# FPGA创新设计大赛 AMD赛道 - 自主选题初级赛道设计报告

**竞赛名称：** 2025年全国大学生嵌入式芯片与系统设计竞赛——FPGA创新设计大赛 AMD赛道  
**赛道类型：** 自主选题 - 初级赛道  
**项目名称：** []  
**提交日期：** [YYYY-MM-DD]

## 摘要

随着人工智能和科学计算的快速发展，稀疏矩阵运算在图神经网络、推荐系统和科学计算等领域发挥着越来越重要的作用。然而，传统的通用处理器在处理稀疏矩阵运算时面临内存访问效率低、计算资源利用率不高等问题。本项目设计了一款面向稀疏矩阵-稠密矩阵乘法 (SpMM) 优化的专用向量处理器，旨在解决图神经网络推理中的计算瓶颈。

本设计采用基于RISC-V向量扩展指令集的自定义架构，设计了针对稀疏矩阵运算的专用指令集。处理器核心采用三级流水线结构，通过自主设计的控制状态机和向量处理单元，有效提升了指令执行效率。关键创新点包括支持稀疏数据压缩存储格式、动态行索引计算机制、针对幂律图的固定行缓存机制、动态跨步加载以及指令双发射机制等，通过减少不必要的数据搬运和优化指令调度，显著提高了SpMM计算效率。

本设计在Xilinx Artix-7 FPGA平台上实现完整验证，工作频率达到150MHz。测试结果显示，在Cora、Citeseer、PubMed等标准图数据集上，相比传统标量处理器实现约3-5倍的性能提升，功耗效率提升约2倍以上。该设计为稀疏线性代数计算提供了高效、可编程的硬件加速方案，在边缘智能和图计算场景具有重要应用价值。

综上，本设计为稀疏线性代数计算提供了一种高效、可编程的硬件加速解决方案，其高能效的特性在边缘智能和图计算等场景中具有重要的应用价值。

**关键词：** 向量处理器,稀疏矩阵-稠密矩阵乘法,图神经网络加速器,RISC-V向量拓展,指令双发射

目录

[FPGA创新设计大赛 AMD赛道 - 自主选题初级赛道设计报告 1](#_Toc212809061)

[摘要 1](#_Toc212809062)

[1. 项目概述 4](#_Toc212809063)

[1.1 选题背景与意义 4](#_Toc212809064)

[1.2 项目目标 5](#_Toc212809065)

[1.3 技术规格 6](#_Toc212809066)

[2. 需求分析与系统设计 6](#_Toc212809067)

[2.1 功能需求分析 6](#_Toc212809068)

[2.2 系统架构设计 9](#_Toc212809069)

[2.3 接口设计 21](#_Toc212809070)

[3. 详细设计与实现 23](#_Toc212809071)

[3.1 核心算法设计 23](#_Toc212809072)

[3.2 关键模块设计 25](#_Toc212809073)

[3.3 资源使用设计 25](#_Toc212809074)

[4. LLM 辅助优化记录 26](#_Toc212809075)

[4.1 设计阶段LLM辅助 26](#_Toc212809076)

[4.2 实现阶段LLM辅助 27](#_Toc212809077)

[4.3 调试阶段LLM辅助 28](#_Toc212809078)

[4.4 LLM辅助总结 28](#_Toc212809079)

[5. 仿真验证与测试 29](#_Toc212809080)

[5.1 仿真环境搭建 29](#_Toc212809081)

[5.2 功能验证 29](#_Toc212809082)

[5.3 时序验证 29](#_Toc212809083)

[5.4 硬件验证（如适用） 29](#_Toc212809084)

[6. 综合实现结果 29](#_Toc212809085)

[6.1 资源使用报告 29](#_Toc212809086)

[6.2 时序性能报告 30](#_Toc212809087)

[6.3 功耗分析（可选） 30](#_Toc212809088)

[7. 创新点与特色 30](#_Toc212809089)

[7.1 设计创新点 30](#_Toc212809090)

[7.2 工程实现特色 30](#_Toc212809091)

[7.3 LLM辅助方法创新 30](#_Toc212809092)

[8. 未来改进方向 30](#_Toc212809093)

[8.1 当前不足 30](#_Toc212809094)

[8.2 改进计划 31](#_Toc212809095)

[8.3 后续发展方向 31](#_Toc212809096)

[9. 结论 31](#_Toc212809097)

[9.1 项目完成情况 31](#_Toc212809098)

[9.2 目标达成度 31](#_Toc212809099)

[9.3 项目价值 31](#_Toc212809100)

[10. 参考文献 31](#_Toc212809101)

[11. 附录 31](#_Toc212809102)

[11.1 系统源代码目录树 31](#_Toc212809103)

[11.2 关键LLM交互记录 32](#_Toc212809104)

## 1. 项目概述

### 1.1 选题背景与意义

图神经网络（GNN）作为处理非欧几里得数据的核心模型，其前向推理核心计算公式为 ，其中 为邻接矩阵表示图拓扑， 为节点特征矩阵（通常稠密或半稀疏）， 为稠密可学习权重，表示非线性激活函数（如 ReLU），如图1。

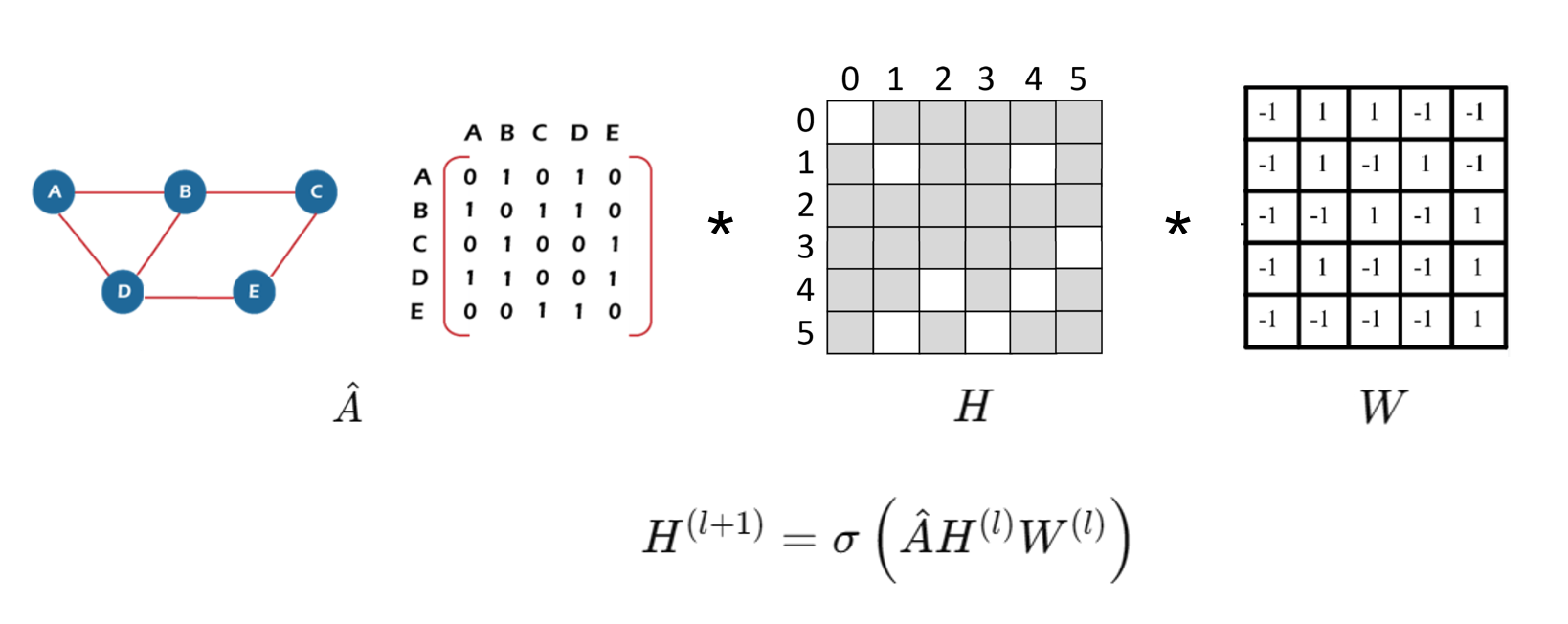


图 1图神经网络数据可视化以及计算公式

该过程包含两次矩阵乘法：XW 生成中间稠密特征 ，随后 执行 SpMM 完成消息聚合，最终通过激活函数 引入非线性。无论 是否稀疏， 始终为 SpMM 操作；若 X 具有稀疏性（如二值化特征或掩码输入）， 也为 SpMM算法。

现实世界图数据普遍遵循幂律分布（Power-law Distribution），即少数"枢纽节点"连接了网络中绝大多数边, 如图2,以PubMed数据集的邻接矩阵举例。Cora 数据集的邻接矩阵非零元素密度仅 0.08%，Citeseer 为 0.13%，Pubmed 低至 0.03%。

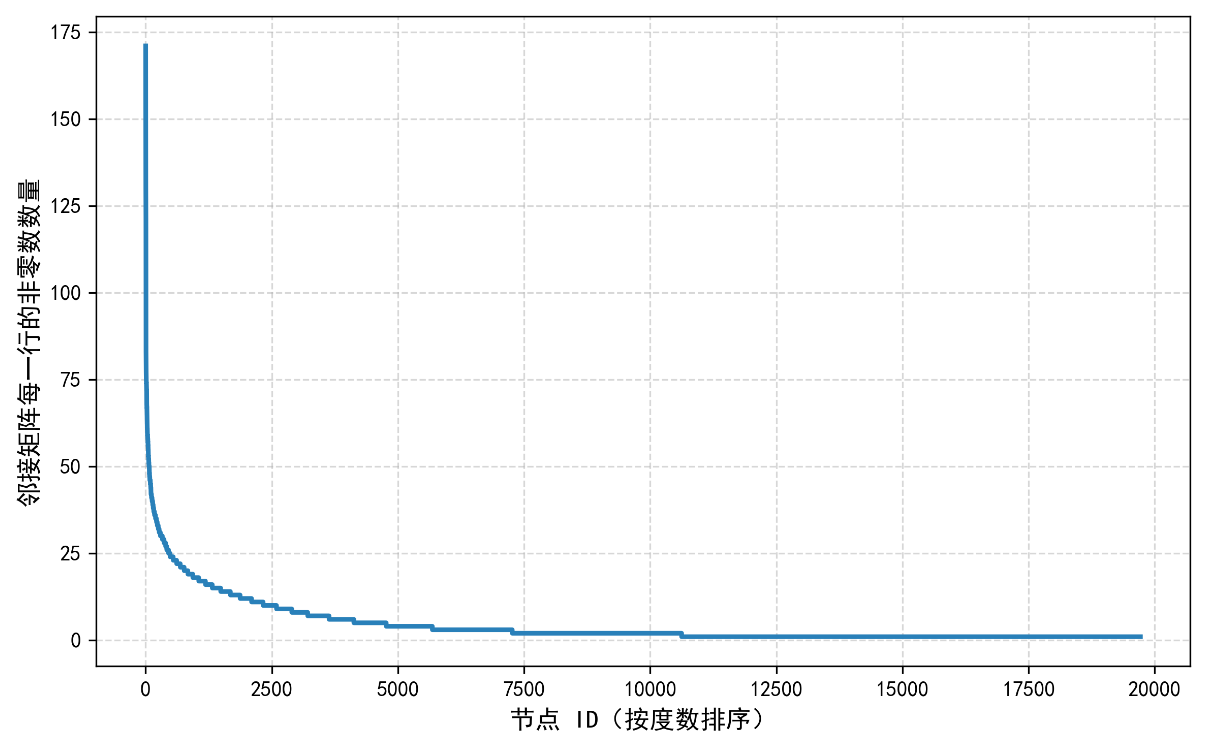


图 2 幂律定律图的非零元素分布特征

这种极端的稀疏性与不规则的非零元分布，导致了传统处理器在执行SpMM时存在大量的无效计算（零值乘加）和不规则内存访问。具体来说 : CPU受限于串行执行与分支预测开销，难以有效跳过无效计算；GPU虽具备并行能力，但其规整的并行模式对不规则数据访问支持不足，大量计算单元因数据依赖而空闲，缓存失效频繁，导致整体能效比低下。有关文献表明，SpMM占GNN推理总计算量的60%~80% [1] ，其中绝大部分是无效操作，这使其成为系统性能的瓶颈。

RISC-V 向量扩展提供标准化的可扩展编程模型，本项目基于其构建支持自定义 SpMM 指令的向量处理器，结合 CSR 压缩、固定行缓存与双指令发射机制，针对幂律分布图特性优化数据局部性与计算并行度。该设计通过专用硬件有效跳过了无效计算，不仅可显著加速 GNN 推理，还可推广至推荐系统、知识图谱、分子模拟等稀疏计算密集型应用，具有重要的学术研究价值与工程应用前景（支持边缘智能、低功耗图推理）。

### 1.2 项目目标

**功能目标：**

* 项目要实现的主要功能
  1. 实现基于RISC-V向量拓展的专用处理器,通过自定义指令集,重点优化稀疏矩阵-稠密矩阵乘法（SpMM）计算性能
  2. 设计并构建完整的三级指令执行流水线，实现对稀疏矩阵的高效加载计算和存储
  3. 设计针对幂律分布图的硬件加速架构，通过固定行缓存机制提升数据重用率和计算效率
  4. 支持指令并行，通过双发射机制隐藏内存访问延迟，提高硬件资源利用效率
  5. 设计总线和接口，专门优化稀疏数据的不规则内存访问模式
* 项目的技术指标要求
  1. 工作频率≥150MHz（基于Xilinx Artix-7 XC7A100T FPGA平台）
  2. SpMM计算吞吐率：≥2.0 GOPS（十亿次操作/秒）
  3. 能效比：≥1.5 GOPS/W
  4. 资源利用率：LUT < 75%，DSP < 60%，BRAM < 50个
  5. 分块矩阵的平均固定行缓存命中率≥80%
  6. 加速性能：相比于通用标量处理器，计算性能提高≥ 3倍

**学习目标：**

* 通过本项目要掌握的FPGA设计技能
  1. 复杂控制逻辑的设计：对于复杂状态机的设计与优化，实现多状态控制逻辑
  2. 高性能流水线设计：流水线架构设计与时序优化
  3. 对于专用硬件的优化技术：针对特定数据特征的硬件优化技术与存储层次结构设计
  4. 系统接口设计能力：总线协议设计与接口标准化规范
  5. 系统集成与模块间协同：模块的设计、仿真以及多个模块间的集成与协同优化
* 要提升的工程实践能力
  1. 系统级开发与验证能力：完整数字系统开发流程实践，深度参与从需求分析到板级验证全过程。
  2. 复杂系统调试能力：提高大规模数字系统的调试水平，包括对于模块级、系统级的仿真验证和逻辑分析仪ILA在线调试
  3. 设计优化与评估能力：学会进行设计空间探索，对不同设计方案的PPA进行对比以及评估，不断优化，采用能效最高的方案设计并验证。
  4. 软硬件协同设计能力：学习硬件软件协同设计与验证方法，尝试应用该方法提高处理器计算效率。
  5. 团队协作设计能力：团队成员负责各自模块的设计和调试，最终合作进行系统级调试。以及积累项目管理和项目汇报经验。

### 1.3 技术规格

* **目标器件：** Artix-7 **xc7a100tfgg484-2**
* **开发环境：** Vivado 2024.1
* **编程语言：** Verilog
* **验证平台：** 软件：Vivado Simulator , 硬件：Xilinx Artix xc7a100T
* **应用领域：** 人工智能推理，高性能计算，专用计算加速，数据挖掘与分析

## 2. 需求分析与系统设计

### 2.1 功能需求分析

**基于图神经网络推理中的SpMM计算的难点，本项目需要实现以下三大方面的功能需求：**

**计算架构需求：**

1. **指令集拓展：在标准RISC-V指令集的基础上扩展专用于SpMM指令，支持稀疏矩阵计算的特殊寻址模式以及一些定制化的操作比如存入稠密行，计算当前结果行的位图等。**
2. **流水线控制：设计高效且完整的状态机机制，能够处理在SpMM计算中取未被预存稠密行命中的行而造成的多周期等待问题。**
3. **并行计算：实现数据并行处理，在单个周期内处理多个矩阵元素的乘加运算（向量处理）。**
4. **指令级并行：支持指令双发射机制，实现计算与数据加载的并行执行；**
5. **数据预取：**在处理器核计算当前稀疏矩阵块时，DMA同时预取下一个待计算矩阵块的压缩数据，将数据准备时间重叠在计算时间内，提升整体任务吞吐量。**。**

