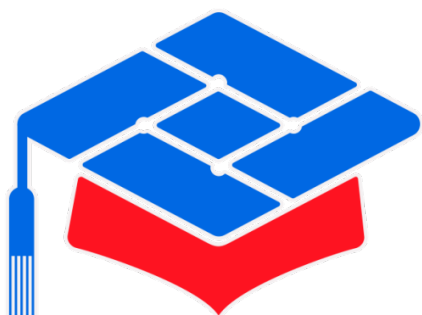




第十届 全国大学生集成电路创新创业大赛

报告类型：设计报告

参赛杯赛：龙芯中科杯

作品名称：ML-KEM 后量子密钥封装硬件 IP 核设计与验证

队伍编号：CICC1008439

团队名称：碳基生物

ML-KEM 后量子密钥封装硬件 IP 核 设计与验证

碳基生物

团队编号: CICC1008439

龙芯中科杯·2026 全国大学生集成电路创新创业大赛

目录

1 绪论	3
1.1 设计背景	3
1.2 关键设计特性	3
1.3 代码规模	3
2 IP 模块规格	4
2.1 算法参数	4
2.2 顶层端口信号	4
3 总体架构设计	5
3.1 设计原则	5
3.2 模块层次结构	5
3.3 顶层 FSM 调度	6
4 各功能模块数据通路与控制逻辑	8
4.1 Keccak-f[1600] 置换核与哈希族	8
4.2 NTT 核	8
4.3 多项式乘法与 Barrett 模乘	10
4.4 采样模块	10
4.5 压缩与解压缩	11
4.6 多项式存储库	11

5	验证方案	11
5.1	分层验证策略	11
5.2	测试平台与 KAT 测试向量	12
5.3	系统级诊断测试	12
6	实现结果与测试数据	13
6.1	FPGA 资源利用	13
6.2	时序性能	14
6.3	功耗分析	14
6.4	功能验证结果	15
6.4.1	KAT 逐字节比对	15
6.4.2	系统级诊断测试	15
6.5	性能分析	15
6.6	逐状态性能分解	16
6.7	其他安全等级结果	18
7	设计评估与对比	18
7.1	设计局限性分析	18
7.2	与现有工作的对比	19
7.3	优化路线图	20
8	结论与展望	20

1 绪论

1.1 设计背景

ML-KEM (Module-Lattice-based Key-Encapsulation Mechanism, FIPS 203) 是 NIST 于 2024 年 8 月发布的后量子密码标准, 用于替代 RSA 和 ECC 在密钥交换中的角色。其安全性建立在模格上的带误差学习问题 (MLWE) 之上, 多项式环为 $\mathbb{Z}_q[x]/(x^{256} + 1)$, 素数模 $q = 3329$, 多项式次数 $n = 256$ 。

软件实现的 ML-KEM 在嵌入式处理器上存在显著性能瓶颈——以 ML-KEM-768 为例, 一轮完整的密钥生成、封装和解封装在通用 CPU 上需要数百万个时钟周期。本文设计的硬件 IP 核 `mlkem_top` 旨在通过专用数据通路和计算资源时分复用, 在 FPGA 平台上实现高性能、低面积的 ML-KEM 加速。

1.2 关键设计特性

本 IP 核在单一硬件模块内具有以下特性:

- (1) **运行时三等级支持**: 通过 $k_i \in \{2, 3, 4\}$ 端口在运行时选择 ML-KEM-512/768/1024 安全等级, 存储资源按 $K_{MAX} = 4$ 静态分配, 无需重综合。
- (2) **全操作覆盖**: 支持 KeyGen (FIPS 203 算法 16)、Encaps (算法 17) 和 Decaps (算法 18, 含 FO 变换隐式拒绝)。
- (3) **Keccak 核跨阶段复用**: 单一 `keccak_core` 实例在 G/H/J 哈希、SampleNTT 采样和 CBD 采样三个消费者之间时分复用, 是面积优化的最大贡献因素。
- (4) **Decaps 重加密路径复用**: D6 步的 K-PKE 重加密直接复用整条 Encaps 流水线的全部 FSM 状态, 避免重复实现加密数据通路。
- (5) **逐字节流式 I/O**: 公钥 ek 、私钥 dk 和密文 c 通过 BRAM 窗口逐字节装载与回读, 免去超宽顶层总线。
- (6) **NIST KAT 完全验证**: 三种等级、三种操作全部逐字节比对 NIST FIPS 203 KAT 标准答案通过。

1.3 代码规模

RTL 代码总计约 5,517 行 (不含测试平台), 其中 `mlkem_top.v` (顶层 FSM 集成 + 数据通路 + 三库存储管理) 约 2,264 行, 各叶子算子模块合计约 3,253 行。全部采用 SystemVerilog 编写, 统一使用 `valid/ready` 流式握手接口。

2 IP 模块规格

2.1 算法参数

`mlkem_top` 的运行时参数由输入端口 k_i 和 op_i 在 `start_i` 脉冲时刻锁存至内部寄存器 k_r 和 op_r ，后续所有数据通路宽度、存储子区间基地址和循环次数均由 k_r 推导。表1列出三种安全等级的密钥和密文尺寸。

表 1: 运行时安全等级参数

k_i	方案	K	η_1	(d_u, d_v)	ek(B)	dk(B)	ct(B)
2	ML-KEM-512	2	3	(10,4)	800	1632	768
3	ML-KEM-768	3	2	(10,4)	1184	2400	1088
4	ML-KEM-1024	4	2	(11,5)	1568	3168	1568

