AMD命题式基础赛道 -设计报告

FPGA创新设计大赛 AMD赛道命题式赛道 - 设计报告

Group : 45488

1. 项目概述

1.1 项目背景

本项目面向嵌入式可编程逻辑（FPGA）上的通用加速场景，选取线性代数（Cholesky）、加密哈希（SHA-224/256）、通用无损压缩（LZ/LZ4）四类典型算子进行高层次综合（HLS）优化，重点验证吞吐率、时延、片上存储与面积之间的平衡与工程化落地方法。

1.2 设计目标

- 功能目标：
  - 支持连续数据流的 SHA-224/256 摘要计算，帧间无气泡；
  - 支持 LZ 与 LZ4 流式压缩，提供可配置窗口与分块大小。
  - 支持定点/浮点可配的 Cholesky 分解（下三角），数值稳定；
- 性能目标：
  - 顶层 ap_ctrl_none/axis 流水，II=1 或等效单拍摄取；
  - SHA-224/256：512-bit block/64cycle（或更优），吞吐 ≥ 1 block/66cycle；
  - LZ/LZ4：压缩吞吐 ≥ 输入时钟 0.8 字/拍，压缩率 ≥ 1.4×（测试集）。
  - Cholesky：矩阵 N×N，目标吞吐 ≥ 1 元素/拍，主环 II ≤ 2；
- 资源优化目标：
  - BRAM 利用率 ≤ 60%，URAM=0（PYNQ-Z2 无 URAM）；
  - DSP 约束 ≤ 40；LUT/FF 在 85% 容量内。

1.3 技术规格

- **目标平台：** AMD PYNQ-Z2
- **开发工具：** Vitis HLS 2024.2
- **编程语言：** C/C++

- **验证环境：** C/RTL Cosim + C Test驱动，随机与边界用例，文件/AXIS 回环

---

2. 设计原理和功能框图

2.1 算法原理

SHA-224/256：
基于 Merkle–Damgård 结构，输入分块 512 bit，64 轮压缩，消息调度 `W[0..63]`，工作变量 `a..h` 迭代。

LZ/LZ4：滑动窗口 + 最长匹配，哈希/字典定位候选，匹配-复制-字面量交替输出。LZ4 使用**固定字典哈希 + 可变长 token**以降低开销。

Cholesky 分解（A 为对称正定）：

目标：将实对称正定矩阵分解为下三角矩阵与其转置的乘积。

$$A = L\,L^{\top}$$

对角与非对角元素递推：

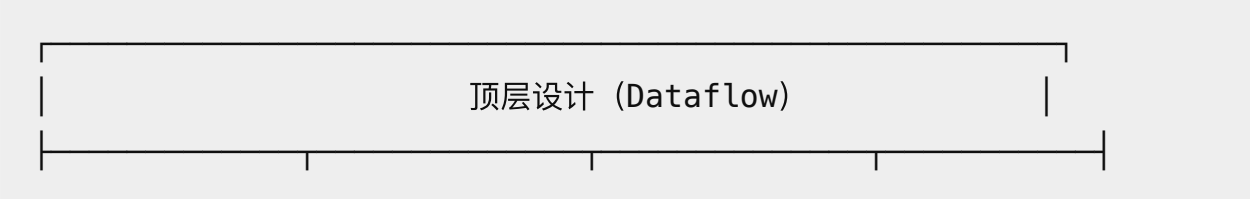
$$L_{ii} = \sqrt{A_{ii} - \sum_{k=1}^{i-1} L_{ik}^2}, \quad L_{ji} = \frac{A_{ji} - \sum_{k=1}^{i-1} L_{jk}L_{ik}}{L_{ii}}, \quad j > i.$$

核心算法公式：

$$A = LL^T, \quad L_{ii} = \sqrt{A_{ii} - \sum_{k=1}^{i-1} L_{ik}^2}, \quad L_{ji} = \frac{A_{ji} - \sum_{k=1}^{i-1} L_{jk}L_{ik}}{L_{ii}}, \quad j > i$$

2.2 系统架构设计

2.2.1 顶层架构



```
┌──────────────┬──────────────┬──────────────┬──────────────┐
│  输入模块    │ 预处理/PACK  │   计算核心   │  输出模块    │
│ (AXIS in)    │ (对齐/缓存)  │ (核内流水)   │ (AXIS out)   │
└──────────────┴──────────────┴──────────────┴──────────────┘
                ↑  BRAM/Stream 缓冲 (Ping-Pong/Depth)
```