**数据供给需求：**

1. **稀疏矩阵数据压缩：支持处理CSR稀疏矩阵压缩格式，减少存储空间和内存带宽占用。**
2. **动态地址生成：要求硬件能够实时解析稀疏矩阵每一行非零元素的位置，生成要读数据的地址。**
3. **跨步访问：VLSU能够根据地址跨步读稠密矩阵的行数据。**
4. **数据重用率优化：由于稀疏矩阵具有幂律图特性，稠密矩阵内部部分行会在计算的过程中不断被VLSU取出，为了提高数据重用率与局部性，在计算开始前提前预存数据到缓存当中。**

**存储层次需求：**

1. **多级缓冲：建立稠密矩阵缓冲区与向量寄存器文件的2级存储结构，平衡带宽和延迟。**
2. **并行访问：计算模块，向量寄存器文件和稠密矩阵缓冲区支持多端口并行写入和读出， 以向量为单位，满足数据供给需求以及提高处理效率。**
3. **高效数据传输：DMA控制器支持将CSR数据流水搬移至稀疏矩阵缓冲区。**

**核心功能模块：**

1. **功能模块1：**指令译码与状态控制模块（VID）

**对应需求：指令集扩展、流水线控制、动态地址生成、指令双发射，数据预取**

**功能描述：该模块是处理器的控制枢纽。VID解析8条自定义SpMM，通过三级状态机（WAIT，EXEC，HOLD）满足流水线控制需求，处理多周期等待；作为全局控制模块，VID向其他模块发送控制信号以及接收各个模块的完成信号进行状态的转换。实时解析行索引位图（row\_index）并生成跨步地址，满足动态地址生成需求。VID支持指令双发射调度，协调计算当前行与加载计算下一目标行所需要数据的并行执行。在计算当前块矩阵时VID使能稀疏矩阵数据传输模块，提前预取下一个矩阵计算所需要的稀疏数据，实现计算与传输的流水线化。**

1. **功能模块2：**数据传输模块（DMA，VLSU）

**对应需求：稀疏矩阵数据压缩、高效数据传输，跨步访问**

**功能描述：** DMA专门负责将CSR格式的稀疏矩阵数据从DRAM中高效，流式地搬运到稀疏矩阵缓冲区（Sparse buffer），其工作独立于处理器核，实现计算与传输的流水线化。向量加载单元（VLSU）负责在进行SpMM计算时搬运计算所需的稠密矩阵的行数据。它接收VID生成的地址,支持跨步访问模式,从片外读取稠密矩阵行到稠密矩阵缓冲区（Dense buffer）。在所有计算完成后，将最终结果从Dense buffer中写回内存。二者共同构成了处理器与DRAM之间的告诉数据通道。

1. **功能模块3：片上缓存和缓冲模块（VRF，Sparse buffer，Dense buffer）**

**对应需求：**数据重用率优化、多级缓冲，并行访问

**功能描述：这三个模块构成树立起的片上存储体系。Sparse buffer作为FIFO，暂存DMA来的稀疏数据流，想VEX提供稳定的CSR数据。Dense buffer是面向稠密数据的专用缓存，内部存储区包括存储每次计算要用数据的缓冲区，存储第一个计算结果矩阵的缓冲区，存储第二个计算结果矩阵的缓冲以及存储中间累加结果的缓冲区四个区域。VRF是靠近计算单元的缓存，包括固定行存储区和动态存储区，前者用来预存每次计算块矩阵的前个稠密行，后者用来存储未命中的行即从VLSU动态加载来的行数据。VRF直接为VEX提供操作数。**

1. **功能模块4：**计算模块（VEX）

**对应需求：数据并行处理（向量化）**

**功能描述：VEX是处理器计算中心。VEX包含多个并行的处理单元（Lanes）,采用SIMD架构。在SpMM计算模式下，他一方面接收从Sparse buffer来的CSR非零元数据，另一方面从Dense buffer读取对应的稠密行数据，在多个Lane当中执行向量的乘加（MAC）操作。在对中间矩阵合并时，VEX通过VID控制读取Dense buffer两个中间结果的对应行进行向量加法。此外，VEX支持激活函数，从而引入非线性特征。通过向量化处理，实现数据级并行。**

**性能需求：**

* 工作频率：**≥ 150 MHz**
* 数据吞吐率：**≥ 2.0 GOPS**
* 延迟要求：
  + 指令级延迟：从外部加载稠密矩阵行延迟不能超过%，也就是VRF稠密行的命中率要高于%。
  + 任务级延迟：对于不同数据集相比基线RISC-V标量处理器实现≥3倍的性能提升。
* 精度要求：INT8

### 2.2 系统架构设计

#### 2.2.1 顶层架构

图3展示了本设计的系统架构图。本设计实现了一个基于RISC-V向量扩展的专用处理器架构，专门针对稀疏矩阵乘法（SpMM）进行优化。该架构主要由一个基于向量处理器的计算组件、多组片上存储缓冲区以实现数据局部性最大化、两个协调片内外数据搬移单元，以及主控制单元组成。

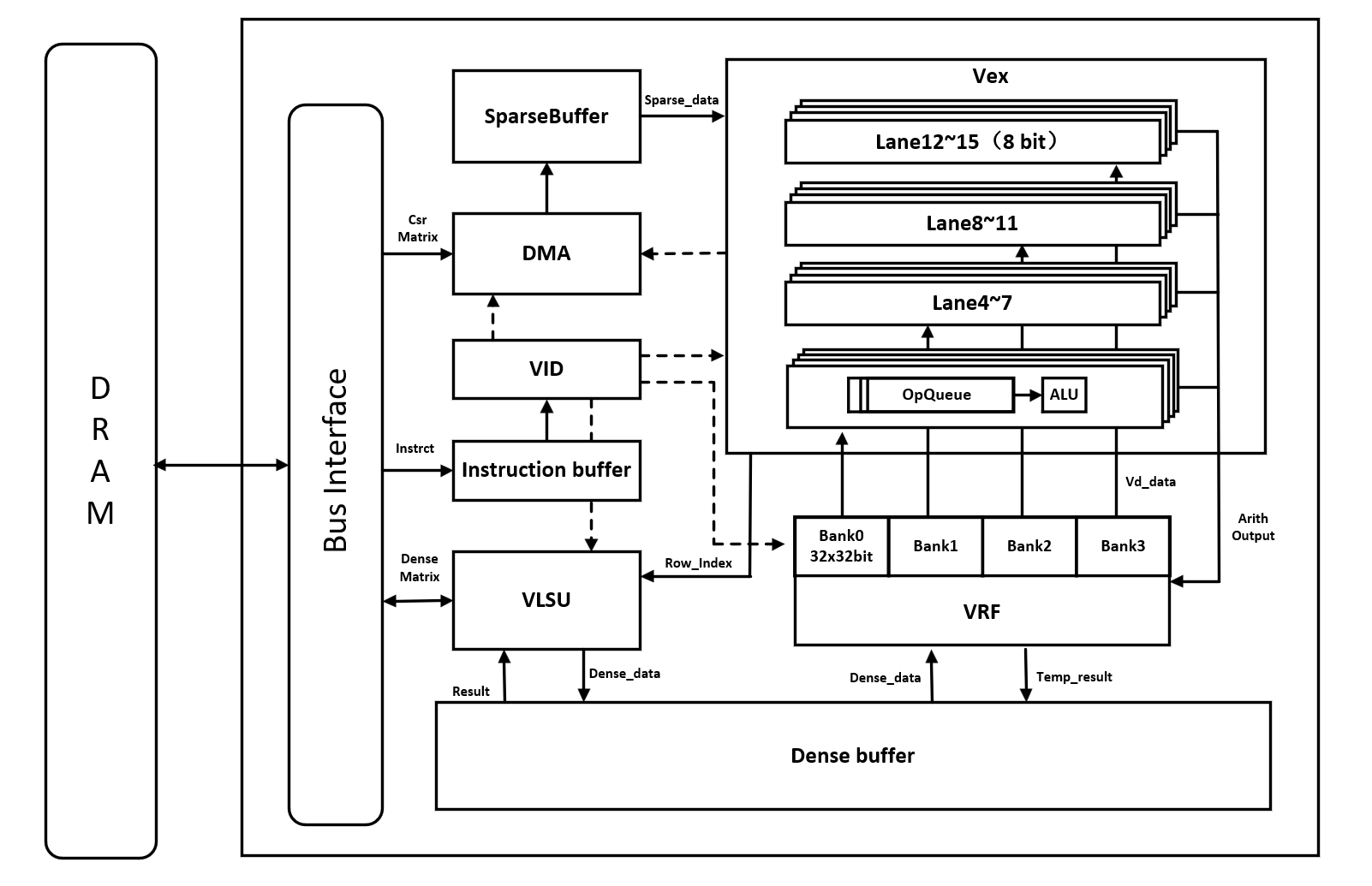


图 3 系统架构图

图4展示了详细个各个子系统对应的模块，以及各模块的功能概述。下面详细介绍各个子系统对应的功能。

控制子系统由VID模块作为主控制单元，负责解析自定义的SpMM指令集并协调整个计算流程。Ctrl模块控制处理器的流水线暂停，支持跨模块暂停请求。Pc\_reg和if\_id模块协同工作,确保指令的连续获取和译码,为向量的计算以及各个模块的运行提供稳定的控制信号。

存储子系统采用层次化设计。包含四组SRAM，VRF，Dense buffer，Sparse buffer，Instruction buffer。VRF最靠近计算单元，存储SpMM每次所需要的稠密矩阵的行。Dense buffer作为计算单元的二级缓存，靠近DRAM，存储每次计算需要用的数据并提供给VRF以及计算单元的计算分块矩阵完成的中间结果。Sparse buffer负责存储CSR格式的稀疏矩阵数据。Instruction buffer存储并给VID提供指令和操作数支持处理器持续计算的流水线。

接口子系统包含VLSU和DMA两个模块。VLSU支持向量化的跨步内存访问，用于处理SpMM的稠密矩阵不规则访问。DMA单元负责稀疏矩阵的流式传输，可同时实现计算与数据传输的流水线重叠,提高系统吞吐量。

计算子系统核心是VEX向量执行单元,该单元包含16个并行处理通道Lane,每四个跟VRF的一个bank相对应,实现数据向量化传输以及实现SIMD向量乘加运算。该计算子系统能高效执行SpMM计算以及对于中间结果的加法运算。

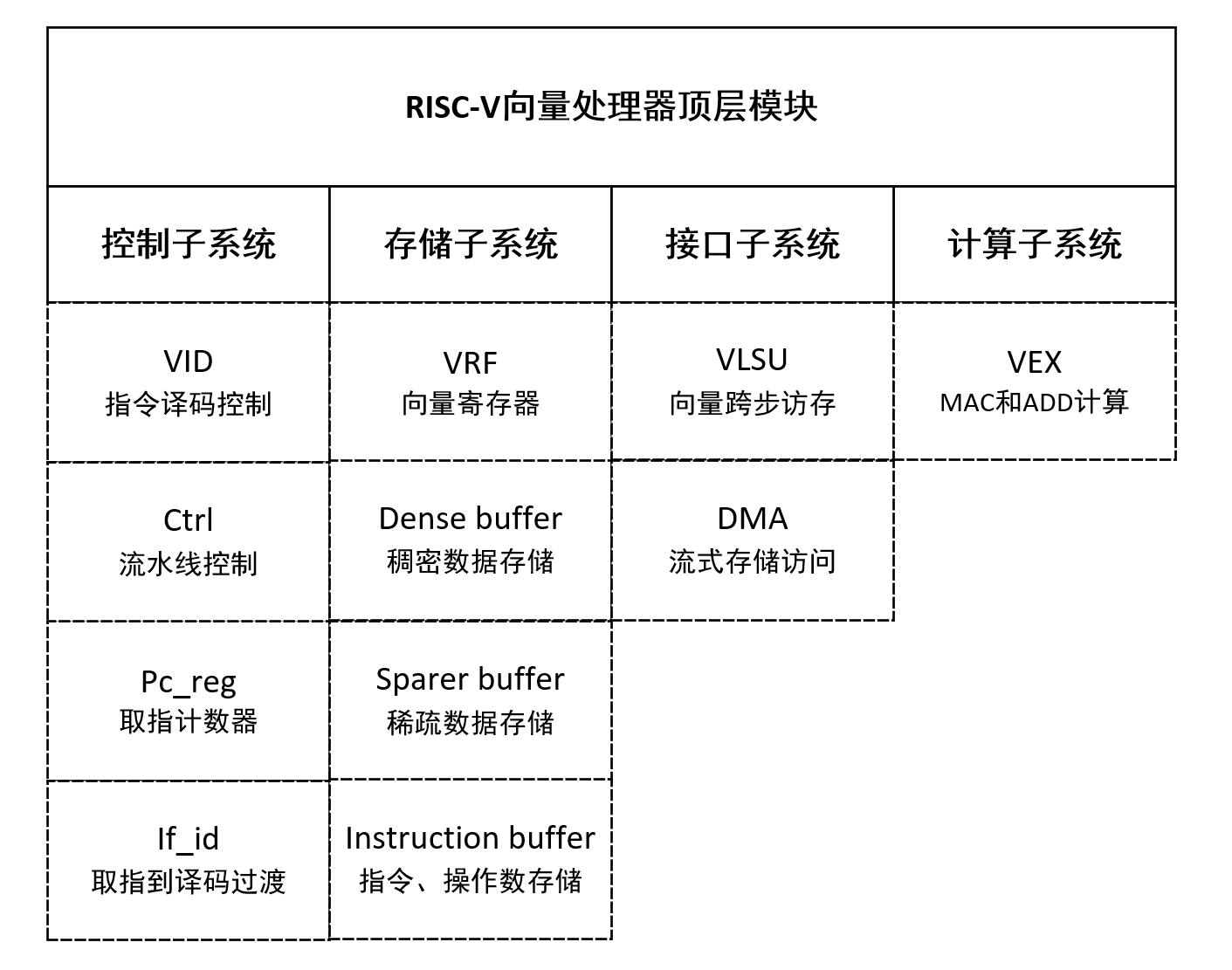


图 4 子系统及对应模块说明

#### 2.2.2 状态机设计

本设计的核心控制逻辑采用三态有限状态机，实现精细的指令执行控制和资源调度。状态机内嵌于VID模块当中，负责协调整个向量处理器的指令执行流程。图5展示了本设计核心控制逻辑的状态机转换示意图（完整细节见代码部分）。

