

## 2.6 总线信号表

• Bool单 wire

• UInt多位 wire

• Enum枚举类型

### 2.6.1 各个总线的信号详述

|  |  |  |
| --- | --- | --- |
| **信号名称** | **方向** | **功能描述** |
| addr | UInt | 下一条指令地址的地址 |

表 11-WBU\_2\_IFU 总线信号

|  |  |  |
| --- | --- | --- |
| **信号名称** | **方向** | **功能描述** |
| data | UInt | 指令数据 |

|  |  |  |
| --- | --- | --- |
| PC | UInt | 指令地址 |

表 12-IFU\_2\_IDU 总线信号

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **信号名称** |  | **方向** |  | **功能描述** |
| CSR\_rdata |  | UInt |  | CSR 读数据 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| GPR\_Adata |  | UInt |  | GPR\_A 的数据 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| GPR\_Bdata |  | UInt |  | GPR\_B 的数据 |

表 13-REG\_2\_IDU 总线信号

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** |  | **功能描述** |
| Branch | Enum |  | 分支类型 |

|  |  |  |  |
| --- | --- | --- | --- |
| MemOp | Enum |  | 内存操作类型 |

|  |  |  |  |
| --- | --- | --- | --- |
| EXU\_A | UInt |  | 传递给执行单元的源操作数 A |

|  |  |  |  |
| --- | --- | --- | --- |
| EXU\_B | UInt |  | 传递给执行单元的源操作数 B |

|  |  |  |  |
| --- | --- | --- | --- |
| EXUctr | Enum |  | 传递给执行单元的操作类型 |

|  |  |  |  |
| --- | --- | --- | --- |
| csr\_ctr | Enum |  | 对 CSR 的操作类型 |

|  |  |  |  |
| --- | --- | --- | --- |
| Imm | Enum |  | 传递给执行单元的立即数 |

|  |  |  |  |
| --- | --- | --- | --- |
| GPR\_waddr | UInt |  | GPR 写地址 |

|  |  |  |  |
| --- | --- | --- | --- |
| PC | UInt |  | 指令地址 |

表 14-IDU\_2\_EXU 总线信号

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** |  | **功能描述** |
| GPR\_Aaddr | UInt |  | GPR\_A 读地址 |

|  |  |  |  |
| --- | --- | --- | --- |
| GPR\_Baddr | UInt |  | GPR\_B 读地址 |

|  |  |  |  |
| --- | --- | --- | --- |
| CSR\_raddr | UInt |  | CSR 读地址 |

表 15-IDU\_2\_REG 总线信号

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** |  | **功能描述** |
| addr | UInt |  | 内存操作的地址 |

|  |  |  |  |
| --- | --- | --- | --- |
| wdata | UInt |  | 写内存数据 |

|  |  |  |  |
| --- | --- | --- | --- |
| wen | Bool |  | 是否写内存 |

|  |  |  |  |
| --- | --- | --- | --- |
| MemOp | Enum |  | 内存操作类型 |

|  |  |  |  |
| --- | --- | --- | --- |
| PC | UInt |  | 指令地址 |

|  |  |  |  |
| --- | --- | --- | --- |
| GPR\_waddr | UInt |  | GPR 写地址 |

表 16-AGU\_2\_LSU 总线信号

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** |  | **功能描述** |
| Branch | Enum |  | 分支类型 |