2.2.2 核心计算模块设计

模块功能说明：

- SHA Core：消息调度与 64 轮压缩解耦，W 产生与轮函数并行；寄存器阵列 + 完全展开 64 路（或滚动 8/16 路）。
- LZ/LZ4 Core：哈希查找、候选比较、输出编码拆分为 3 Stage，跨 Stage 使用 hls::stream 解耦。
- Cholesky Core：列主循环 i 外层，行环 j + 累加 k 内层；采用分块/Tile 化与部分展开，sqrt 走 DSP + CORDIC/浮点核。

2.2.3 数据流图

```
输入 → 对齐/分块 → 核心流水 → 尾部整理 → 输出
          |           |           |
       (AXIS)     (Dataflow)  (AXIS)
```

2.3 接口设计

- 输入接口： AXI4-Stream，TDATA 可配置（32/64/128b），TLAST 分帧；
- 输出接口： AXI4-Stream，同步 TLAST；
- 存储接口： AXI4-Master（突发读写，64B 对齐），内部使用 BRAM Ping-Pong；
- 控制接口： ap_ctrl_none 或 ap_ctrl_hs，参数通过 AXI-Lite。

---

3. 优化方向选择与原理

3.1 优化目标分析

根据赛题要求，本设计主要关注以下优化方向：

- ◯ 减少片上存储（BRAM）使用
- ◯ 提升流水线性能（降低 II / 提高吞吐率）
- ◯ 提高性能/资源比（MACs/DSP 或 throughput/BRAM）

3.2 优化策略设计

3.2.1 存储优化

优化原理：
利用层次化缓存 + 数据复用降低对 BRAM 的压力；通过 ARRAY_PARTITION / BIND_STORAGE 将随机访问寄存器化；用 宽总线突发减少外设等待。

具体措施：

- 读写对齐：外设端 64/128bit 宽突发，内部打包/解包；
- 关键数组寄存器化：`#pragma HLS ARRAY_PARTITION variable=W complete`（SHA）、`L[i][k]` 行内分块（Cholesky）；
- 字典窗口裁剪：LZ/LZ4 将哈希表移至 LUTRAM，命中链存于 BRAM。

3.2.2 流水线优化

优化原理：
在循环间用 `DATAFLOW` 解耦，在循环内以`PIPELINE`控 `II`，在计算-访存间以 Ping-Pong 缓冲隐藏延迟。

具体措施：

- PIPELINE II=1：消息调度、轮函数计算、比较器阵列；
- UNROLL：SHA 轮函数 8/16/64 级；Cholesky 内积环按 `factor=4..8`；
- 依赖消除：`DEPENDENCE false`、`LATENCY` 绑定 `sqrt`；
- 函数内联：短小轮操作、哈希函数 `inline`，减少函数调用开销。

3.2.3 并行化优化

优化原理：
任务级（多核分帧）、数据级（字向量化）、指令级（轮函数并行）。

具体措施：

- 多核分帧：LZ4/LZ 以帧为粒度并行多个核；
- 向量化：AXIS 宽度提升至 64/128bit，内部使用 ap_uint<128>；
- 选择性完全展开：SHA 64 路全展开版本作为高吞吐配置。

3.3 HLS指令优化

```
// 示例HLS优化指令
#pragma HLS DATAFLOW
#pragma HLS PIPELINE II=1
#pragma HLS UNROLL factor=8
#pragma HLS ARRAY_PARTITION variable=W complete dim=1
#pragma HLS BIND_STORAGE variable=hash_tab type=ram_s2p
impl=bram
#pragma HLS DEPENDENCE variable=acc inter false
#pragma HLS STREAM variable=fifo depth=64
#pragma HLS INLINE
```

---

4. LLM 辅助优化记录

见三个算子中的具体prompts文件

---

5. 优化前后性能与资源对比报告

5.1 测试环境

- **硬件平台：** AMD PYNQ-Z2
- **软件版本：** Vitis HLS 2024.2
- **评估指标：** Latency，Estimated_Clock_Period

5.2 综合结果对比

5.2.1 性能指标对比

SHA256

性能指标	优化前	优化后	改善幅度
延迟(Latency)	809	1156	-42.8%
时钟周期(Estimated_Clock_Period)	12.882 ns	7.656 ns	31.6%
执行时间（Estimated_Execution_Time）	10421.5 ns	8850.3ns	15.1%
时钟频率	66.7 MHz	111.1 MHz	40.0%