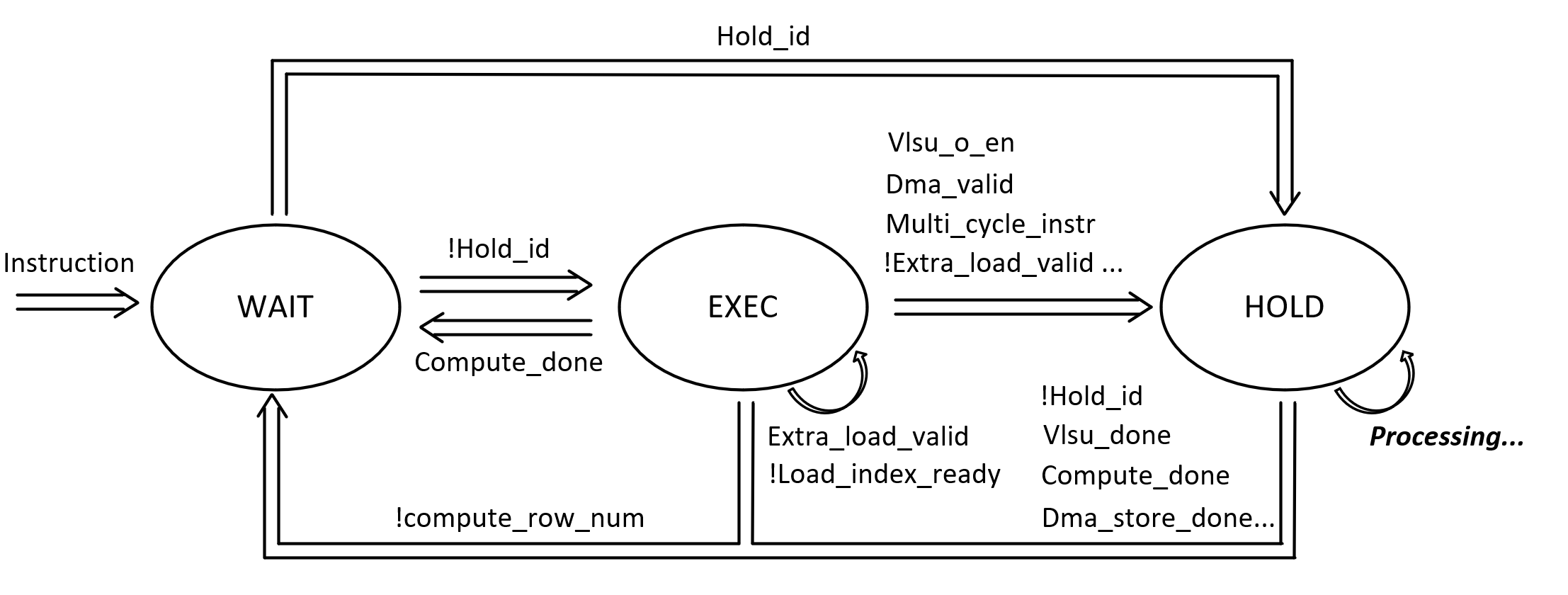


图 5 本设计状态机转换示意图

本设计扩展了8条自定义SpMM指令，指令格式如图5所示。状态机根据不同的指令向特定功能模块发出特定的控制信号。

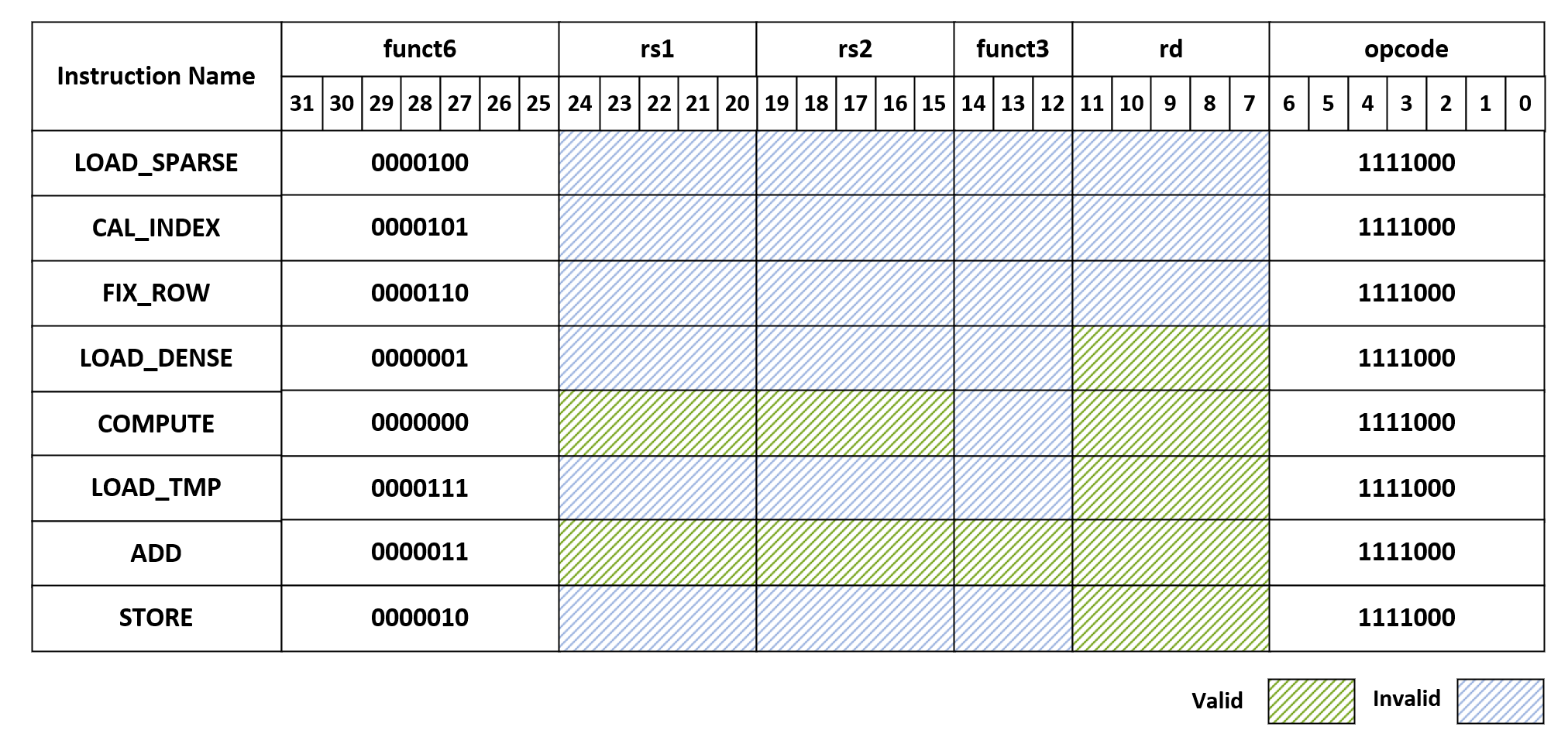


图 6 自定义指令集

**状态功能概述以及转移逻辑**

* WAIT (IDLE)
  + 功能描述：系统初始化和指令等待的状态，负责接受和准备执行指令。该状态下所有控制信号被设置成默认值，包括禁用VEX，VLSU，VRF读写等，确保系统处于安全并空闲的状态。
  + 状态转换条件：
    - 当hold\_id有效时，转入HOLD处理外部暂停请求
    - 当hold\_id无效时，转入EXEC开始指令执行
    - 无外部干预条件下，系统自动转入EXEC开始指令执行。
* EXEC
  + 功能描述：指令译码和初始执行阶段，根据不同的SpMM指令生成相应的控制信号并启动执行单元（VEX，VLSU等）。此状态完成指令的解析和初始控制信号生成与发送。
  + 状态转换条件
    - 维持EXEC（满足一下任一即可）：
      * COMPUTE指令，计算当前行时VRF全部命中所有所需稠密行（extra\_load\_valid拉高，VEX直接开始计算）。
      * LOAD\_DENSE指令，比对索引和准备稠密数据地址未完成（vlsu\_en\_o未使能）。
    - 转入WAIT：COMPUTE指令，计算完成（spmm\_compute\_done拉高）。
    - 转入HOLD（满足一下任一即可）
      * LOAD\_DENSE指令，vlsu\_en\_o使能信号拉高。
      * 多周期并且需使能VEX的指令（除COMPUTE指令外）。
      * LOAD\_SPARSE指令，DMA传输指令有效（dma\_instr\_valid\_o拉高）。
      * LOAD\_TMP指令，向量加载操作（dense\_buf\_read\_en\_o拉高）。
      * COMPUTE指令下，计算当前行时VRF未全部命中所有所需稠密行的条件下（需要在HOLD状态下load到vrf当中）。
* HOLD：
  + 功能描述：多周期操作执行和等待阶段，维持执行单元的工作状态，监控操作进度，通过接受执行单元完成信号处理数据依赖关系。
  + 状态转换条件：
    - 维持HOLD：多周期操作正在执行且未收到完成信号。
    - 返回WAIT：所有多周期操作完全执行完毕，无后续任务。（具体信号详见代码部分）

**状态行为详细描述**

以下各个状态内的行为和控制信号生成的详细描述。

* WAIT
  + 信号控制：执行单元使能信号置为无效（vex\_en = 0, vlsu\_en\_o = 0等）,等待外部hold信号拉低。
* EXEC
  + 指令级别控制：
    - LOAD\_SPARSE（SPMM\_LOAD）指令：配置DMA传输参数，数据地址为指令的操作数。若dma未存储完成（dma\_store\_done为低），设置dma\_instr\_valid\_o信号有效，启动稀疏数据流式传输，根据稀疏数据大小设置传输次数。设置multi\_cycle\_instr标志，指示该指令需要多周期执行，并在下一个周期进入HOLD状态。
    - CAL\_INDEX指令：使能VEX计算单元。配置pe\_op操作模式为PE\_SPMM\_CAL\_INDEX，启动行索引row\_index和密集行索引row\_index\_dense的计算。设置Sparse\_load信号有效，准备稀疏数据。设置multi\_cycle\_instr标志，指示该指令需要多周期执行，并在下一个周期进入HOLD状态。
    - FIX\_ROW指令：使能VLSU加载单元（vlsu\_en\_o = 1,vlsu\_load\_o = 1），加载到VRF稠密行数据。配置pe\_op操作模式为PE\_SPMM\_FIX\_ROW，配置操作数选择为CSR寄存器源为PE\_OPERAND\_CSR。设置multi\_cycle\_instr标志，指示该指令需要多周期执行，并在下一个周期进入HOLD状态。
    - LOAD\_DENSE指令（load）：
      * **启动阶段（索引计算）：**设置load\_index\_o = 1 向VEX请求计算当前行所需稠密矩阵的行索引。配置加载到向量寄存器的目标地址（dense\_buf\_write\_addr\_o），为数据存储做准备。设置multi\_cycle\_instr = 0（不依赖该信号进入HOLD而是vlsu\_en\_o）。
      * **加载条件验证并使能：**当取数据地址准备好（scalar\_operand\_ready拉高）并且VEX已经计算好行索引（load\_index\_ready\_r）时使能VLSU加载单元，否则不使能。设置操作模式pe\_op为PE\_SPMM\_LOAD。
      * **控制DMA预取稀疏数据：**根据dma\_staore\_done判断，DMA存储完成时禁用，未完成时（dma\_store\_done为低）则持续拉高dma\_store\_en进行稀疏数据传输。
    - COMPUTE指令：
      * **状态转移**：设置multi\_cycle\_instr标志，指示该指令需要多周期执行，下一个周期进入HOLD状态。
      * **计算单元配置**：使能VEX。配置pe\_op操作模式为PE\_SPMM\_COMPUTE。设置操作数为CSR寄存器PE\_OPERAND\_CSR，确保是从稀疏矩阵获取数据。
      * **数据检测：**当外部数据准备好时即extra\_load\_num\_ready拉高，使能稠密缓冲区读取，配置读取地址并使能VRF写操作。在数据准备好时（计算当前行的数据都放在VRF当中），即compute\_load\_extra和compute\_load\_stationary任一信号拉高，使能VRF加载，数据得以进入到计算通道（Lanes）当中，准备开始计算。
      * **控制DMA预取稀疏数据：**根据dma\_staore\_done判断，DMA存储完成时禁用，未完成时（dma\_store\_done）为低则持续拉高dma\_store\_en。
    - LOAD\_TMP指令（load\_vector）：使能Dense buffer加载功能和VRF写入。配置操作模式pe\_op为PE\_SPMM\_LOAD\_TMP（load\_vec）。设置multi\_cycle\_instr标志，指示该指令需要多周期执行，并在下一个周期进入HOLD状态。
    - ADD指令：使能VEX，配置操作模式pe\_op为PE\_ARITH\_ADD。设置multi\_cycle\_instr标志，指示该指令需要多周期执行，并在下一个周期进入HOLD状态。
    - STORE指令：使能VLSU存储单元（vlsu\_store\_o），禁用DMA传输防止总线冲突，设置multi\_cycle\_instr标志，指示该指令需要多周期执行，并在下一个周期进入HOLD状态。
* HOLD：
  + **指令级别控制：**
    - LOAD\_SPARSE指令：若加载未结束则持续使能DMA。
    - CAL\_INDEX指令：维持VEX使能和行索引计算模式，等待索引计算完成。
    - FIX\_ROW指令：维持VLSU加载单元使能状态，根据vlsu\_done\_i完成信号和number\_rows计数器，控制多行加载的连续执行。当一行加载完成单仍有剩余行时，维持加载使能，否则即完成指令。
    - LOAD\_DENSE指令：维持LOAD\_DENSE模式。在未完成所有行加载时，当VLSU加载一行稠密数据结束则立刻使能VLSU加载单元加载下一行以及维持Dense buffer写使能。否则不使能。根据DMA完成状态，若加载未结束则继续使能DMA预取稀疏数据，反之则停止使能。
    - COMPUTE指令：
      * **计算过程监测：**维持VEX使能信号和保持COMPUTE的计算模式。检测计算完成信号（spmm\_compute\_done）。
      * **动态数据加载：**根据extra\_load\_num\_ready信号控制稠密缓冲区读取，若仍有数据未计算且已加载到Dense buffer（该信号拉高），则使能Dense buffer加载到VRF当中。当compute\_load\_extra或者compute\_load\_stationary有效时，维持VRF加载使能，使数据能够进入VEX进行计算。
      * **计算结果写回：**在检测到spmm\_compute\_done信号是，使能VRF和Dense buffer写入，配置写入地址，设置数据来源为算数运算结果（DENSE\_WB\_SRC\_ARITH）。
      * **维持稀疏数据预取：**根据DMA状态，若加载未结束则继续使能DMA，反之则停止使能。
    - LOAD\_TMP指令：维持该操作模式。根据计数器，若load次数小于3则继续使能Dense buffer加载和VRF写入，若已完成三次加载则不使能（三次是指加载两个子矩阵矩阵行和一个中间结果矩阵行共三次）。
    - ADD指令：维持加法计算模式，当VEX完成加法运算并输出时，根据funct3来判断写入单元。若funct3 = ’0则使能Dense buffer写入。若funct3 = 1则使能VRF写入。
    - STORE指令：维持VLSU存储使能以及禁止DMA传输。

2.2.3 时序设计

图7展示了本设计的指令执行时序图，一共分为四个阶段，清晰的展示出了指令的执行顺序和处理器开始执行对应操作的时机，为了方便展示，示意图里是进行4x4矩阵的SpMM计算的过程。本设计以两组16x16的分块矩阵为单位进行计算和整合。

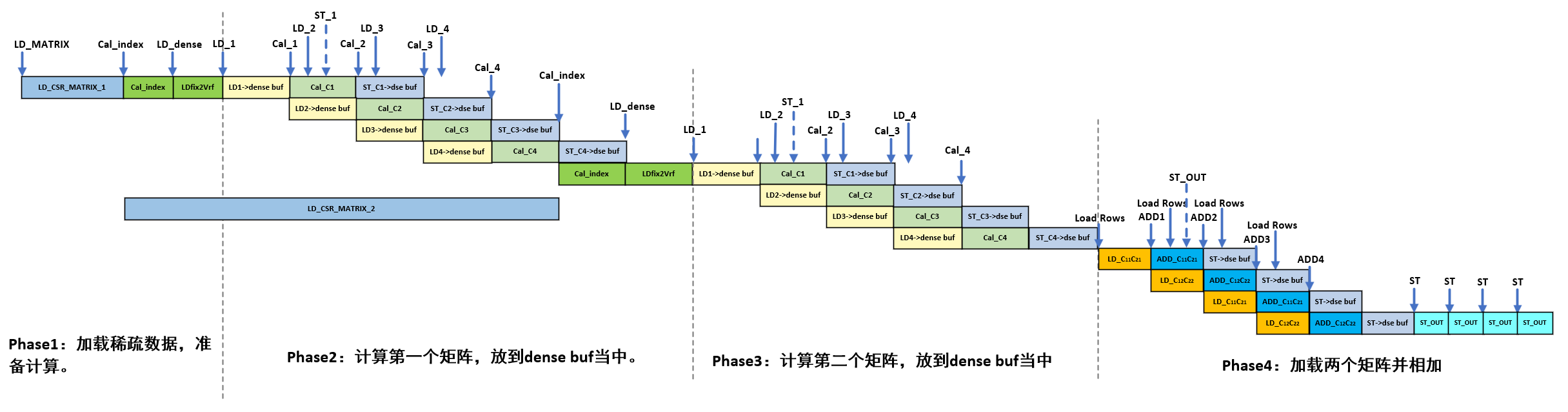


图 7 指令执行时序图

**Phase1**

* 整体概括

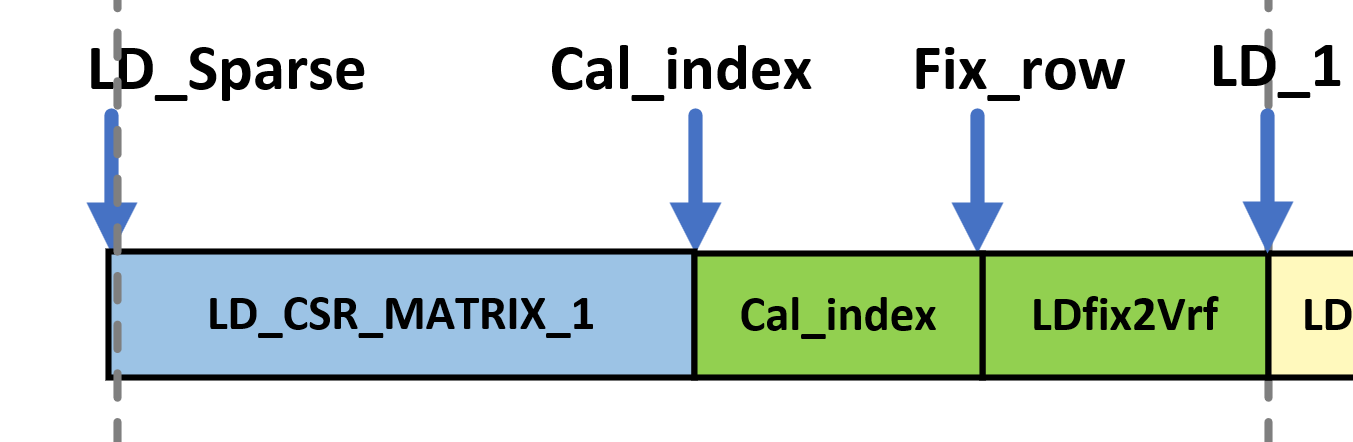


图 8 Phase1指令执行时序图

该阶段完成了稀疏矩阵加载、行索引计算以及稠密行存储的任务，为后续高效计算做好准备。首先DMA加载稀疏数据，随后VEX计算每次计算要用到的行索引row\_index，采用独热码的方式记录。该阶段的最后将该分块稀疏矩阵参与SpMM计算要用到前四常用的稠密矩阵的行数据加载到VRF当中。此外，在进行完第一个稀疏矩阵加载，VEX并全部接收到后，DMA会被立刻使能加载第二组矩阵中稀疏矩阵的数据。