固定参数：模数 $q = 3329$ 、多项式次数 $n = 256$ 、 $\eta_2 = 2$ （所有等级）、共享密钥 32 字节。

2.2 顶层端口信号

`mlkem_top` 的端口按功能分为四组。表2列出主要信号。

表 2: `mlkem_top` 主要端口

信号	位宽	功能
<code>clk, rst_n</code>	1, 1	时钟 (50 MHz) 与低电平异步复位
<code>k_i, op_i</code>	3, 2	安全等级 (2/3/4) 与操作码 (0=KG, 1=Enc, 2=Dec)
<code>d_i, z_i, msg_i</code>	256×3	KeyGen 种子 d 、隐式拒绝种子 z 、Encaps 消息 m
<code>start_i</code>	1	启动脉冲，锁存 k_i 和 op_i
<code>busy_o, done_o</code>	1, 1	运行标志与完成脉冲
<code>ss_o</code>	256	共享密钥输出 (<code>done_o</code> 时有效)
<code>ek_in_we/addr/byte</code>	1,11,8	Encaps 公钥流式预装载
<code>dk_in_we/addr/byte</code>	1,12,8	Decaps 私钥流式预装载 (自动路由至 <code>dkp_bram/ek_bram/寄存器</code>)
<code>c_in_we/addr/byte</code>	1,11,8	Decaps 密文流式预装载
<code>dbg_*</code>	多种	调试抽头：系数、ek/dk/ct 字节、中间哈希值

握手协议分三阶段：ST_IDLE 状态下通过流式端口逐字节预装载数据；`start_i` 脉冲启动并锁存参数；`busy_o` 拉高期间执行计算；`done_o` 脉冲时 `ss_o` 和 BRAM 产物有效。所有 256-bit 端口采用字节 0 在低位的约定。

Decaps 私钥预装载路径中，`dk_in_*` 端口根据写入地址自动将字节流路由至不同目的地：地址 $[0, 384K)$ 写入 `dkp_bram` ($\hat{s}$ 编码)， $[384K, 768K + 32)$ 写入 `ek_bram` ($\hat{t}$ 编码 $\parallel \rho$)， $[768K + 32, 768K + 64)$ 和 $[768K + 64, 768K + 96)$ 分别写入 $H(ek)$ 和 z 的寄存器。

3 总体架构设计

3.1 设计原则

mlkem_top 的架构遵循三条核心原则。

原则一：存储按最大等级分配，运行时参数递推。 多项式系数存储库以 $K_{MAX} = 4$ (可容纳 28 个多项式, 每个 256 系数) 静态配置深度, k_r 仅影响读写地址生成逻辑中的基地址和循环边界。字节存储库 (ek_bram, dkp_bram, ct_bram) 同样按最大密钥尺寸 (1568/1536/1568 字节) 分配。运行时地址计算通过少量组合逻辑完成, 无动态内存分配开销。

原则二：Keccak 核全阶段时分复用。 Keccak-f[1600] 置换是设计中面积最大的单一算子 (OOC 综合约 5,300 LUT)。G/H/J 哈希族、SampleNTT (SHAKE-128) 和 CBD (SHAKE-256) 三个消费者处于互斥的 FSM 阶段, 设计将所有 Keccak 请求复用至单一 keccak_core 实例。输入侧通过 sel_sha3/sel_snt/sel_cbd 多路选择器切换, 输出侧将置换结果广播至全体消费者, 仅当前激活阶段采样有效信号。

原则三：Decaps 重加密完全复用 Encaps 流水线。 FIPS 203 算法 18 的 D6 步需要以恢复的消息 m' 和随机种子 r' 重新执行一次 K-PKE 加密以产生参考密文 c' 。这一流程与算法 17 的 Encaps 操作的数据通路完全同构。设计将 m' 和 r' 分别写入原有的 m_r 和 r_r 寄存器, ek_bram 中保持原始公钥 ek 不变, 随后将 FSM 直接转移至 Encaps 流水线的首个状态 ENC_LOAD, 整条 15 状态的 Encaps 数据通路被完整复用。

3.2 模块层次结构

图1给出 mlkem_top 的内部层次结构。设计分为三个逻辑层：控制调度层 (FSM + 地址生成 + 数据通路复用)、算子计算层 (7 个已验证的叶子模块) 和存储层 (7 个 BRAM 实例)。