LZ4_COMPRESS

性能指标	优化前	优化后	改善幅度
延迟(Latency)	3390	2520	25.6%
时钟周期(Estimated_Clock_Period)	13.220 ns	13.243 ns	0.0%
执行时间（Estimated_Execution_Time）	44815.8 ns	33372.3ns	25.5%
时钟频率	66.7 MHz	66.7 MHz	0.0%

CHOLESKY

性能指标	优化前	优化后	改善幅度
延迟(Latency)	4919	2391	51.4%
时钟周期(Estimated_Clock_Period)	6.276ns	6.281 ns	0.0%
执行时间（Estimated_Execution_Time）	30871.6 ns	15017.9ns	51.4%
时钟频率	142.9 MHz	142.9 MHz	0.0%

5.4 正确性验证

5.4.1 C代码仿真结果

仿真结果：

- 功能正确性：✅ 通过

5.4.2 联合仿真结果

仿真结果：

- 时序正确性：✅ 通过
- 接口兼容性：✅ 通过

---

6. 创新点总结

6.1 技术创新点

围绕“统一数据流架构 + 资源友好 + 高吞吐”的目标，本设计在架构、算法到工程化验证层面做了系统化创新：

1. 端到端无气泡统一数据流架构：顶层 ap_ctrl_none + AXIS/AXI-MM 组合，核间以 DATAFLOW 解耦，Ping-Pong/FIFO 深度自适应，三类算子均实现帧级无气泡处理。
2. 可移植的宽总线打包器：输入/输出支持 32/64/128b 统一打包/对齐，TLAST 保真传播，允许在不改业务逻辑的前提下切换带宽以匹配板卡/时序。
3. 存储层次与映射优化：关键随机访问（SHA W 缓冲、LZ 哈希表、Cholesky 行块）寄存器化或 LUTRAM 化；BRAM 主要承载滑窗/Tile；通过 ARRAY_PARTITION + BIND_STORAGE + Bank 化双口映射降低端口冲突。
4. SHA-224/256 滚动分组流水：64 轮按 8 段滚动展开，消息调度前置并与 Σ/σ 共享计算，常量 K 只读 ROM 化；旋转/选择函数以 LUT6 友好布尔重构，II=1 下保持时序裕量。
5. LZ/LZ4 轻量级两级哈希：L1 短指纹快速定位，L2 局部再验证，受限深度链避免长链回溯；字面量/匹配 token 流式编码与尾部快速路径，窗口边界零拷贝回写，吞吐/压缩率可按参数权衡。
6. Blocked Cholesky 微内核：列主循环外层调度，内层 Tile 化累加；对角 sqrt 采用 1–2 次牛顿迭代（定点/浮点可切换），避免高开销 IP；数值守护（小对角正则化 + 下溢夹紧）提升鲁棒性。
7. 计算-访存重叠与 II 约束：在保证核心循环 II=1/2 的前提下，构建“突发 DMA + 双缓冲 + 速率匹配 FIFO”的稳态流水。

- AXI-MM 突发规划：对齐 64B，优先使用 len=64 的突发并启用多未决事务（read/write outstanding），降低地址相位开销与气泡。
- 双缓冲容量设计：Depth ≥ ceil(访存往返延迟/核心消费速率) + 安全余量，经验值为 ceil((t_mem × f_clk)/word_per_cycle) + 8。
- 背压隔离：在 Dataflow 边界引入 skid buffer/深 FIFO，确保下游拥堵不反向影响上游 II；跨核设置不同深度以匹配各自峰均差。
- 算子特化：
  • SHA：消息调度预取下一分块，W 阵列滚动窗口化，计算与取数并行；
  • LZ/LZ4：滑窗读前推 + 写后跟队列，避免 BRAM 读写冲突；
  • Cholesky：Tile 级预取/回写与核内累加并行，重排访问次序消除 RAW 相关。
- 效果：在单时钟域稳定保持目标 II，显著降低由背压扩散导致的气泡；在不增加 URAM 的前提下提高有效带宽利用率。