* 仿真细节

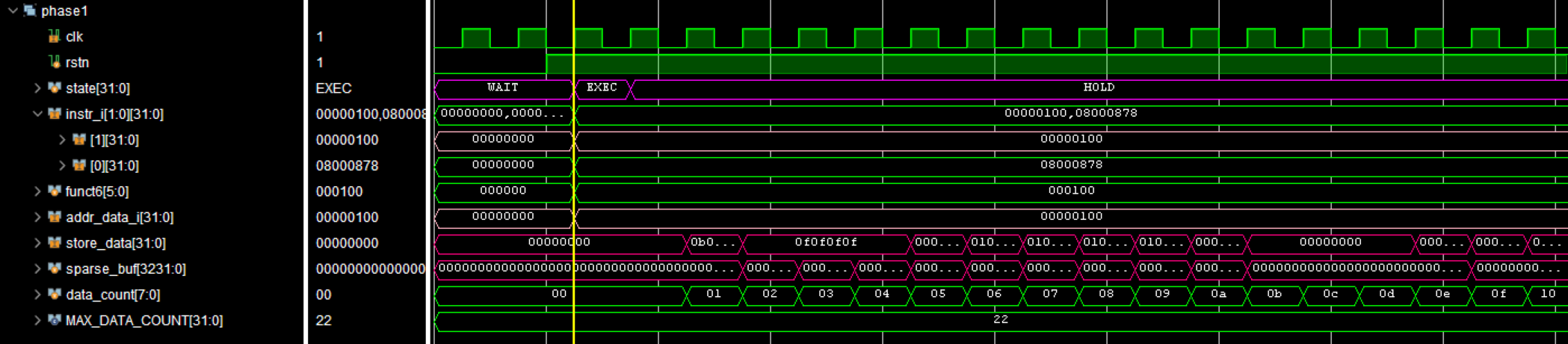


图 9 处理器开始工作并使能DMA

由图8可知，VID模块在EXEC接收到LOAD\_SPARSE指令（funct6 = 0000100）接着译码并使能DMA开始加载数据。进入的指令为一个二维数组，第一个为指令操作数，表示待取外部稀疏数据的首地址，DMA根据内部的计数器在首地址的基础上加上n倍的步长得到数据地址。Store\_data为DMA取出来的数据并存入Sparse buffer，每次取32bit数据，我们设定稀疏矩阵非零元素最大占比为31.25%（80/256个），一共要取22次数据，对应MAX\_DATA\_COUNT为22。

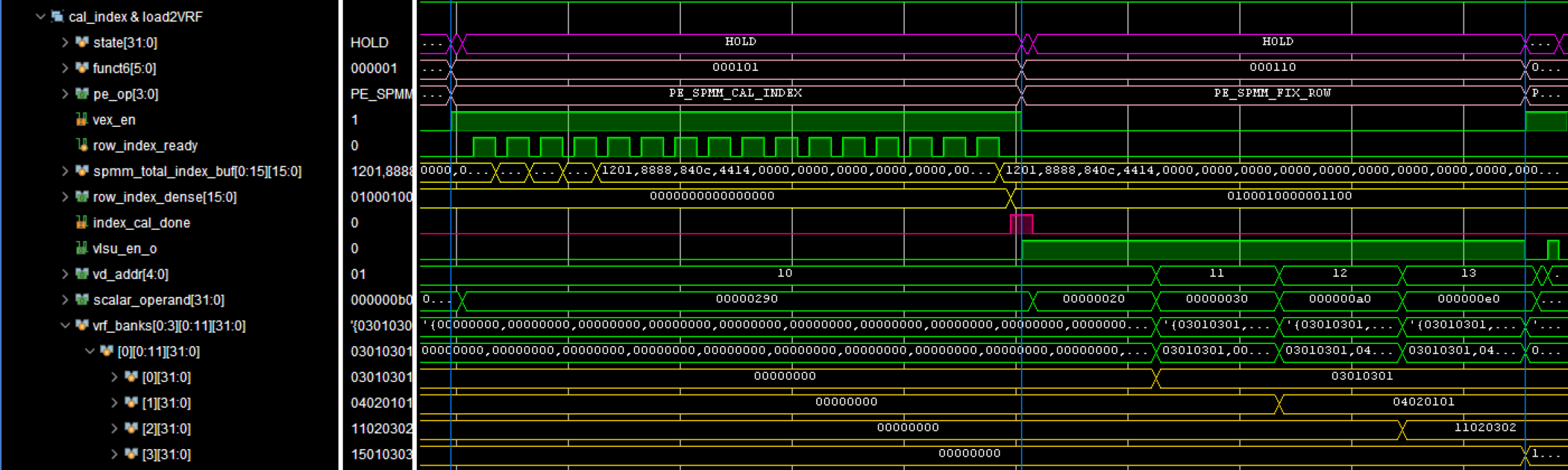


图 10 计算行索引以及存入VRF

在这之后，接收CAL\_INDEX和FIX\_ROW指令，由图9所示。首先处理器执行CAL\_INDEX指令，由于矩阵大小是16x16，我们在VEX当中设置16维数组用来存放计算好的row\_index，存放16行数据的索引。VEX先对csr格式的数据进行解析，根据csr数据的行指针和列索引进行计算。row\_index长度为16，采用独热码的形式，例如，稀疏矩阵第一行的第1,3,5,14个为非零元素，此时对应的row\_index为1010100000000100。此外，依旧在CAL\_INDEX指令下，我们通过解析整个矩阵的数据非零元素在每个列的出现频率，计算后续计算前四个会常用行的row\_index\_dense,长度为16，输入给VID，该指令结束（两个索引在图中黄色高亮）。

图10展示了row\_index的计算，为了方便展示，使用更小的矩阵和更少的常用行计算，虚线代表矩阵当中不存在该数据。

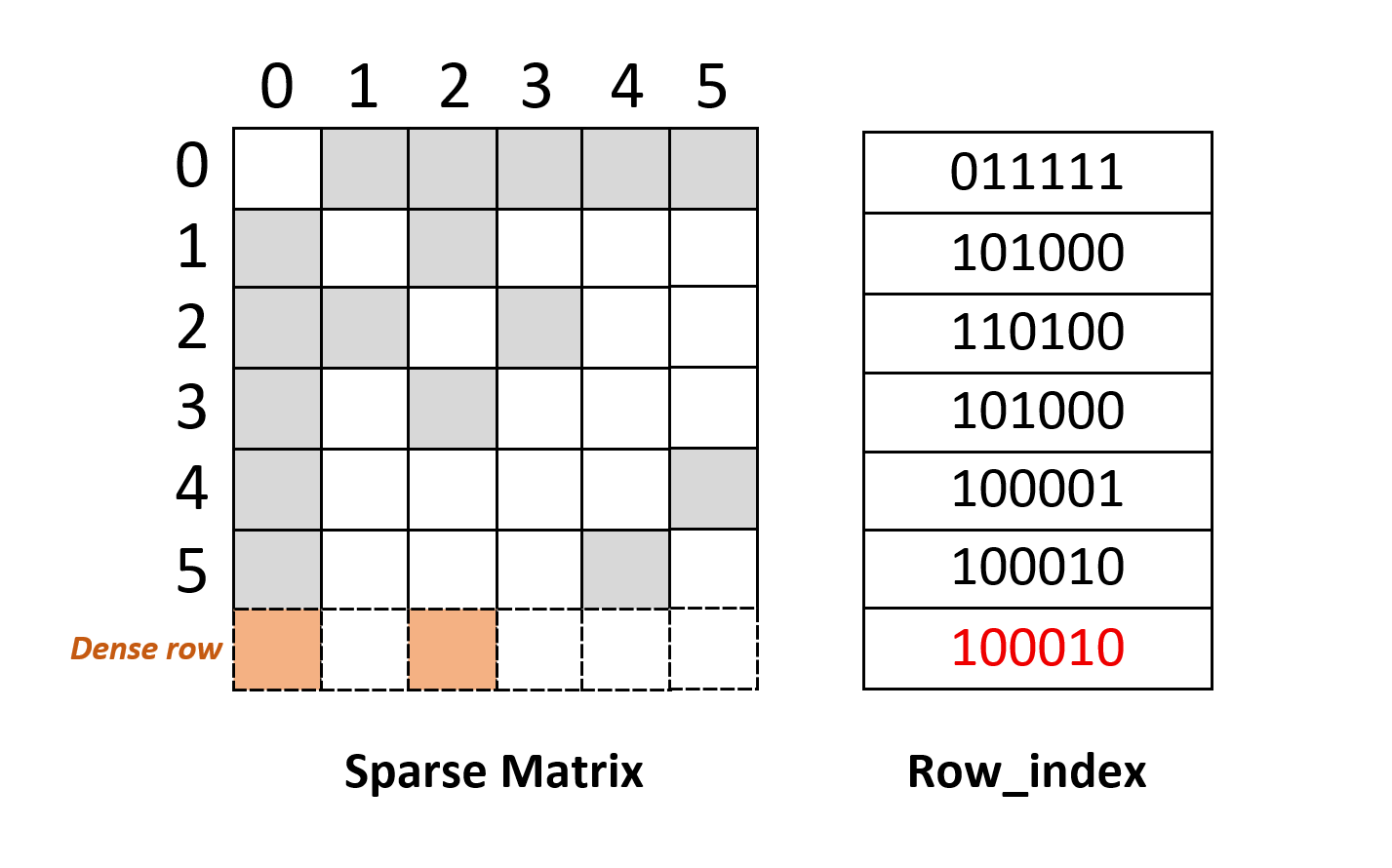


图 11 计算row\_index示意图

随后执行FIX\_ROW指令，VID根据row\_index\_dense生成地址，使能VLSU向DRAM取数据存到VRF第1-4个向量寄存器当中。

**Phase2**

* 整体概括

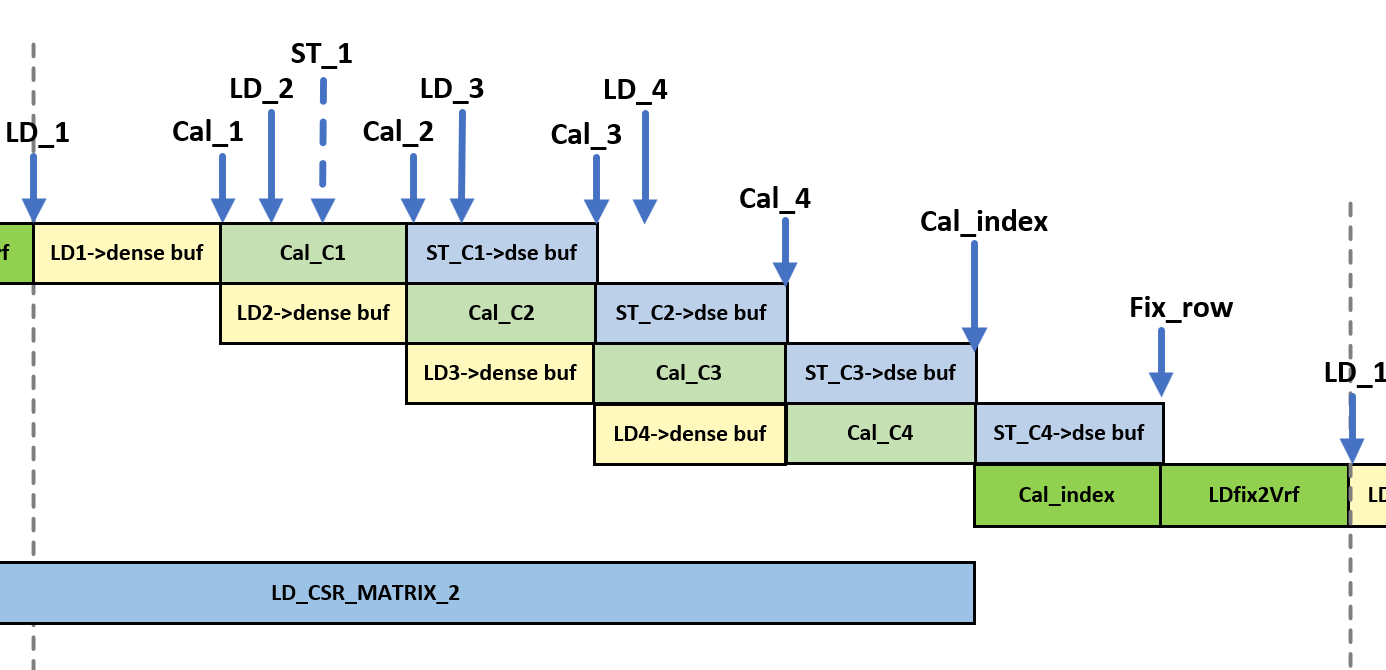


图 12 Phase2指令执行时序图

该阶段完成了第一组矩阵的SpMM计算。由于采用矩阵行积，以行为单位进行计算。在对每一行进行计算之前，先根据Phase1阶段的产生的该行的row\_index，生成地址，使能vlsu向DRAM取稠密数据，存到Dense buffer当中。COMPUTE指令执行时先将数据加载到VRF再加载到计算单元Lane当中进行乘加计算，随后存到Dense buffer中预留的存储第一组的结果的区域。该阶段采用指令双发射策略，在执行当前行计算的COMPUTE指令时同时接收LOAD\_DENSE指令提前加载下一行所需要用到的稠密行数据到Dense buffer。

在进行SpMM计算过程中，DMA在总线 空闲的情况下（COMPUTE指令时），同时取下一组稀疏矩阵的数据，计算结束后数据已经存入Sparse buffer当中。随后处理器执行计算下一个矩阵的row\_index以及将新矩阵的前四常用行存入到VRF当中。