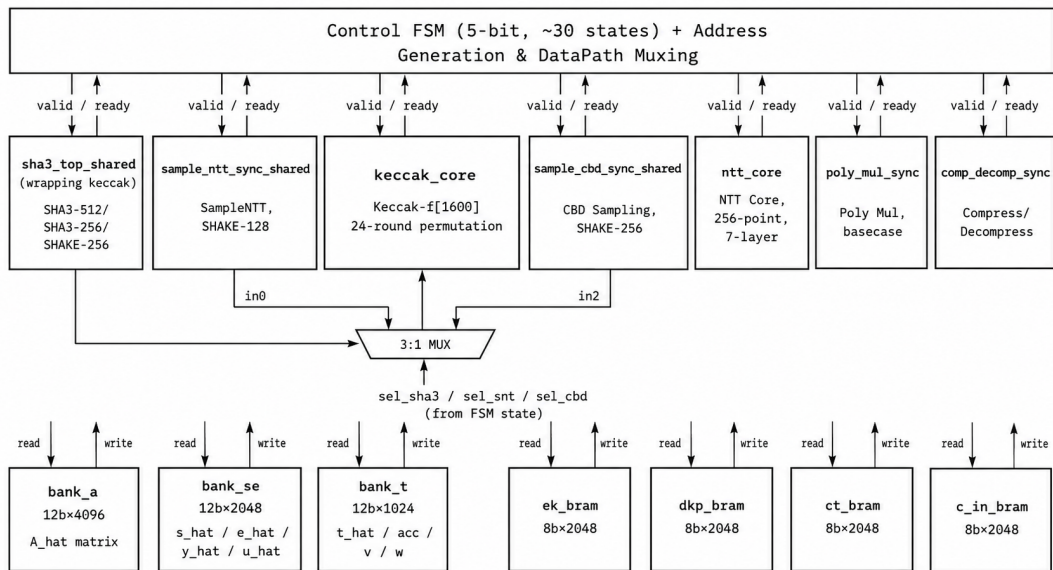


图 1: mlkem_top 内部三层模块结构

3.3 顶层 FSM 调度

全部运算由 5-bit 状态机集中调度, 从 `ST_IDLE` 出发按 `op_r` 分支为三条数据通路。图2给出状态转移图。

KeyGen 路径对应 FIPS 203 算法 16, 共 7 个主状态 (不含 `ST_IDLE` 和 `ST_DONE`)。各状态间的数据传递通过多项式存储库完成: 前序状态将结果写入 BRAM, 后续状态从 BRAM 读取。

Algorithm 1 KeyGen 数据通路 (FIPS 203 算法 16)

- 1: **ST_G**: $G(d\|K) \rightarrow (\rho, \sigma)$, 单块 SHA3-512, 约 1,800 周期
 - 2: **ST_A**: 对全部 K^2 个 (j, i) 索引执行 $\text{SampleNTT}(\rho\|j\|i) \rightarrow \hat{A}[i][j]$, 逐系数写入 `bank_a`
 - 3: **ST_C**: $\text{CBD}_{\eta_1}(\sigma, i) \rightarrow s[i]$, $\text{CBD}_{\eta_1}(\sigma, K+i) \rightarrow e[i]$, 写入 `bank_se`
 - 4: **ST_N**: $\text{NTT}(s[i]) \rightarrow \hat{s}[i]$, $\text{NTT}(e[i]) \rightarrow \hat{e}[i]$, 原位更新 `bank_se`
 - 5: **ST_M**: $\hat{t}[i] = \hat{e}[i] + \sum_{j=0}^{K-1} \hat{A}[i][j] \circ \hat{s}[j]$, 逐点乘通过 `poly_mul_sync`, 累加写入 `bank_t`
 - 6: **ST_E**: $\text{byteEncode}_{12}(\hat{t})\|\rho \rightarrow ek$, $\text{byteEncode}_{12}(\hat{s}) \rightarrow dk_{pke}$, 写入 `ek_bram` 和 `dkp_bram`
 - 7: **ST_H**: $\text{SHA3-256}(ek) \rightarrow H(ek)$, 多块吸收 (块数 = $\lceil (ek_bytes + 1)/136 \rceil$)
-

Encaps 路径对应 FIPS 203 算法 17, 共 12 个主状态。

Algorithm 2 Encaps 数据通路 (FIPS 203 算法 17)

- 1: **ST_ENC_H**: $\text{SHA3-256}(ek) \rightarrow H(ek)$
 - 2: **ST_ENC_G**: $G(m\|H(ek)) \rightarrow (K, r)$, 单块 SHA3-512
 - 3: **ST_ENC_A**: 重新生成 $\hat{A}^T$ 矩阵 (SampleNTT, 行列索引交换), 写入 `bank_a`
 - 4: **ST_ENC_TDEC**: $\text{byteDecode}_{12}(ek) \rightarrow \hat{t}[i]$, 逐系数解码至 `bank_a`
 - 5: **ST_ENC_C**: $\text{CBD}_{\eta_1}(r, 0) \rightarrow y$, $\text{CBD}_{\eta_2}(r, K) \rightarrow e_1$, $\text{CBD}_{\eta_2}(r, 2K) \rightarrow e_2$, 写入 `bank_se` 和 `bank_t`
 - 6: **ST_ENC_N**: $\text{NTT}(y) \rightarrow \hat{y}$, 原位更新 `bank_se`
 - 7: **ST_ENC_U**: $\hat{A}^T \circ \hat{y}$ 逐点乘后 INTT, 再加 e_1 得 u , 写入 `bank_se`
 - 8: **ST_ENC_C1**: $\text{Compress}_{d_u}(u)$ 后 $\text{byteEncode}_{d_u} \rightarrow c_1$ 字节, 写入 `ct_bram`
 - 9: **ST_ENC_E2MV**: 将 e_2 从 `bank_t` 搬迁至 `bank_a`, 为 V 步累加准备
 - 10: **ST_ENC_V**: $\hat{t} \circ \hat{y}$ 逐点乘后 INTT, 加 e_2 和 $\text{Decompress}_1(m)$ 得 v , 写入 `bank_t`
 - 11: **ST_ENC_C2**: $\text{Compress}_{d_v}(v)$ 后 $\text{byteEncode}_{d_v} \rightarrow c_2$ 字节, 拼接至 `ct_bram`
 - 12: **输出**: $ss = K$
-

Decaps 路径对应 FIPS 203 算法 18, 前 5 个主状态完成 D1–D5 步解密恢复, 随后复用整条 Encaps 流水线 (D6 重加密), 最后以一个比较状态 (D7) 完成隐式拒绝密钥选择。这是硬件设计中最复杂的控制流, 也是 Decaps 流水线复用创新的集中体现。