|  |  |  |  |
| --- | --- | --- | --- |
| Jmp\_Pc | Enum |  | 跳转地址(若跳转) |

|  |  |  |  |
| --- | --- | --- | --- |
| MemtoReg | Bool |  | 是否读内存 |

|  |  |  |  |
| --- | --- | --- | --- |
| csr\_ctr | Enum |  | 对 CSR 的操作类型 |

|  |  |  |  |
| --- | --- | --- | --- |
| GPR\_waddr | UInt |  | GPR 写地址 |

|  |  |  |  |
| --- | --- | --- | --- |
| PC | UInt |  | 指令地址 |

|  |  |  |  |
| --- | --- | --- | --- |
| CSR\_rdata | UInt |  | CSR 读数据 |

|  |  |  |  |
| --- | --- | --- | --- |
| Result | UInt |  | EXU 计算结果 |

|  |  |  |  |
| --- | --- | --- | --- |
| Mem\_rdata | UInt |  | 读内存数据 |

表 17-EXU\_2\_WBU 总线信号

**2.6.2 各个枚举类型详述**

|  |  |  |
| --- | --- | --- |
| **枚举类型** |  | **描述** |
| Bran\_NJmp |  | 非跳转 |

|  |  |  |
| --- | --- | --- |
| Bran\_Jmp |  | 无条件跳转 |

|  |  |  |
| --- | --- | --- |
| Bran\_Jmpr |  | 寄存器基址跳转 |

|  |  |  |
| --- | --- | --- |
| Bran\_Jeq |  | 相等跳转 |

|  |  |  |
| --- | --- | --- |
| Bran\_Jne |  | 不相等跳转 |

|  |  |  |
| --- | --- | --- |
| Bran\_Jlt |  | 小于跳转 |

|  |  |  |
| --- | --- | --- |
| Bran\_Jge |  | 大于等于跳转 |

|  |  |  |
| --- | --- | --- |
| Bran\_Jcsr |  | CSR 跳转 |

表 18-Branch 枚举类型

|  |  |  |
| --- | --- | --- |
| **枚举类型** | | **描述** |
| MemOp\_ 1BU |  | 无符号 1 字节 |

|  |  |  |
| --- | --- | --- |
| MemOp\_ 1BS |  | 有符号 1 字节 |

|  |  |  |
| --- | --- | --- |
| MemOp\_2BU |  | 无符号 2 字节 |

|  |  |  |
| --- | --- | --- |
| MemOp\_2BS |  | 有符号 2 字节 |

|  |  |  |
| --- | --- | --- |
| MemOp\_4BU |  | 无符号 4 字节 |

表 19-MemOp 枚举类型

|  |  |  |
| --- | --- | --- |
| **枚举类型** |  | **描述** |
| EXUctr\_ADD |  | 加法运算 |

|  |  |  |
| --- | --- | --- |
| EXUctr\_SUB |  | 减法运算 |

|  |  |  |
| --- | --- | --- |
| EXUctr\_Less\_U |  | 无符号小于运算 |

|  |  |  |
| --- | --- | --- |
| EXUctr\_Less\_S |  | 有符号小于运算 |

|  |  |  |
| --- | --- | --- |
| EXUctr\_A |  | 直接输出 A |

|  |  |  |
| --- | --- | --- |
| EXUctr\_B |  | 直接输出 B |

|  |  |  |
| --- | --- | --- |
| EXUctr\_SLL |  | 逻辑左移运算 |

|  |  |  |
| --- | --- | --- |
| EXUctr\_SRL |  | 逻辑右移运算 |

|  |  |  |
| --- | --- | --- |
| EXUctr\_SRA |  | 算数右移运算 |

|  |  |  |
| --- | --- | --- |
| EXUctr\_XOR |  | 异或运算 |

|  |  |  |
| --- | --- | --- |
| EXUctr\_OR |  | 或运算 |

|  |  |  |
| --- | --- | --- |
| EXUctr\_AND |  | 与运算 |

|  |  |  |
| --- | --- | --- |
| EXUctr\_LD |  | 读内存数据 |

|  |  |  |
| --- | --- | --- |
| EXUctr\_ST |  | 写内存数据 |

表 20-EXUctr 枚举类型

|  |  |
| --- | --- |
| **枚举类型** | **描述** |
| CSR\_N | 不做任何事情 |

|  |
| --- |
| CSR\_R1W0 |

|  |
| --- |
| 不读写一， 目前只有 mret 符合 |

|  |  |
| --- | --- |
| CSR\_R1W1 | 一读一写 |

CSR\_R1W2 一读二写， 目前只有 ecall 符合

表 21-csr\_ctr 枚举类型

# 3.性能优化

## 3.1 时序优化

我们对 cpu 的时序做了深度优化，在 nanogate45 工艺，yosys 综合器综合下，频率 能够达到 877.484MHz ，没有latch。

我们主要的优化方式有：

•更合理的任务分配——将源操作数选择交给译码阶段实现，从而提高了 ALU 的极限频率；将访存阶段分为 AG 和 LS 两个阶段，以避免总线仲裁称为时序拼劲。

•更好的译码实现——使用QMC 最优化算法优化译码真值表的实现，从而使译码更加高速。

## 3.2 访存优化

为了减少在高访存延迟条件下依旧能够高效率运行，我们使用 Icache 来加速取指。

Icache 的 set 和 way 参数可参数化分配，使用LRU 算法来进行 cache 的替换，能通过 AXI burst transfer 加速访存

# 4.特色功能

## 4.1 AXI 总线

cpu支持输出标准 AXI4Full 总线，不过目前要求使用官方的IROM 和DRAM 总线，因此删除了不需要的总线与仲裁器。

在使用 AXI 总线的情况下，cpu 可配置特定内存区域为 Icache 缓存区域，此后如 果在这个区域内取指，则会尝试从 Icache 中直接取指。如果 cachemiss 了，则会发起 AXI 总线请求来取指。