* 仿真细节

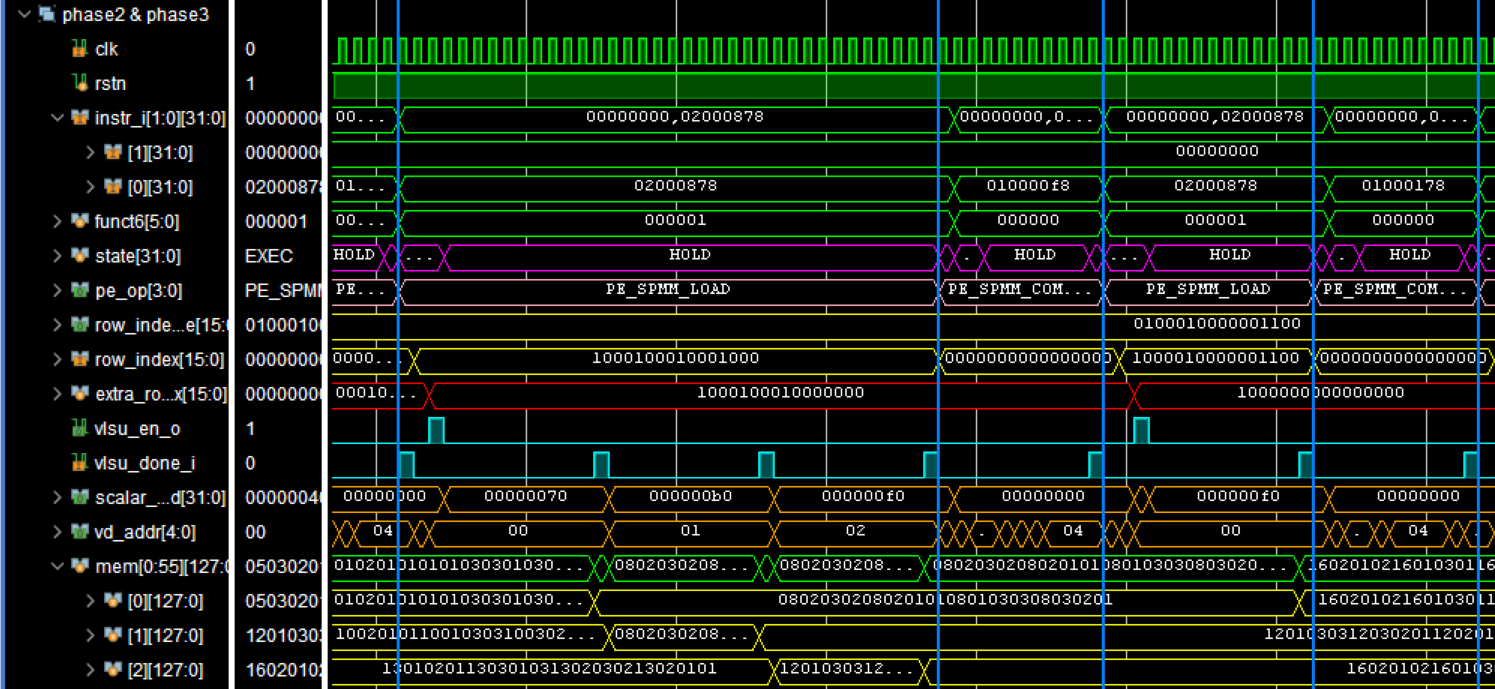


图 13

如图13，处理器进入Phase2首先执行LOAD\_DENSE指令，在该指令下，首先VID接收从VEX来的计算第一个结果行的row\_index。VID将该行索引和VRF内部已经存下的row\_index\_dense相比较，用于判断哪些行需要从外部DRAM当中取，得到extra\_row\_index（图中红色高亮显示）。以图13的第一次load指令举例，对比完之后extra\_row\_index为1000100010000000显示第8,12,16行需要从DRAM当中取（从右向左数），第四行数据由于已经存到VRF当中所以在extra\_row\_index当中没有在对应位置显示为1。随后根据该索引产生取数据的地址，对应图中的scalar\_operand，使能vlsu，vlsu取三次数据（vlsu\_done\_i拉高三次代表从DRAM完整加载三次数据，一次128bit）存到Dense buffer当中的第0,1,2行的位置。

随后执行COMPUTE指令，如图14，VEX根据上一步LOAD\_DENSE存入Dense buffer的数量和数据，将所有要计算的数据加载到VRF的第4,5,6个寄存器当中，随后将需要的全部数据加载到VEX当中进行乘加计算，最后根据指令的rd加上偏移地址得到输出地址，将计算结果arithouput输出到Densebuffer的对应位置当中

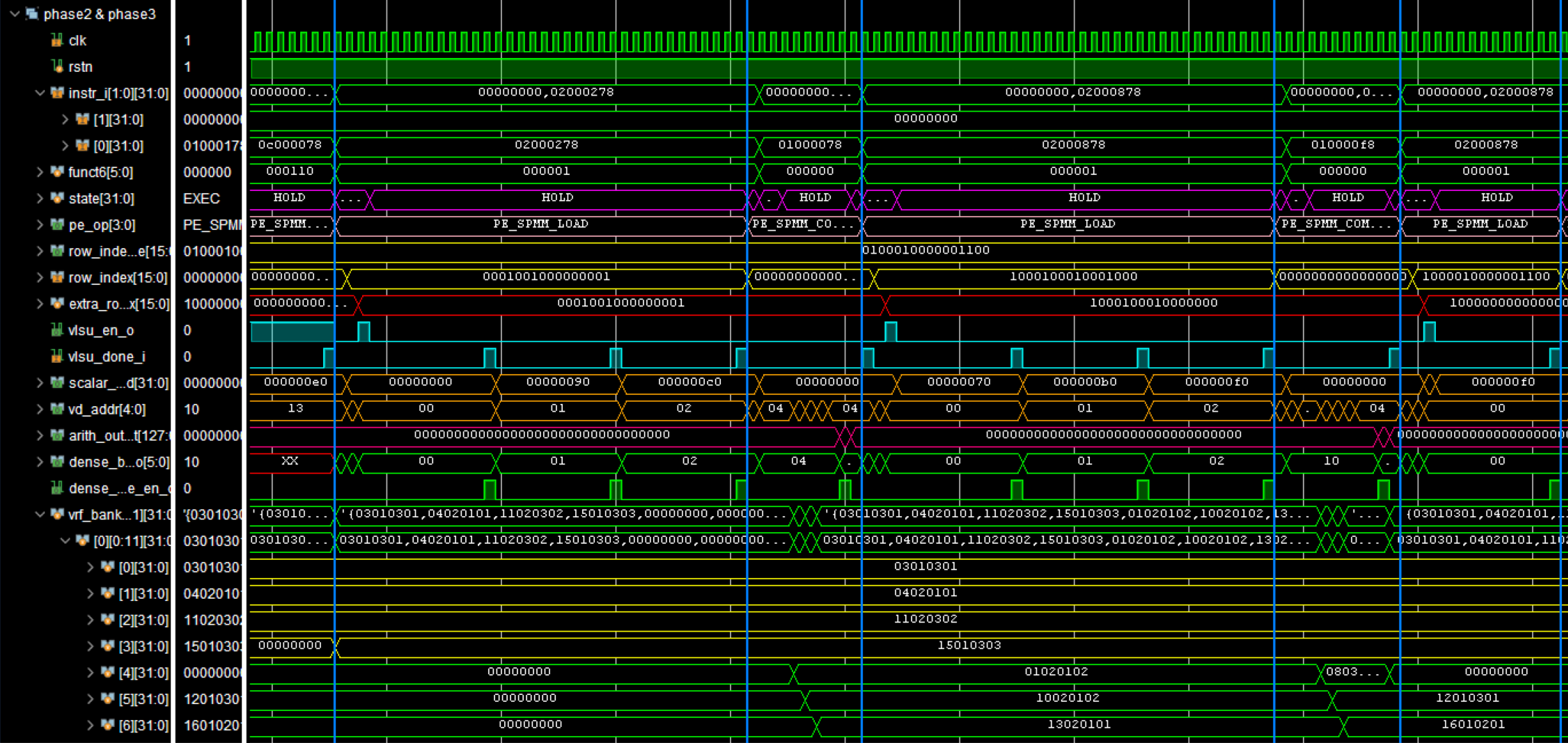


图 14

对于双发射，描述上两个同时执行，图

**Phase3**

* 整体概括

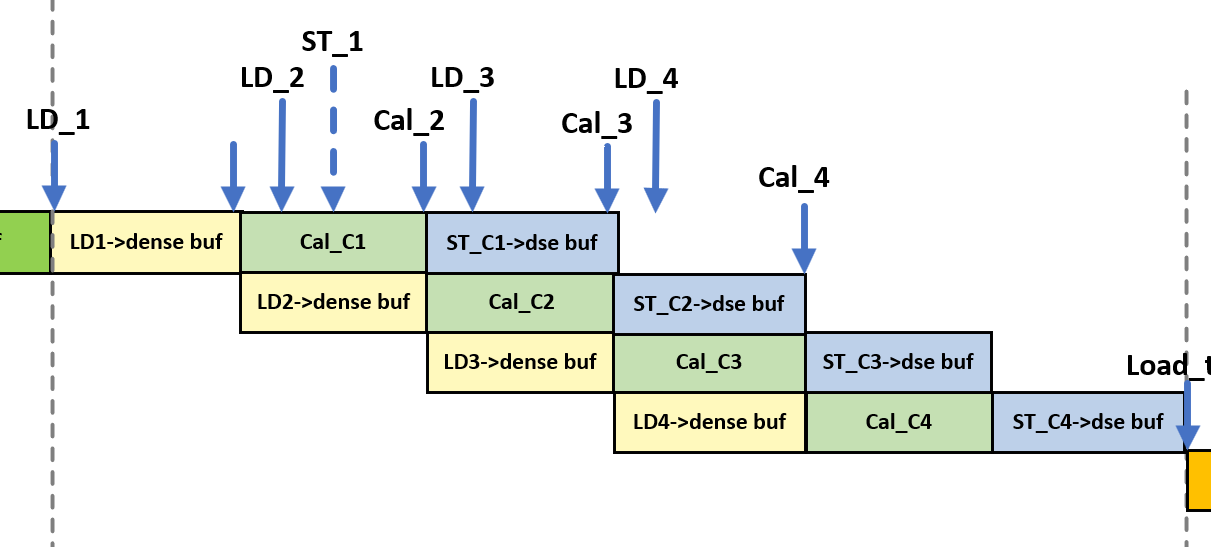


图 15 Phase3指令执行时序图

该阶段指令执行过程跟Phase2基本相同。由于是两组矩阵计算为单位，该阶段后续不涉及到执行CAL\_INDEX和FIX\_ROW指令，也不会使能DMA取稀疏数据。计算完成后的数据存到Dense buffer中预留的存储第二组的结果的区域。

**Phase4**

* 整体概括

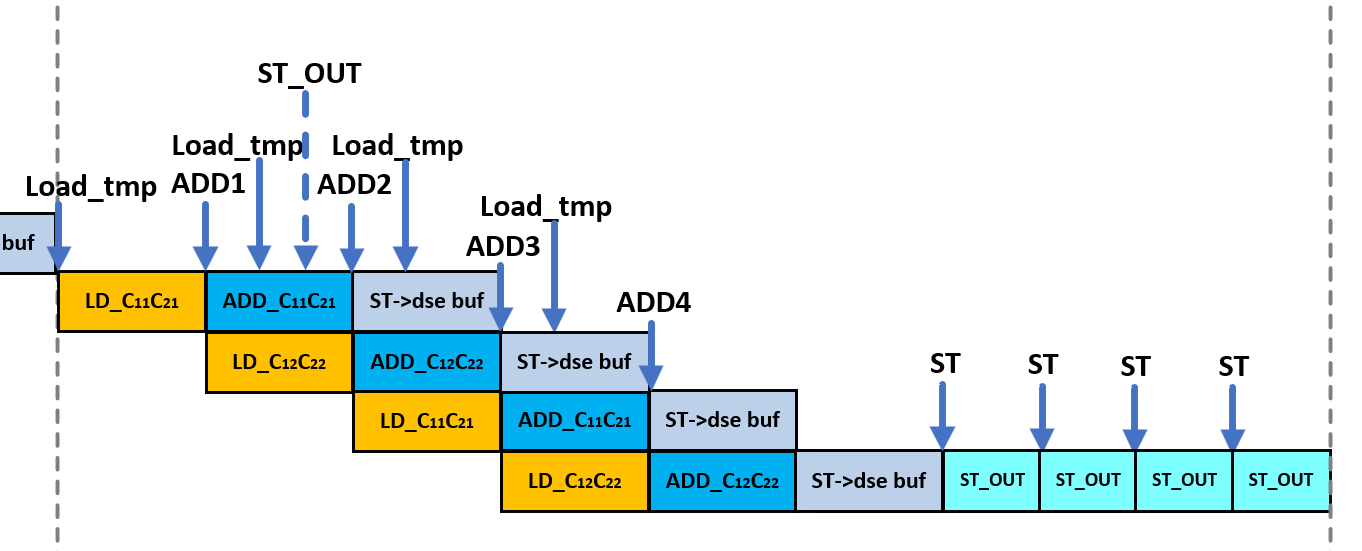


图 16 Phase4指令执行时序图

Phase4是进行结果整合阶段。由于产生了两组分块矩阵的计算结果，需要将两个矩阵从Dense buffer加载并向量化的对行一一相加。执行LOAD\_TMP指令后，根据偏移地址rd，用三个周期将Dense buffer三块矩阵的共三行结果加载到VRF当中（三个矩阵为：第一组矩阵，第二组矩阵，累计中间结果矩阵）。由于三个向量需要进行两次相加，故使用两条ADD指令，根据funct3，第一条ADD指令执行第一次，存入VRF中。第二条ADD指令执行第二次，根据rd存入Dense buffer当中。该阶段采用指令双发射策略，在执行当前行计算的ADD指令时同时接收LOAD\_TMP指令提前加载下一行所需要用到的矩阵数据。

如果对于结果矩阵的16x16块已经全部计算累加完毕，并存在了Dense buffer的累计中间结果矩阵当中，此时执行STORE指令，将结果通过总线写入到DRAM当中。

* 仿真细节

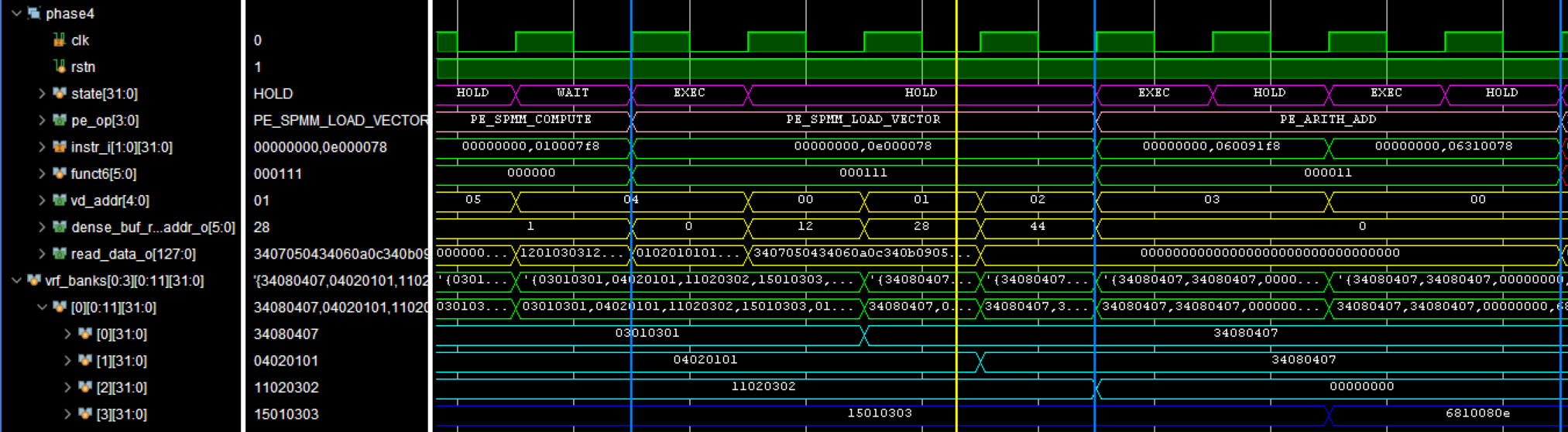


图 17

如图12，处理器在该阶段先执行LOAD\_TMP指令，用三个周期读Dense buffer当中的三行数据存入到VRF。由于三个矩阵分别存在Dense buffer的第12~27,28~43,44~59行（0~11行放LOAD\_DENSE加载的数据）。rd作为矩阵内部的偏移。图中指令的rd为0，在HOLD期间进行加载数据，所以在HOLD第一个周期读地址为12，第二个周期为28，第三个周期为44。由于SpMM的计算已经完毕， VRF中的常用行已不再使用，所以我们指定这三个向量存到VRF的前三个向量寄存器，对应图中的vd\_addr为0,1,2。