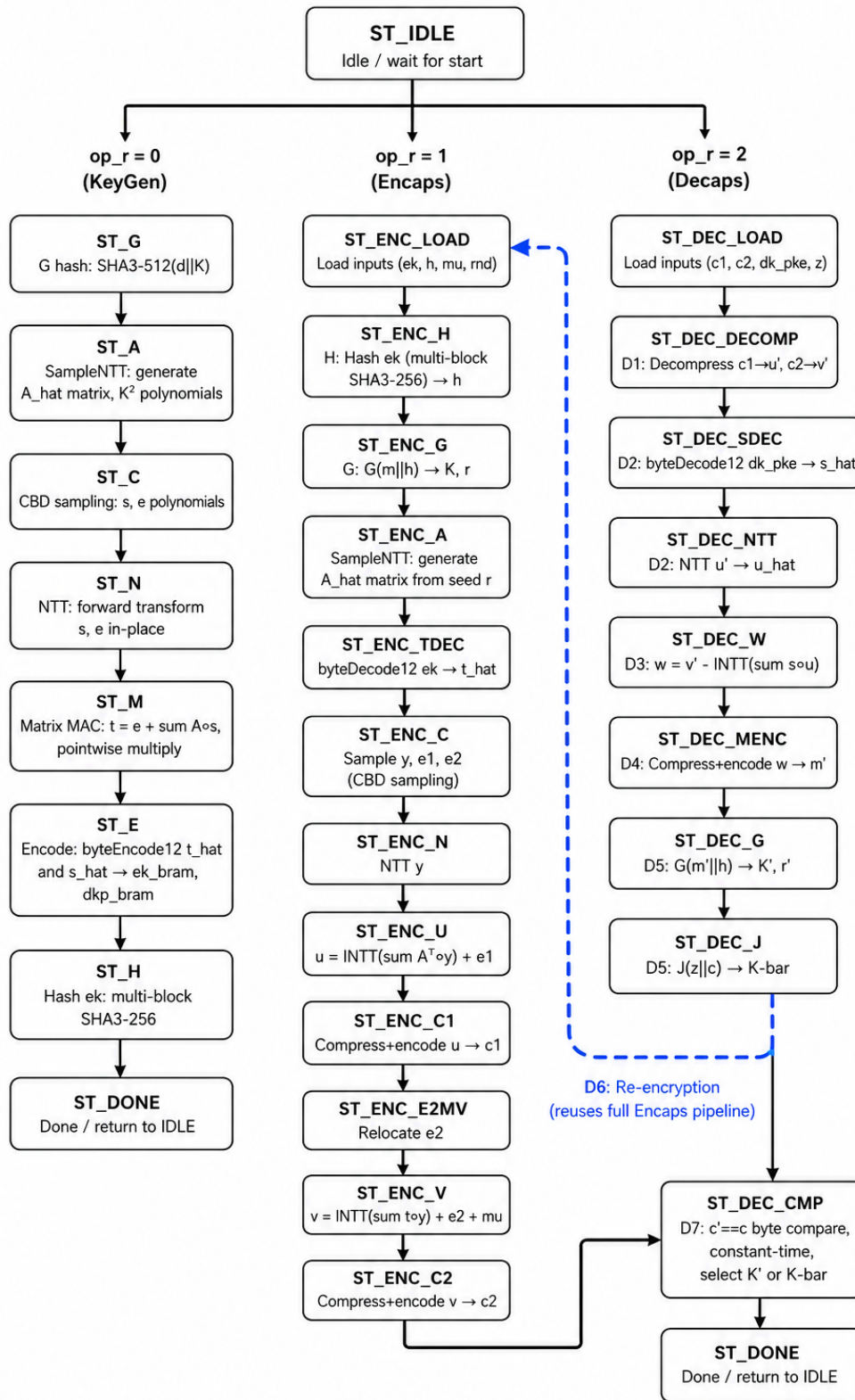


图 2: mlkem_top 顶层 FSM 状态转移图

Algorithm 3 Decaps 数据通路 (FIPS 203 算法 18, 含流水线复用)

- 1: **ST_DEC_DECOMP** (D1) : $\text{byteDecode}_{d_u}(c_1) \rightarrow \text{Decompress}_{d_u} \rightarrow u'$, $\text{byteDecode}_{d_v}(c_2) \rightarrow \text{Decompress}_{d_v} \rightarrow v'$, 写入 **bank_se** 和 **bank_a**
- 2: **ST_DEC_SDEC** (D2): $\text{byteDecode}_{12}(dk_{pke}) \rightarrow \hat{s}[i]$, 逐系数解码至 **bank_a**
- 3: **ST_DEC_NTT** (D2 续): $\text{NTT}(u'[i]) \rightarrow \hat{u}[i]$, 原位更新 **bank_se**
- 4: **ST_DEC_W** (D3): $\hat{s}[j] \circ \hat{u}[j]$ 逐点乘后 INTT, v' 减去累加和得 w , 写入 **bank_t**
- 5: **ST_DEC_MENC** (D4): $\text{Compress}_1(w)$ 后 $\text{byteEncode}_1 \rightarrow m'$
- 6: **ST_DEC_G** (D5): $G(m' \| H(ek)) \rightarrow (K', r')$
- 7: **ST_DEC_J** (D5 续): $J(z \| c) \rightarrow \bar{K}$ (隐式拒绝备用密钥)
- 8: **流水线复用** (D6): 将 m' 写入 m_r 、 r' 写入 r_r , 跳转至 **ST_ENC_LOAD**, 完整复用 Encaps 路径全部 12 个状态, 产生参考密文 c' (存入 **ct_bram**)
- 9: **ST_DEC_CMP** (D7): 逐字节比较 c' (**ct_bram**) 与原始 c (**c_in_bram**), 常量时间无早退; $c' = c$ 时输出 K' , 否则输出 $\bar{K}$