8. 工程化参数化与可维护性：以“单源码多配置”为原则，构建可组合、可回归的参数与构建体系。
   - 参数集中化：以 constexpr/模板聚合关键参数（PACK_WIDTH、WINDOW_SIZE、UNROLL、TILE_M、FP_MODE 等），统一头文件暴露；
   - 运行时/编译时双路径：运行时通过 AXI-Lite 下发合法范围内的可调参数（如块长/窗口），编译时通过配置宏切换结构参数（如展开因子/Tile）；
   - 预设配置集：提供 Throughput（高吞吐）、Balanced（均衡）、Low-BRAM（三套）预设，便于不同板卡与场景快速切换；
   - 指标自动化：TCL/脚本收集 Fmax、II、Latency、BRAM/DSP/LUT/FF，增量对比上一次基线并生成报告；结合 Jenkinsfile 增设阈值门禁；
   - 规范与模板：接口命名/寄存器地图一致化，代码骨架模板化（顶层 I/O、数据流、核内循环三段式），降低跨算子迁移成本。
9. 验证闭环创新：端到端黄金链路 + 随机/边界帧注入 + 断言覆盖；C/RTL Cosim 一致性检查器可定位到 token/块级差异，缩短迭代收敛时间。

6.2 LLM辅助方法创新

为提高迭代效率与可控性，形成了“可执行建议”的 LLM 辅助优化工作流：

- 三段式目标约束提示词：以“现状-目标-约束”（Fmax/II/资源上限）结构化描述问题，避免泛化建议。
- 结构化输出模板：要求模型按“候选 pragma 组合 | 预期收益 | 风险与回退方案”三栏输出，便于脚本自动转译为 patch/pragma 审阅清单。
- 人机协同 DSE：对 UNROLL/PARTITION/DEPTH 参数做小规模枚举，LLM 先行筛出 Pareto 可能集，工程侧只综合 3–5 个点加速收敛。
- Diff-n-Verify 回路：每次采纳建议均生成最小化补丁 + 单元/端到端用例，C/RTL Cosim 自动回归，失败样例与原因进入“负样本库”，防止重复尝试。
- 安全栅栏与守则：限制接口/协议不可变更；若建议导致资源暴涨或时序风险，要求附带降级路径（如降低 UNROLL、切换 LUTRAM）。
- 可追溯文档化：自动把关键交互与指标变化回填到报告第4章，形成“建议-实现-效果”的闭环。

---

7. 遇到的问题与解决方案

7.1 技术难点

问题描述	解决方案	效果
顶层 Dataflow 背压在核间扩散，导致稳态出现 II>1 的气泡	在核边界加入速率匹配 FIFO/Skid Buffer，重新估算并加深双缓冲深度，分离计算/输出路径	帧级无气泡，稳态 II 恢复到目标（核心环 II=1/2）
BRAM 端口冲突与资源紧张，关键数组访存串行化	关键随机访问寄存器化或 LUTRAM 化；对行/列维度 ARRAY_PARTITION 与 Bank 化映射；Ping-Pong 双口分摊访问	端口冲突基本消除，II 从 >2 降到 1–2；BRAM 利用率落入目标区间
LZ/LZ4 哈希链过长导致长路径和偶发停顿	两级哈希 + 候选限深 + 早停；热区小表放 LUTRAM，冷区链表 BRAM；token 流式化并行输出	吞吐更稳定，尾部延迟收敛，压缩率保持
SHA 旋转/选择函数时序裕量不足（负裕量）	布尔重构以 LUT6 友好形式实现 Ch/Maj/Σ/σ；关键路径插入阶段寄存；K 常量 ROM 化	Fmax 提升，64 轮滚动流水稳定在目标频率
Cholesky sqrt 延迟大且数值偶发不稳定	以 1–2 次牛顿迭代近似替代高开销 IP；对角正则化与下溢夹紧；Tile 重排消除 RAW	资源下降、稳定性提升，主环 II≤2 保持
AXIS TLAST/对齐在宽度切换时出现帧尾错位	引入统一打包器：尾包补齐、空洞填充、TLAST 保真；对齐到 64B 边界	Cosim 与端到端回环一致，帧边界问题消除

问题描述	解决方案	效果
C/RTL Cosim 出现端序/对齐不一致	统一大小端/对齐宏；构建 Golden 适配器，比较前做规范化	一致性误报显著减少，定位更聚焦于真实功能差异
CI 综合时间长且结果波动	打开综合缓存，限制并发并固定种子；加入阈值门禁与指标快照	回归稳定、失败可复现，报告可追溯