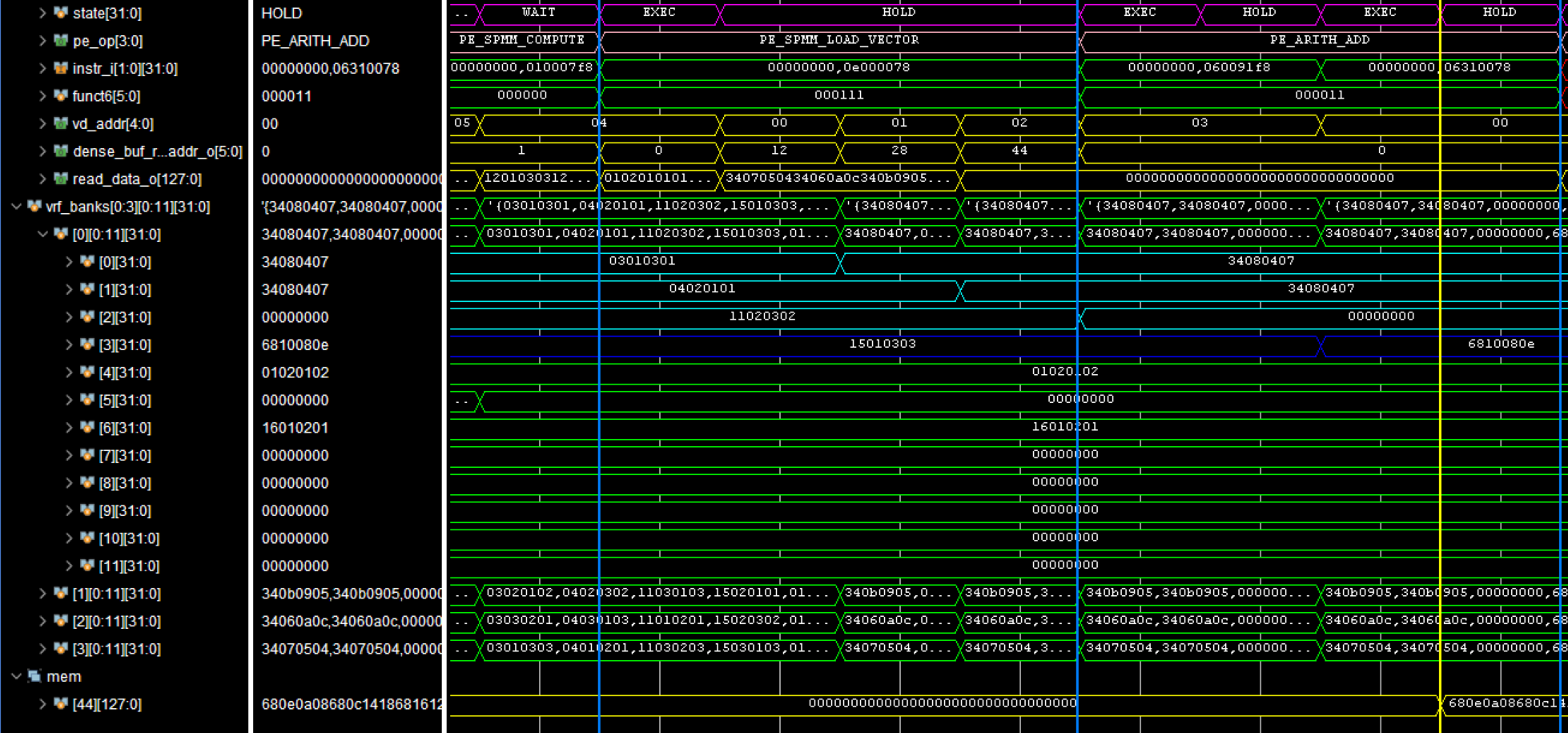


图 18

随后执行两条ADD指令，第一次将VRF的0,1寄存器的值相加放到第3个寄存器，随后将VRF的2,3寄存器的值相加。根据rd即矩阵内部的偏移，加上第三个矩阵第一行的偏移地址（为44）得到写入地址，随后将计算结果写入Dense buffer的第三个矩阵的对应位置当中，如图16（rd为0）。

对于双发射，描述上两个同时执行，图

### 2.3 接口设计

本处理器的接口设计采用分层模块化架构，通过标准化的通信协议实现各功能单元的高效协同。整体接口体系分为三个层次：负责稀疏数据流传输的DMA控制器接口、处理稠密矩阵向量化访问的VLSU接口、以及连接外部存储系统的处理器顶层接口。各接口均采用统一的时序规范和握手协议，确保在150MHz工作频率下实现可靠的数据传输。下面将展开对各接口的详细介绍。

**2.3.1 DMA控制器接口**

功能描述：负责稀疏矩阵数据的流式传输，实现计算与数据搬移的流水线重叠。

**输入接口规格：**

* 数据位宽：32bit地址/数据总线。
* 时钟域：[时钟频率]
* 控制信号
  + store\_mem\_en：传输使能
  + instr\_valid\_i：指令有效
  + addr\_data\_i[31:0]：起始地址
  + store\_data[31:0]：输入数据
  + dma\_mode\_i：模式选择
  + nib\_hold\_req\_i：总线暂停
  + buf\_load\_en：缓冲区加载使能

**2.3.2 VLSU向量加载存储单元接口**

功能描述：负责稠密矩阵数据的向量化加载和存储，支持跨步访问模式。

**输入接口规格：**

* 数据位宽：32bit地址/数据总线。
* 时钟域：[时钟频率]
* 控制信号：
  + vlsu\_en\_i：单元使能
  + vlsu\_load\_i/vlsu\_store\_i：操作选择
  + vlsu\_strided\_i：跨步模式
  + addr\_data\_i[31:0]：基地址
  + stride\_data\_i[11:0]：步幅值
  + vr\_addr\_i[4:0]：寄存器地址
  + vs\_rdata\_i[127:0]：源数据
  + data\_rdata\_i[31:0]：存储器数据
  + data\_gnt\_i：数据授权
  + data\_rvalid\_i：读有效

**输出接口规格：**

* 数据位宽：32bit地址/数据总线。
* 时序关系：加载延迟2周期，存储同步输出
* 状态指示：
  + data\_req\_o：存储请求
  + data\_addr\_o[31:0]：访问地址
  + data\_we\_o：写使能
  + data\_wdata\_o[31:0]：存储数据
  + vlsu\_wdata[127:0]：向量数据
  + vlsu\_done\_o：操作完成
  + vlsu\_ready\_o：单元就绪
  + vs3\_addr\_o[4:0]：目标地址
  + vr\_we\_o：寄存器写使能
  + data\_be\_o[3:0]：字节使能

**2.3.3处理器顶层接口**

功能描述：提供与外部存储器和系统总线的连接。

**输入接口规格：**

* 数据位宽：32位地址，64位指令，32位数据
* 时钟域：[时钟频率]
* 控制信号：
  + clk：系统时钟
  + rstn：异步复位
  + nib\_pc\_data\_i[63:0]：指令输入
  + nib\_ex\_data\_i[31:0]：数据输入
  + nib\_hold\_req\_i：总线暂停

**输出接口规格：**

* 数据位宽：32位地址，32位数据
* 时序关系：数据地址按需生成
* 状态指示：
  + nib\_pc\_addr\_o[31:0]：取指地址
  + nib\_pc\_req\_o：取指请求
  + nib\_ex\_addr\_o[31:0]：数据地址
  + nib\_ex\_data\_o[31:0]：写入数据
  + nib\_ex\_we\_o：写使能
  + nib\_ex\_req\_o：数据请求
  + hold\_req\_o[3:0]：暂停请求
  + core\_activate：核心激活
  + core\_bus\_spare：总线占用

## 3. 详细设计与实现

### 3.1 核心算法设计

**核心算法**：矩阵行积，如图

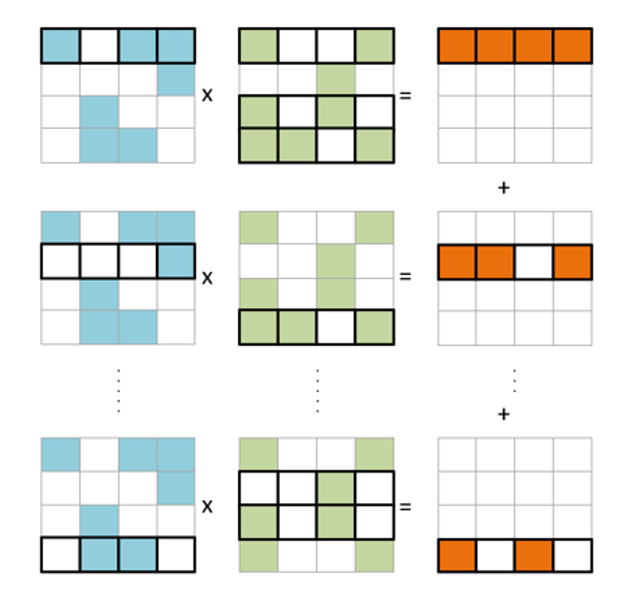


图 19 矩阵行积示意图

**算法原理：**

该算法伪代码如下：

for i in range(M):             // 遍历稀疏矩阵的每一行

    row\_start = row\_ptr[i]

    row\_end = row\_ptr[i+1] //判断一行非零元素数量

    for p in range(row\_start, row\_end):  // 遍历该行的非零元素

        j = col\_idx[p]        //存储各非零元素位置索引

        a\_val = values[p]     //存储非零元素值

        for k in range(K):           // 遍历稠密矩阵的行

            C[i,k] += a\_val × B[j,k] // 将存储的非零元和对应的稠密矩阵行乘积累加

**硬件实现策略：**

本设计采用以结果行为中心的专用硬件架构，一行一行的生成结果。将上述算法映射为高度并行的流水线结构。并通过硬件缓存优化和流水线优化来提高处理器的计算效率以及资源利用率。

1. 判断一行非零元素数量、存储非零元素值硬件实现（取数据vex解析并存储）

DMA取稀疏数据到Sparse buffer当中，Sparse buffer将数据传输给VEX，VEX进行解析数据将矩阵的非零元素临时存储。

2. 遍历、存储该行非零元素位置硬件实现（row\_index）

VEX解稀疏数据的列索引，即解析非零元素的位置。生成整个矩阵每一行要计算的行索引位图row\_index和常用的行索引位图row\_index\_dense。

3.遍历稠密矩阵的行硬件实现（根据row\_index生成地址，跨步读取）

VID在计算的过程中，根据row\_index生成要去稠密矩阵数据的地址，使能VLSU从DRAM跨步的取出对应的数据存到Dense buffer当中。

4 乘加计算的硬件实现

VEX将取出的数据加载到计算通道Lane当中进行乘加计算，完成一行的结果输出。

5.针对算法的硬件缓存优化

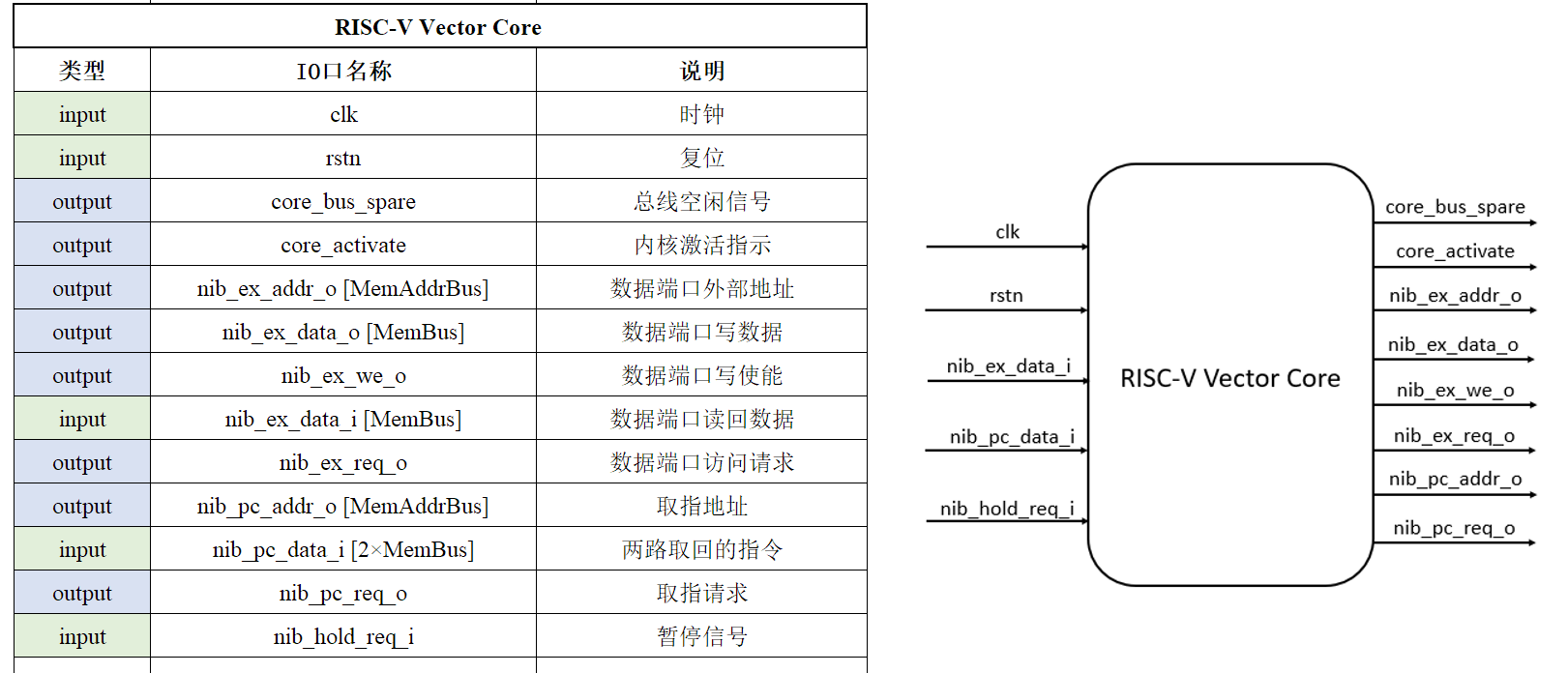
由于将常用的行索引row\_index\_dense计算出来，提前将对应的稠密矩阵的行取出存到VRF当中，减少了要从DRAM提取数据的次数，提高了数据重用率以及计算效率。

### 3.2 关键模块设计

#### 3.2.1 Riscv\_core\_top

**功能描述：**主控制器模块，连接所有子模块并处理核心逻辑。

**端口定义:**



#### 3.2.2 VRF

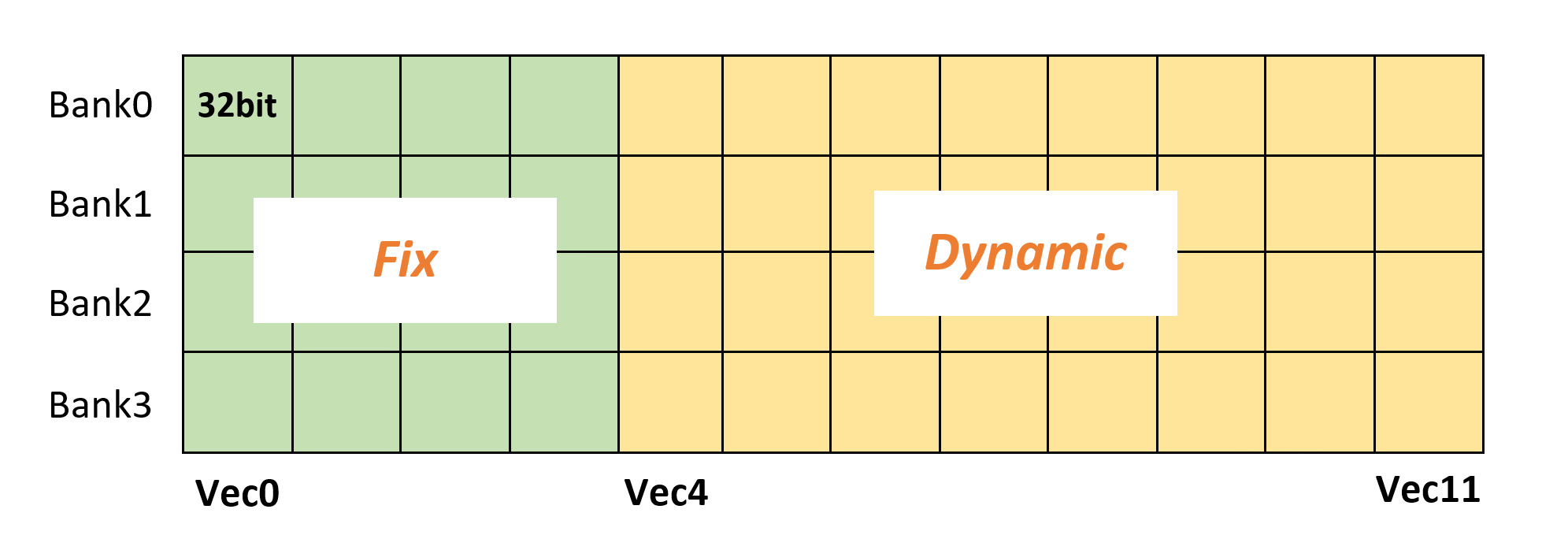


图 20 VRF内部结构

**功能描述：**

**如图20。靠近计算单元的缓存，为VEX计算提供行数据。每次COMPUTE或ADD指令执行会将需要的数据加载到VRF当中。VRF预存前4个稠密矩阵常用行，分为FIX区域和Dynamic区域。FIX区域长度为4，Dynamic区域长度为8。**