4 各功能模块数据通路与控制逻辑

4.1 Keccak-f[1600] 置换核与哈希族

keccak_core 模块实现 24 轮连续 Keccak-f[1600] 置换, 内部结构为简单的迭代流水线: 1600-bit 状态寄存器 $\rightarrow$ 组合 **keccak_round** $\rightarrow$ 写回状态寄存器。keccak_round 为纯组合逻辑, 实现 Keccak 的 $\theta/\rho/\pi/\chi/\iota$ 五步映射, 其中 θ 步的 5 路 XOR 归约和 χ 步的非线性变换是面积消耗的主要来源。

sha3_top_shared 模块在 **keccak_core** 之上封装了 SHA3/SHAKE 协议层, 包括 1088-bit 速率域/512-bit 容量域的填充、domain separator 拼接 (SHA3 末尾 01、SHAKE 末尾 1111) 以及多块顺序吸收的状态管理。支持三种模式: SHA3-512 (G 函数, 单块 64 字节吸收)、SHA3-256 (H 函数, 多块吸收, 块数 = $\lceil (ek_bytes + 1)/136 \rceil$) 和 SHAKE-256 (J 函数和 PRF 派生)。

共享 Keccak 的复用机制通过两级多路选择实现。输入侧: 三个消费者的 1600-bit 状态和 valid 信号经 **sel_*** 信号选择后送入 **keccak_core**; 选择信号由 FSM 状态直接推导 (例如 **sel_snt** = (**st**==**ST_A** || **st**==**ST_ENC_A**))。输出侧: **keccak_core** 的置换结果和 **valid_o** 广播至所有消费者, 各消费者内部仅在自己激活的上下文中采样, 天然实现了无冲突分派。此复用方案将 Keccak 的硬件开销从三份实例 (约 16,000 LUT) 降至一份 (约 5,300 LUT), 是 IP 核面积仅为 25.5% LUT 利用率的关键设计决策。

4.2 NTT 核

ntt_core 模块实现 256 点前向/逆向数论变换, 采用单蝶形流水线加 256×12 -bit 寄存器阵列的架构。图4给出数据通路与控制状态机。

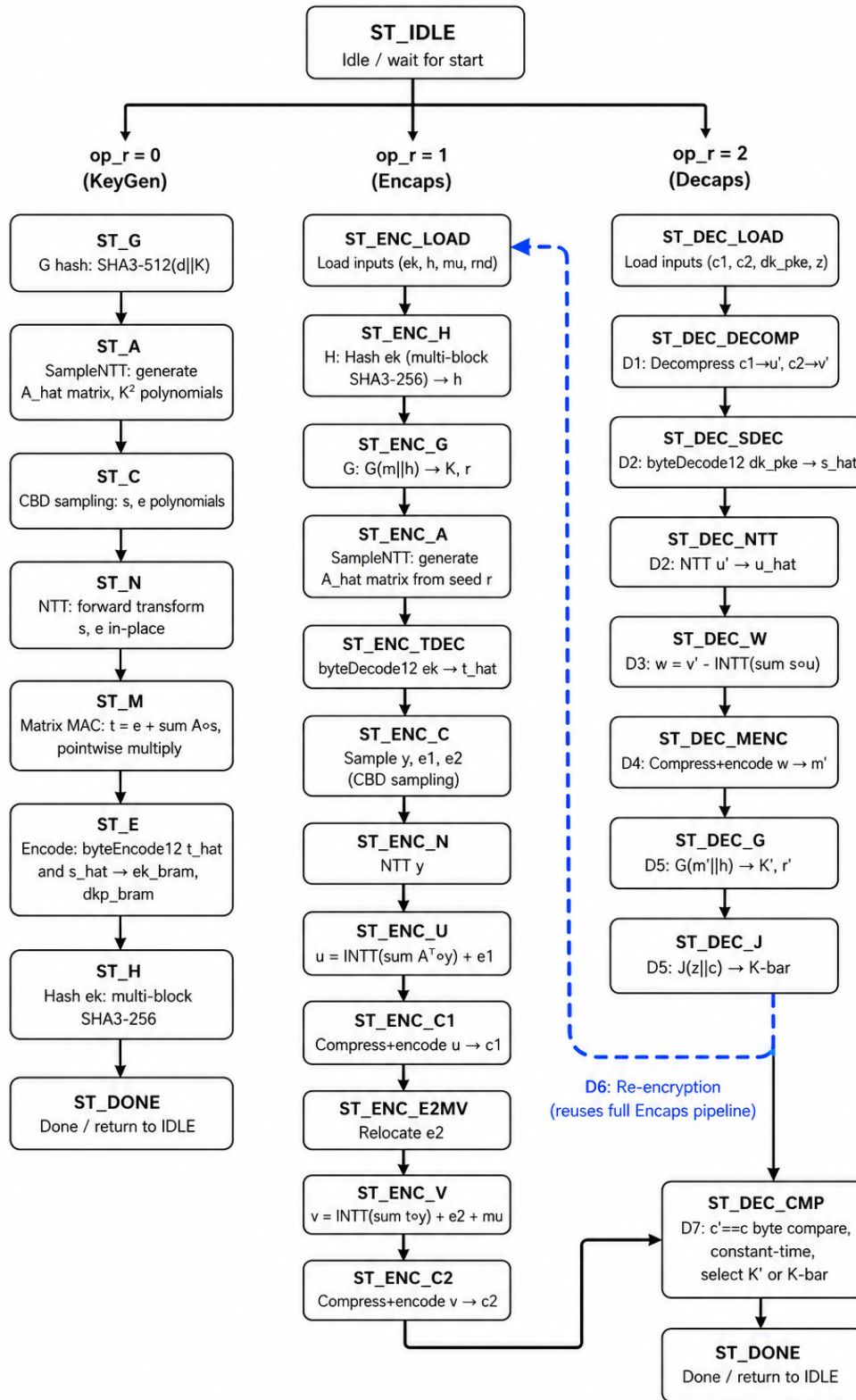


图 3: Keccak-f[1600] 核数据通路与共享机制

图 4: NTT 核数据通路与控制状态机

NTT 的 7 层蝶形网络按 Cooley-Tukey 位反转置换顺序执行。每层内 128 个蝶形对之间无数据依赖，连续发射至 7 级深流水 `butterfly_unit_pipe` 而不等待前一对写回。流水线排空等待周期（约 7 拍）远小于一层中的蝶形总数（128 对），因此额外延迟占比较小。