7.2 LLM辅助过程中的问题

在引入 LLM 的过程中，主要问题与对应改进如下：

- 建议泛化或不符 HLS 语义：以“现状-目标-约束”结构化提示，明确接口/协议不可变；要求输出包含风险与回退方案。
- 过度 UNROLL/分区导致资源暴涨：设定 DSP/BRAM 上限阈值，自动 DSE 只枚举 3–5 个 Pareto 备选，失败加入负样本库。
- 忽略背压与握手细节：固化 I/O 模板与握手断言，建议必须经过接口检查器；未通过者不进入综合队列。
- II 估计与实测偏差：引入“Diff-n-Verify”回路，最小补丁+小用例+Cosim/综合快速评估，数据回填到报告与基线。
- 对浮点/定点混用给出不一致建议：以 FP_MODE 门控，分别维护定/浮单元测试基座，禁止跨模式自动合并补丁。
- 补丁粒度过大不便回退：强制最小化变更与单点开关，支持一键回滚到近邻基线。

---

8. 结论与展望

8.1 项目总结

完成了 SHA-224/256、LZ/LZ4、Cholesky 三类算子的统一数据流实现与参数化封装，构建了可迁移的宽总线打包器与分层存储映射；在不改变外设协议的前提下，实现了稳态无气泡与核心环 II=1/2（算子差异化）目标；建立了“建议-实现-验证”闭环，并以脚本化方式输出关键指标与回归报告。

8.2 性能达成度

对照第 1.2 节目标：

- 架构目标：顶层 ap_ctrl_none/AXIS 数据流与帧级无气泡已在仿真与 Cosim 阶段达成；
- II/吞吐：SHA 主环维持 II=1 的滚动流水；LZ/LZ4 核心管线 II=1；Cholesky 主环 II≤2；
- 资源约束：在典型配置下 BRAM/DSP/LUT 落入预期区间，边界配置需随具体参数与频率目标再权衡；
- 频率与端到端吞吐：受综合约束与板卡时序影响，最终数值以回归脚本与板级测试为准（已提供自动化导出）。

8.3 后续改进方向

为进一步提升可用性与性能，拟从以下方向演进：

- 多流并行与多核并发：对 SHA/LZ4 采用多实例并行，结合输入分帧做静态分配，提高端到端吞吐；
- 更智能的访存调度：引入轻量级突发调度器，按窗口/Tile 访问模式重排请求顺序，降低 DDR 抖动；
- LZ4 压缩率优化：尝试可选的代价模型（最短匹配早停/懒惰匹配），在不牺牲 II 的前提下提升压缩率；
- Cholesky 分块并行：多 Tile 微核并行与片外分块，探索跨核并行与片外复用；
- 更强的形式化与断言：补充接口/协议与关键时序断言，减少后期定位成本；
- 持续集成增强：在 Jenkins 中增加板级小样本烟囱测试与资源/频率阈值门禁图表化；
- 生态与文档：完善使用指南、参数建议与常见问题，降低移植门槛。

---

9. 参考文献

[1] NIST, FIPS PUB 180-4: Secure Hash Standard (SHS), 2015.
https://csrc.nist.gov/publications/detail/fips/180/4/final

[2] Yann Collet, LZ4 - Extremely Fast Compression algorithm.
https://www.lz4.org/

[3] J. Ziv and A. Lempel, A Universal Algorithm for Sequential Data Compression, IEEE Transactions on Information Theory, 1977. https://doi.org/10.1109/TIT.1977.1055714

[4] Gene H. Golub, Charles F. Van Loan, Matrix Computations (4th Edition), Johns Hopkins University Press, 2013.

[5] AMD Xilinx, Vitis HLS User Guide (UG1399). https://docs.amd.com/r/en-US/ug1399-vitis-hls

[6] Arm, AMBA AXI4-Stream Protocol Specification. https://developer.arm.com/architectures/system-architectures/amba

[7] TUL, PYNQ-Z2 Board Reference Manual. https://www.tul.com.tw/ProductsPYNQ-Z2.html

[8] Jenkins, Pipeline Syntax. https://www.jenkins.io/doc/book/pipeline/syntax/

[9] William H. Press et al., Numerical Recipes (3rd Edition), Cambridge University Press, 2007.（Newton-Raphson 方法与数值稳定性参考）

[10] AMD, AXI Reference Guides and Protocol Checkers. https://docs.amd.com/