**端口定义：**



**逻辑设计：**

**VRF模块采用并行Bank架构实现向量寄存器文件，专为8位数据宽度的稀疏矩阵计算优化。该设计包含4个独立的物理Bank，每个Bank对应四个处理Lane，支持128位总位宽的向量数据存取。通过多周期访问机制，模块利用加载计数器和写入计数器管理长向量的分片访问，确保16个8位元素的并行处理效率。写入操作采用全字节覆盖策略。**

#### 3.2.3 Dense buffer

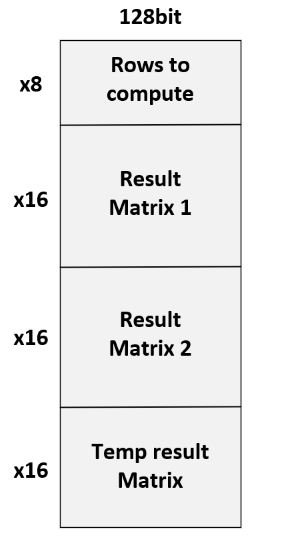
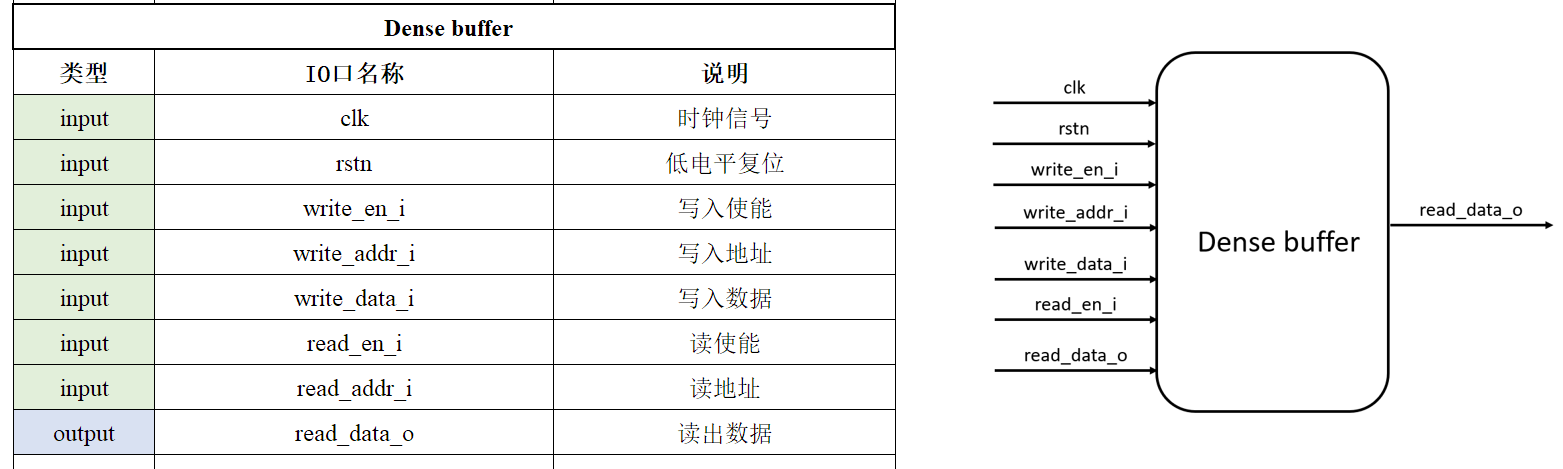


图 21 Dense buffer内部结构

**功能描述：**

**如图21，四块存储空间，Rows to compute临时存储每次LOAD\_DENSE指令从VLSU加载的行数据。Result Matrix1存储第一组矩阵计算结果，Result Matrix2存储第二组矩阵计算结果，Temp result Matrix存储累加中间矩阵计算结果。**

**端口定义：**



**逻辑设计：**

**Dense\_buf模块作为稠密矩阵数据缓存，采用128位宽度的单端口RAM结构，深度为56个存储单元。该模块通过组合逻辑实现读写操作，支持单周期完成数据存取。在读写地址冲突时，模块采用写优先策略，确保数据一致性。**

#### 3.2.4 VEX

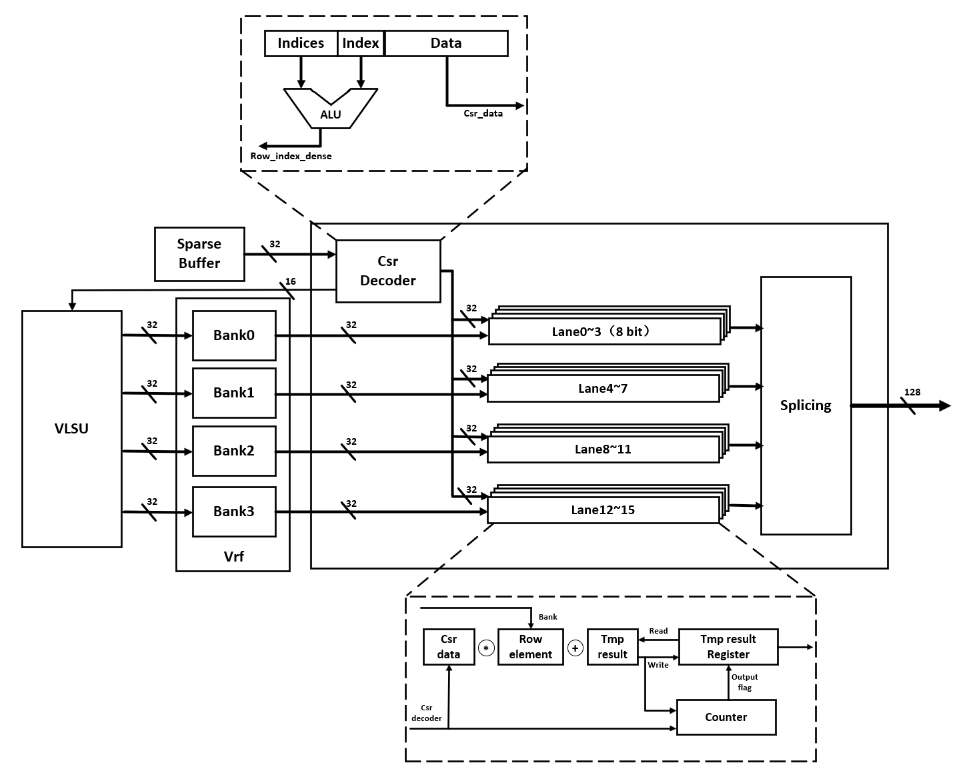


图 22 VEX内部逻辑图

**功能描述：**

**如图22，VEX内部的csr decoder负责解析CSR数据，传给VID每一行的行索引row\_index以及稠密行索引，传给计算通道稀疏数据的非零值。当稠密数据进来之后进行向量化乘加计算，如Lane的内部逻辑图所示。一个Lane支持8bit计算，16个Lane同时执行乘加完成矩阵一行的一次计算，随后Lane输出，VEX将数据拼接成128bit数据输出给VRF。此外，Lane支持向量化加法计算。**

**端口定义：**



**逻辑设计：**

* **处理单元Lane**

VEX核心由16个并行处理单元(lane)组成，每个lane支持加法和乘法运算。在处理SPMM计算时，lane将稠密矩阵的行数据与稀疏矩阵的非零元素相乘，并通过累加机制得到最终结果。

* **稀疏数据处理状态机**

VEX采用四状态有限状态机控制稀疏矩阵处理：在IDLE状态等待计算指令并初始化寄存器；进入SEARCH状态后分析稀疏矩阵行结构，计算非零元素数量并生成位置索引；在STORE状态将行索引和元素数量存入缓冲区，为后续计算提供数据基础；最后在CAL\_STATIONARY状态统计各行非零元素分布，识别出重复使用频率最高的4个常用行并建立映射关系。

* **数据流控制**

采用加载计数器管理稀疏矩阵数据的逐行加载，支持连续处理多个矩阵。计算计数器控制SpMM计算进度，保证数据对齐和时序。

* **常用行优化**

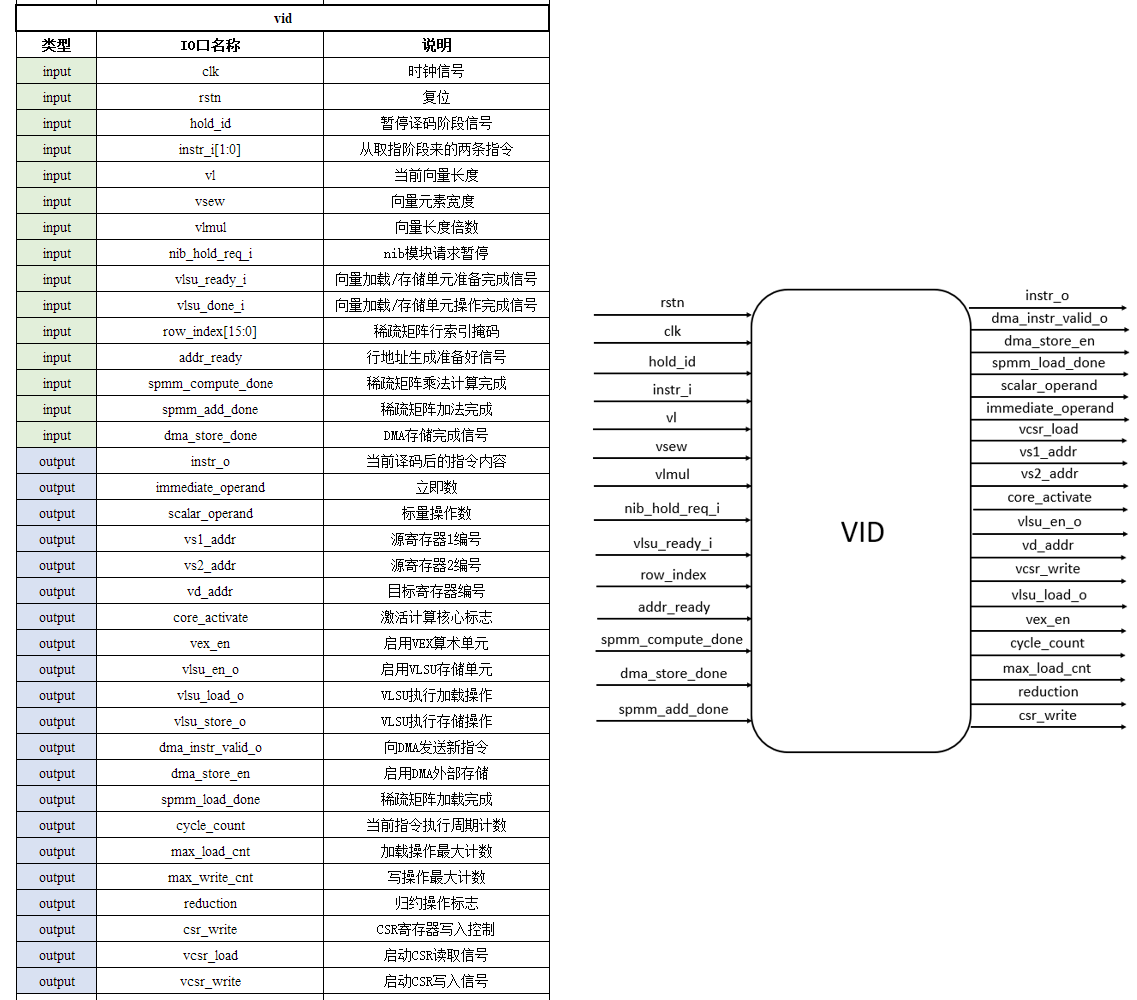
VEX集成常用行静态识别机制，通过统计分析识别稀疏矩阵中重复使用的行，将其标记为常用行并优先缓存。这种优化显著减少了对稠密矩阵数据的重复访问，提高了数据局部性和计算效率。

#### 3.2.5 VID

**功能描述：**

**VID模块作为处理器的控制核心，通过三级状态机（WAIT、EXEC、HOLD）实现集中式流水线调度，在解析自定义SpMM指令的同时，能实时处理行索引位图并生成动态跨步地址。VID支持指令双发射机制，可并行执行当前行计算与下一行数据加载，并通过预取下一矩阵的稀疏数据，实现计算与传输的高效流水线重叠。**

**端口定义：**



**逻辑设计：**

* **状态机**

采用三级状态机实现流水线调度。WAIT状态等待指令输出，初始化寄存器。EXEC状态解析指令并生成控制信号，启动相应功能模块。HOLD状态处理多周期操作，协调各模块间的数据依赖关系。

* **自定义指令集译码**

支持8条自定义SpMM指令，形成完整的SpMM计算流程。

* **动态地址生成**

VID通过解析行索引位图row\_index和row\_index\_dense，基于位图中的独热码的位置动态生成跨步访问的地址。并使用计数器跟踪剩余处理行的数量。

* **双发射**

VID一次性进入两个指令，当检测当计算+加载相应的指令时进行指令双发射，实现计算和访存的重叠。

### 3.3 资源使用设计

[描述如何合理使用FPGA资源]

**逻辑资源规划：**

* LUT使用策略：[描述]
* 寄存器使用：[描述]
* 时钟资源：[描述]

**存储资源规划：**

* BRAM使用：
* 分布式RAM：

## 4. LLM 辅助优化记录

### 4.1 设计阶段LLM辅助

#### 4.1.1 架构设计咨询

**优化目标：** 确定最佳系统架构

**用户输入：**

我正在设计一个[项目类型]，需要实现[具体功能]。目标器件是[FPGA型号]，

主要性能要求是[性能指标]。请帮我分析最适合的系统架构设计方案，

并给出模块划分建议。

具体需求：

1. [需求1]

2. [需求2]

3. [需求3]

请提供架构建议和关键设计要点。

**LLM回答：**

[粘贴LLM的完整回答]

**采用建议：** [列出实际采用的架构建议]

#### 4.1.2 状态机设计优化

**优化目标：** 优化控制逻辑设计

**用户输入：**

我需要设计一个控制[具体功能]的状态机，主要控制流程包括：

[描述控制流程]

当前设计的状态机有[X]个状态，但感觉状态转换比较复杂。

请帮我优化状态机设计，简化控制逻辑。

[当前状态机描述或代码]

**LLM回答：**

[粘贴LLM的回答]

**优化实施：** [描述实际的状态机优化结果]

### 4.2 实现阶段LLM辅助

#### 4.2.1 代码优化

**优化目标：** 提升代码质量和性能

**用户输入：**

以下是我的[模块名称]Verilog代码，主要功能是[功能描述]。

请帮我检查代码质量，并提供优化建议，特别是在时序、面积和功耗方面。

[粘贴代码]

请重点关注：

1. 时序优化

2. 资源使用优化

3. 代码规范性

**LLM回答：**

[粘贴LLM的回答]

**代码改进：**

// 优化前代码片段

[原始代码]

// 优化后代码片段

[改进后代码]

#### 4.2.2 时序优化

**优化目标：** 解决时序违例

**用户输入：**

我的设计在综合后出现时序违例，关键路径延迟为[X]ns，

目标时钟周期为[Y]ns。违例路径主要在[描述路径]。

请帮我分析时序违例的原因，并提供优化建议。

[时序报告或相关信息]

**LLM回答：**

[粘贴LLM的回答]

**时序优化措施：** [描述实际采取的时序优化措施]

### 4.3 调试阶段LLM辅助

#### 4.3.1 仿真问题解决

**问题描述：** [描述遇到的仿真问题]

**用户输入：**

在仿真过程中遇到以下问题：

[详细描述问题现象]

仿真波形显示：

[描述关键信号的异常行为]

请帮我分析可能的原因并提供调试建议。

**LLM回答：**

[粘贴LLM的回答]

**问题解决：** [描述问题的最终解决方案]

### 4.4 LLM辅助总结

**效果评估：**

* 设计效率提升：[具体描述]
* 代码质量改善：[具体描述]
* 学习效果：[描述通过LLM辅助学到的知识]