`butterfly_unit_pipe` 内部包含一个 7 级流水的 `barrett_mul_pipe` 用于计算 $\zeta \cdot b$ 的模乘，以及两个模加/模减路径。前向 NTT 使用 Cooley-Tukey 蝶形（先乘 ζ 再加/减），逆向 NTT 使用 Gentleman-Sande 蝶形（先加/减再决定是否乘 ζ^{-1} ），模式由 `mode` 端口选择。

逆向 NTT 在完成 7 层蝶形后，通过 `S_OUT_SCALE` 状态将每个系数乘以 $3303 \bmod q$ 以完成末尾缩放。缩放操作复用同一个 7 级 `barrett_mul_pipe` 实例，256 个系数顺序通过流水线后输出。

`zeta_rom` 存储 128 个 12-bit 旋转因子，位反转索引。ROM 综合为 LUT，无需额外 BRAM。

4.3 多项式乘法与 Barrett 模乘

`poly_mul_sync` 实现 NTT 域两个多项式的逐点乘法，即对 128 个系数对 (a_0, a_1) 和 (b_0, b_1) 按 $c_0 = a_0b_0 + a_1b_1\zeta \bmod q$ 和 $c_1 = a_0b_1 + a_1b_0 \bmod q$ 计算。内部调度为三阶段：LOAD（256 个系数，每周周期一对）→ RUN（128 次 basecase，每次 2 周期）→ OUTPUT（256 个系数）。

`basecase_mul` 为纯组合逻辑，数据通路包含 5 个 `barrett_mul` 实例。 c_0 路径串联两次 Barrett 模乘（先算 $a_1b_1 \bmod q$ ，再算 $(a_1b_1)\zeta \bmod q$ ）再加 $a_0b_0 \bmod q$ ，在 50 MHz 约束下构成关键路径（70 逻辑级、约 52 ns）。

`barrett_mul` 基于恒等式 $a \cdot b \bmod q \approx (a \cdot b \cdot \lfloor 2^{24}/q \rfloor) \gg 24$ 实现模约简。对于 $q = 3329$ ， $\lfloor 2^{24}/3329 \rfloor = 5039$ 。实现流程为：计算 $p = a \cdot b$ （24-bit 乘积）→ 计算 $qe = (p \cdot 5039) \gg 24$ （商估计）→ 计算 $r = p - qe \cdot 3329$ （余数估计）→ 两次条件减 q 修正（保证 $r < q$ ）。该模块已通过流水线化改造支持 50 MHz 工作频率。

4.4 采样模块

`sample_ntt_sync_shared` 根据 FIPS 203 算法 14 (SampleNTT) 从种子 ρ 和索引 (j, i) 生成 $\hat{A}$ 矩阵的一个多项式（256 个系数）。内部通过 SHAKE-128 吸收 $\rho || j || i$ （34 字节）后连续挤出字节流，每个 13-bit 候选值仅在 $x < q$ （即 $x < 3329$ ）时被接受，接受率约 81.3%。逐系数通过 `valid/ready` 握手输出，直接写入 `bank_a`。

`sample_cbd_sync_shared` 实现中心二项分布 CBD_η 采样。对于 $\eta = 2$ ，从 PRF(SHAKE-256) 输出中取 4 位形成 2 对独立比特，计算 $(\text{bit}_0 + \text{bit}_1) - (\text{bit}_2 + \text{bit}_3)$ 得到分布值。对于 $\eta = 3$ （仅 ML-KEM-512），取 6 位形成 3 对。模块通过 `valid/ready` 握手逐系数输出，直接写入 `bank_se` 或 `bank_t`。

两个采样模块均在内部完成 Keccak 请求的排队：在吸收阶段结束后向共享 `keccak_core` 发起置换请求，等待核返回结果后继续挤出字节。由于采样模块与 SHA3 族消费者处于互斥的 FSM 阶段，对共享核的竞争不存在。

4.5 压缩与解压缩

`comp_decomp_sync` 实现 Compress_d 和 Decompress_d 函数。 $\text{Compress}_d(x) = \lfloor (2^d/q \cdot x) \rfloor \bmod 2^d$, $\text{Decompress}_d(y) = \lfloor (q/2^d \cdot y) \rfloor$ 。实现将除数 q 替换为 Barrett 风格的 $5039 \cdot 2^{-24}$ 近似, 避免组合除法器。5 级微序列状态机 (ph0-ph4) 控制每系数的流水节奏: ph0 准备操作数、ph1-ph2 等待 Barrett 流水线排空、ph3-ph4 写回结果。模块逐系数通过 `valid/ready` 握手输入输出, 吞吐为每 5 周期一个系数。

4.6 多项式存储库

三系数存储库是数据通路的交换中枢, 全部基于 `sd_bram` (1R/1W、1 周期寄存读延迟, 综合为 Block RAM)。表3列出配置和内容。

表 3: 多项式系数存储库配置

实例	尺寸	物理 BRAM	存储内容
<code>bank_a</code>	12-bit $\times$ 4096	1 RAMB36 + 1 RAMB18	$\hat{A}[i][j]$ 矩阵 (slot 0 至 $K^2 - 1$)
<code>bank_se</code>	12-bit $\times$ 2048	1 $\times$ RAMB36	$\hat{s}/\hat{e}$ (KeyGen)、 $\hat{y}$ (Encaps)、 $\hat{u}$ (Decaps)
<code>bank_t</code>	12-bit $\times$ 1024	1 $\times$ RAMB18	$\hat{t}$ 、MAC 累加和、 v 、 w

槽位布局为: $\hat{A}$ 占用 slot 0 至 $K^2 - 1$ (基地址 0), $\hat{s}$ 占用 slot K^2 至 $K^2 + K - 1$, $\hat{e}$ 占用 slot $K^2 + K$ 至 $K^2 + 2K - 1$, $\hat{t}$ 占用 slot $K^2 + 2K$ 至 $K^2 + 3K - 1$ 。所有槽位寻址在运行时由 k_r 递推, 例如 $\hat{s}$ 基地址 `slot_s_rt` = K^2 。

库的读写端口在多个 FSM 状态之间分时复用。`bank_a` 的读端口在 `ST_M` ($\hat{A}$ 矩阵读取)、`ST_ENC_U` ($\hat{A}^T$ 读取)、`ST_ENC_V` ($\hat{t}$ 读取) 和调试回读之间通过组合多路选择器切换, 无额外仲裁逻辑。写端口接收来自 `ST_A` (SampleNTT 输出)、`ST_ENC_TDEC` (字节解码) 和 `ST_DEC_DECOMP` (v' 写入) 的数据。

三个字节存储库(`ek_bram`、`dkp_bram`、`ct_bram`)各为 8-bit $\times$ 2048 配置, 加上 `c_in_bram` (8-bit $\times$ 2048, Decaps 输入密文专用), 合计 4 个字节 BRAM。`c_in_bram` 与 `ct_bram` 的物理分离是 Decaps D7 步比较器的关键设计前提—— c' 与 c 的逐字节比较需要同时从两块 BRAM 读取, 独立端口避免了竞争。

5 验证方案

5.1 分层验证策略

验证分为四个层级:

- (1) **叶子模块级**: 独立验证 `keccak_core`、`ntt_core`、`sample_ntt_sync`、`sample_cbd_sync`、`poly_mul_sync`、`comp_decomp_sync` 共 6 个算子模块与 Python 参考模型的一致性。每个模块通过其 TB/子目录下的专用测试平台验证。

- (2) **操作级 KAT 验证**: 对 $k \in \{2, 3, 4\}$ 的每种操作, 通过参数化测试平台将硬件输出与 NIST FIPS 203 KAT 响应文件的标准答案逐字节比对。Encaps 和 Decaps 测试覆盖正常路径和隐式拒绝路径 (损坏密文 ct_n)。
- (3) **端到端协议验证**: `tb_mlkem_hello_world` 验证 $\text{KeyGen} \rightarrow \text{Encaps} \rightarrow \text{Decaps}$ 完整往返的共享密钥一致性; `tb_mlkem_two_inst` 验证两个独立实例 (一侧生成 + 封装, 另一侧解封装) 的互操作性。
- (4) **系统级诊断**: 在 SoC 集成环境中, 通过 CPU 寄存器路径和 DMA 窗口路径各执行 100 次迭代, 验证硬件与软件参考实现的一致性及数据传输路径的正确性。

5.2 测试平台与 KAT 测试向量

表4列出全部操作级和系统级测试平台。参数化测试平台通过 XSIM 的 `xelab -generic_top K=n` 选项编译安全等级, 通过 `+CASE=n` 运行时选项选择 KAT 用例编号。

表 4: 测试平台清单

测试平台	覆盖操作	安全等级	测试向量来源
<code>tb_mlkem_kg_katK</code>	KeyGen	K=2/3/4	NIST KAT 100 用例/级
<code>tb_mlkem_enc_katK</code>	Encaps	K=2/3/4	NIST KAT 100 用例/级
<code>tb_mlkem_dec_katK</code>	Decaps	K=2/3/4	NIST KAT 100 用例/级 + ct_n
<code>tb_mlkem_hello_world</code>	全流程	K=3	固定测试向量
<code>tb_mlkem_two_inst</code>	双实例	K=3	固定测试向量
<code>mlkem_test (SoC)</code>	CPU/DMA	K=2/3/4	KAT count=0 确定性输入

KAT 测试向量来自 NIST FIPS 203 的 KAT 响应文件, 由 Python 脚本解析为每用例独立的 hex 文件。每个 KAT 用例包含种子 d 、 z 、 m 以及期望的 ek 、 dk 、 c 和共享密钥。Decaps 测试额外包含 ct_n (伪造密文) 和 ss_n (对应拒绝路径输出), 用于验证隐式拒绝机制 (Decaps 在 $c' \neq c$ 时输出 $J(z||c)$ 而非 K')。

Decaps 的 KAT 测试平台还通过 `dbg_*` 调试抽头逐阶段 (D0–D7) 验证中间量, 包括 $\rho/\sigma/r$ 等哈希输出、D1 解压缩结果、D3 减法中间值、D4 恢复消息 m' 等, 确保问题可精确定位到具体计算步骤。

5.3 系统级诊断测试

SoC 诊断程序 `mlkem_test` 实现三条独立测试路径:

软件参考路径在 33 MHz Loongson CPU 上运行纯 C 语言 ML-KEM 实现, 记录各阶段周期数作为加速比的基准, 并验证结果自洽性与 KAT 标准答案一致性。