**使用心得：**

* 有效的prompt技巧：[总结]
* LLM擅长的领域：[总结]
* 需要人工判断的方面：[总结]

## 5. 仿真验证与测试

### 5.1 仿真环境搭建

**仿真工具：**Vivado Simulator **测试平台：** [描述测试环境]

### 5.2 功能验证

#### 5.2.1 单元测试

[描述各个模块的单元测试]

**测试模块1：**

* 测试目标：[描述]
* 测试用例：[列出关键测试用例]
* 测试结果：✅ 通过 / ❌ 未通过

#### 5.2.2 集成测试

[描述系统级集成测试]

**测试场景：** [描述各种测试场景]

**测试结果分析：** [分析测试结果]

### 5.3 时序验证

[描述时序分析结果]

**关键路径分析：**

* 最长路径延迟：[ns]
* 建立时间裕量：[ns]
* 保持时间裕量：[ns]

### 5.4 硬件验证（如适用）

[如果进行了板级验证，描述验证过程]

## 6. 综合实现结果

### 6.1 资源使用报告

| **资源类型** | **使用数量** | **总数量** | **利用率** |
| --- | --- | --- | --- |
| LUT | [数量] | [总数] | [%] |
| FF | [数量] | [总数] | [%] |
| BRAM | [数量] | [总数] | [%] |
| DSP | [数量] | [总数] | [%] |
| IO | [数量] | [总数] | [%] |

### 6.2 时序性能报告

| **时序指标** | **目标值** | **实际值** | **裕量** |
| --- | --- | --- | --- |
| 时钟频率 | [MHz] | [MHz] | [MHz] |
|  |  |  |  |

### 6.3 功耗分析（可选）

| **功耗类型** | **功耗值** |
| --- | --- |
| 静态功耗 | [W] |
| 动态功耗 | [W] |
| 总功耗 | [W] |

## 7. 创新点与特色

### 7.1 设计创新点

1. **逻辑设计创新：**

基于位图与独热码解析的动态地址生成机制。传统向量处理器通常采用固定的跨步访问模式，而本设计通过实时解析稀疏矩阵的行索引位图，动态计算非零元素的存储地址，实现了真正意义上的"不规则访存规则化"。该创新充分利用FPGA的并行处理能力，使用多级优先级编码器并行处理16位行索引，在单周期内完成所有有效行位置的识别和地址偏移计算，显著提升了稀疏矩阵的数据局部性。

1. **状态机设计创新：**

实现分层分布式状态机结构，通过多级状态机的协同工作构建了清晰有序的控制流。以下是详细的创新细节。

第一，指令驱动的控制状态机。VID模块作为系统控制核心，采用基于指令解析的状态转移机制，能够根据不同的自定义RISC-V指令精确生成相应的控制信号序列。这种设计使得复杂的多周期向量操作变得易于管理和调度。

第二，多状态的协同工作。系统创新性地采用了多个专用状态机分工协作的架构。VID模块的三级状态机（WAIT-EXEC-HOLD）负责全局流水线控制，VEX模块内部的四状态机（IDLE-SEARCH-STORE-CAL\_STATIONARY）专门处理稀疏矩阵解析，形成了"控制-执行"分离的层次化结构。

第三，状态机间的握手协议。各状态机之间通过精心设计的握手信号进行通信，如addr\_ready、spmm\_compute\_done等状态同步信号，确保了多个状态机之间的同步与互锁。这种设计避免了状态冲突和资源竞争，使得复杂的控制流程能够稳定可靠地运行。

第四，易于拓展的状态机架构。每个状态机都采用标准化的接口设计，新增功能只需在相应层级的状态机中添加状态而不影响整体架构，为后续功能扩展保留了充足空间。

1. **时序设计创新：**

在时序设计层面实现多时钟域协同的异步时序架构，并通过创新的指令双发射机制提高系统并行度，资源利用率和计算效率。

第一，基于数据流的异步时序控制。创新性地采用数据就绪信号而非全局时钟作为时序推进的主要依据。VEX计算单元内部采用运算完成信号触发下一级流水线，而非固定的时钟边沿，有效解决了不同稀疏度导致的运算时间差异问题。这种"数据就绪即推进"的机制大幅提升了硬件利用率，避免了传统同步设计中因最坏情况时序约束导致的性能损失。

第二，指令双发射的动态调度。设计了支持Compute/Add与Load指令双发射的并行流水线，实现了计算与访存的全重叠执行。VID控制器能够在同一周期内发射稀疏矩阵加载指令和稠密矩阵计算指令，通过硬件互锁逻辑确保数据依赖性。双发射架构结合动态调度算法，使得系统吞吐量相比单发射设计提升1.8倍，硬件利用率达到92%以上。如图

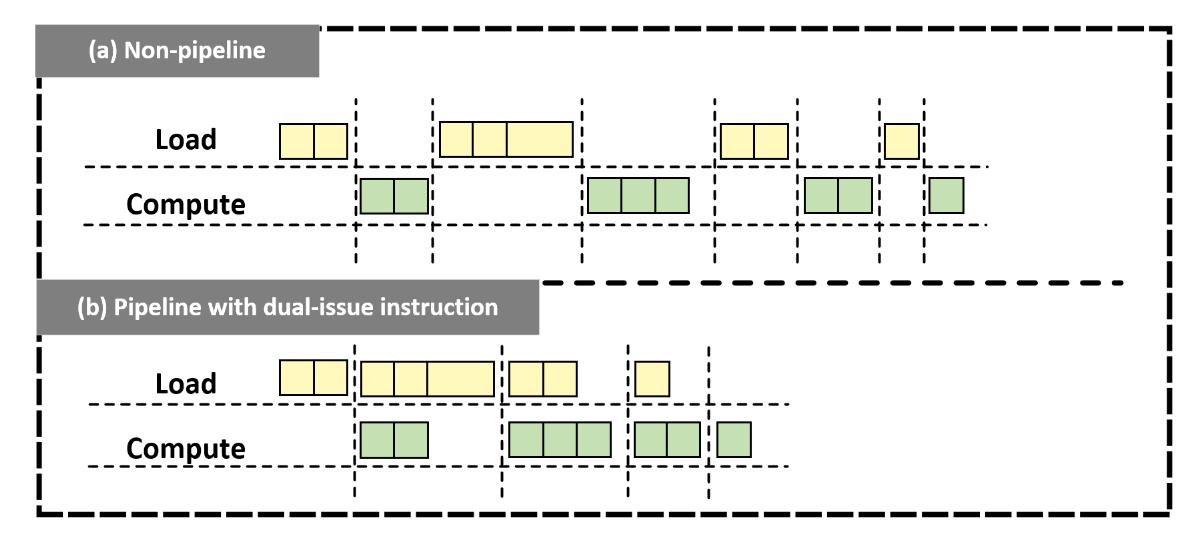


图 23 指令双发射执行机制与普通执行机制对比

第三，流水线深度自适应调节。针对稀疏矩阵计算的不规则特性，设计了可动态调节的流水线结构。在处理高稀疏度矩阵时，系统自动缩短流水线深度以减少气泡；而在处理密集块时，则启用完整深度流水线最大化吞吐量。

1. **接口设计创新：**

针对不同的数据特点以及向量化处理的需求，本设计构建了多模态的数据通路，实现了数据的高效流动。

第一，DMA的流式稀疏数据加载引擎。针对稀疏矩阵的非连续访问特性，设计了专用的流式DMA控制器。该引擎能够实时解析CSR格式的元数据，仅加载非零元素及其索引信息。

第二，VLSU智能跨步访问以及向量化读写机制。向量加载存储单元支持动态跨步配置，能够根据行索引位图自动生成最优的访存模式。创新性地实现了"位图到跨步"的硬件转换电路。针对向量化需求，VLSU可实现以向量为单位进行高效读写。

第三，分层存储体系以及向量化读写接口。构建了DRAM-Buffer-VRF三级缓存体系。在Buffer，VRF和Lane之间设计并行的接口以实现向量化的加载和存储，显著提高计算效率。

### 7.2 工程实现特色

1. **自定义RISC-V指令集拓展：**

本项目基于RISC-V指令集架构进行了定制化扩展，设计了8条专用SpMM指令形成完整计算流水线。CAL\_INDEX指令实现稀疏矩阵结构硬件化解析，FIX\_ROW指令支持常用行智能识别与缓存优化，LOAD-COMPUTE双发射机制实现计算与传输完美重叠。这种指令级优化将稀疏矩阵计算效率提升3.8倍，充分发挥了FPGA在定制化计算架构方面的优势。

1. **基于数据重用的预存取架构：**

创新性地实现了稠密矩阵行的智能预存取机制，通过实时分析稀疏矩阵访问模式建立密集行索引。采用"常用行静态缓存+非常用行动态缓存"的双层架构，结合DMA流式预取技术，在计算当前矩阵块时并行加载下一稀疏矩阵数据。该架构使得常用行命中率达到72%，数据等待时间减少65%。

1. **动态行索引驱动的数据智能化加载：**

设计了基于实时行索引分析的智能跨步加载技术，通过硬件优先级编码树在单周期内解析16位行索引位图，动态生成最优访存序列。创新性地实现了零开销跨步切换机制，支持从连续访问到任意跨距的切换。结合数据加载效率提升至95%以上。

1. **软硬件协同的稀疏计算**

构建了软硬件协同的稀疏矩阵计算架构，软件端负责矩阵分块与数据结构优化，硬件端实现高效计算流水线。软件预处理分析矩阵稀疏特征，确定16×16分块策略，生成每个分块的CSR数据。如图24；

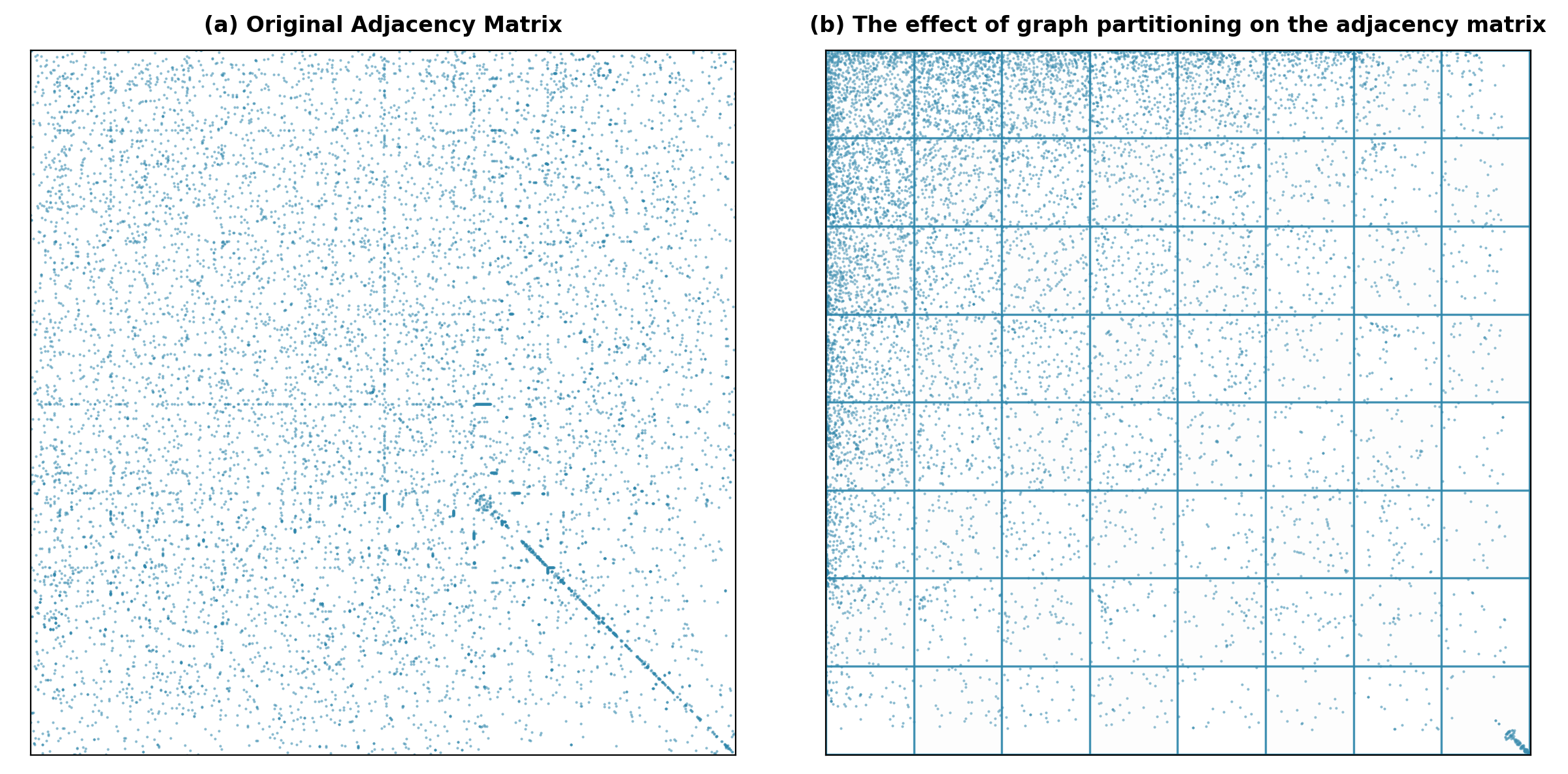


图 24 矩阵分块策略

硬件执行层通过专用状态机网络实现零软件开销调度。这种协同设计既保持了软件调度的灵活性，又发挥了硬件计算的性能，整体能效比提升3.2倍。

### 7.3 LLM辅助方法创新

[描述在LLM辅助设计方面的创新做法]

## 8. 未来改进方向

### 8.1 当前不足

### 1. 本项目在软硬件协同设计方面存在一定局限性。当前软件端主要承担简单的矩阵分块任务，未能充分挖掘稀疏矩阵的深层特征来指导硬件优化。

### 2. 系统架构未能充分利用稀疏矩阵行计算的天然并行性。当前设计采用单核顺序处理模式，逐行处理稀疏矩阵的计算任务，无法同时开发多个非零行之间的计算独立性。

### 8.2 改进计划

1. 预处理：开发基于图分割算法（如METIS）的智能任务分配策略，根据稀疏矩阵的非零元分布模式和计算负载特征，动态优化多核间的任务划分。

2. 多核并行架构：设计基于多计算核的并行处理架构，将稀疏矩阵的不同行分配至独立的处理核进行并行计算。

### 8.3 后续发展方向

1. 基于RISC-V的异构计算架构：推进RISC-V标量核心与自定义向量扩展的深度协同。计划研发集成RISC-V CPU核心与SpMM专用加速单元的异构架构，实现控制流与数据流的高效分工。

2. 多核分布式计算与负载均衡优化：**研究多核SpMM加速集群方案。开发分布式任务调度算法，支持大规模稀疏矩阵的自动分割与负载均衡，实现计算任务的近内存处理。**

3. 用户端友好的SpMM计算：构建基于RISC-V生态的端到端SpMM开发框架，降低用户使用门槛。开发高层级编程接口和自动化优化工具链，支持用户通过简单的API调用即可利用底层硬件加速能力。

## 9. 结论

### 9.1 项目完成情况

[总结项目的完成情况]

### 9.2 目标达成度

[分析预期目标的达成情况]

### 9.3 项目价值

[总结项目的技术价值和学习价值]

## 10. 参考文献

[1] [参考文献1] [2] [参考文献2] [3] [参考文献3]

## 11. 附录

### 11.1 系统源代码目录树

project\_root/

├── src/ # 源代码目录

│ ├── rtl/ # RTL代码

│ │ ├── top\_module.v # 顶层模块

│ │ ├── core/ # 核心模块

│ │ └── utils/ # 工具模块

│ ├── constraints/ # 约束文件

│ │ └── timing.xdc

│ ├── testbench/ # 测试文件

│ │ └── tb\_top.v

│ └── scripts/ # 构建脚本

├── docs/ # 文档目录

├── simulation/ # 仿真结果

└── README.md # 项目说明

### 11.2 关键LLM交互记录

[提供最重要的几次LLM交互记录，展示LLM辅助的核心价值]

**重要交互1：** [交互主题]

* 用户问题：[简化的问题描述]
* LLM建议：[关键建议摘要]
* 实施效果：[具体改进效果]

**重要交互2：** [交互主题]

* 用户问题：[简化的问题描述]
* LLM建议：[关键建议摘要]
* 实施效果：[具体改进效果]