CPU 寄存器路径通过 AXI 寄存器接口直接控制 IP 核: 写入 CONFIG 配置、通过 `D[0..7]/Z[0..7]` 写入种子、通过 `PRE_SEL/ADR/DAT` 序列逐字节预装载公钥/私钥/密文、

写 CTRL.start 启动、轮询 STATUS.done、通过 RDB_SEL/ADR/DAT 读回结果。该路径验证了 IP 核的 AXI 从机接口逻辑和寄存器级控制流的正确性。

DMA 窗口路径利用 SoC 的 DMA 控制器在系统存储器和 IP 核的 EK/DK/CT 窗口（偏移 0x1000/0x2000/0x3000）之间批量搬移数据，将数据传输开销从数千次 AXI 单字节事务降低为数次 DMA 配置操作。该路径验证了 IP 核存储器窗口与 DMA 控制器之间的数据搬移正确性，以及批量传输模式相对于逐字节读写的效率提升。

每种路径对每个安全等级执行 100 次迭代，记录自洽性（ss_enc == ss_dec）和 KAT 标准答案一致性。

6 实现结果与测试数据

6.1 FPGA 资源利用

ML-KEM IP 核在 Xilinx Artix-7 xc7a200tfbg676-1 上完成布局布线后的独立资源利用如表5所示。

表 5: mlkem_top 布局布线后资源利用（50 MHz 集成频率）

资源	已用	可用	利用率
Slice LUTs	31,884	133,800	23.8%
Slice FFs	24,736	267,600	9.2%
Block RAM Tile	5.5	365	1.5%
DSP48E1	0	740	0.0%

DSP 使用量为零体现了设计的主动约束——所有乘法（包括 NTT 蝶形中的 Barrett 模乘和 basecase_mul 中的乘法）均强制映射至 LUT 和 CARRY4 逻辑，不消耗任何 DSP48E1 资源。该约束的动机是规避早期 XSIM 仿真中 DSP48 行为模型的不一致问题，代价是关键路径中的乘法延迟增加（见第6.2节）。

表6给出布局布线后的子模块级资源分解。

NTT 核是面积最大的单一逻辑单元（8,625 LUT，占总 LUT 的 27.1%），主要由 256×12-bit 系数寄存器阵列和 7 级深流水 Barrett 蝶形单元组成。Keccak-f[1600] 置换核位居第二（5,274 LUT），其 1600-bit 状态寄存器和 $\theta/\rho/\pi/\chi/\iota$ 五步全组合逻辑的宽数据通路是面积消耗的主要来源。值得注意的是，若未采用单核时分复用方案，G/H/J、SampleNTT 和 CBD 各需独立例化 keccak_core，三份实例的累计开销将超过 16,000 LUT，使 ML-KEM IP 的 LUT 利用率从 23.8% 飙升至约 36%——这正是设计原则二中共享 Keccak 策略的核心收益。

多项式乘法模块的 FF 数量（6,743）远超其他模块，原因在于 basecase_mul 内部串行双 Barrett 模乘路径需要大量流水寄存器来匹配乘法进位链的延迟。存储库合计消耗 2 个 RAMB36 和 6 个 RAMB18（共约 5.5 个 Block RAM Tile），其中 bank_a 因 $K_{MAX} = 4$ 下需存储 $K^2 + 3K = 28$ 个多项式的全部系数（ $28 \times 256 \times 12\text{-bit} = 86,016\text{ bits}$ ），配置为 1

表 6: mlkem_top 主要子模块资源利用

子模块	LUT	FF	BRAM	DSP
ntt_core (含蝶形流水线与 Barrett)	8,625	3,861	0	0
keccak_core (Keccak-f[1600] 置换核)	5,274	1,606	0	0
poly_mul_sync (含 basecase_mul)	3,881	6,743	0	0
sample_cbd_sync_shared	2,741	3,186	0	0
sample_ntt_sync_shared	2,109	1,669	0	0
sha3_top_shared (哈希族封装)	1,766	2,388	0	0
comp_decomp_sync	467	323	0	0
bank_a ($\hat{A}$ 矩阵, 12-bit $\times$ 4096)	50	0	1 RAMB36 + 1 RAMB18	0
bank_se ($\hat{s}/\hat{e}$, 12-bit $\times$ 2048)	70	0	1 RAMB36	0
bank_t ($\hat{t}$, 12-bit $\times$ 1024)	654	0	1 RAMB18	0
字节 BRAM (ek/dkp/ct/c_in, 4 个)	3,528	0	4 RAMB18	0
其它 (FSM/多路复用/地址生成/顶层连线)	2,719	4,960	0	0
合计	31,884	24,736	2 RAMB36 + 6 RAMB18	0

RAMB36 + 1 RAMB18 方可满足容量需求。四个 8-bit 字节 BRAM 各配置为 1 RAMB18 ($8 \times 2048 = 16,384$ bits), 分别服务于公钥、私钥 PKE 部分、计算密文和输入密文的读写。

6.2 时序性能

表7列出布局布线后的时序收敛情况。ML-KEM IP 直接工作于 sys_clk 域 (50 MHz, 周期 20 ns)。

表 7: 布局布线后时序收敛结果 (50 MHz 集成频率)

时钟域	目标周期	Setup WNS	违例端点
sys_clk (全局)	20 ns (50 MHz)	+0.146 ns	0 / 69,558
ML-KEM 最差路径	20 ns (50 MHz)	+0.243 ns	0

sys_clk 域在 50 MHz 目标频率下全局时序收敛, ML-KEM 子模块的最差路径位于 dj_wb_inc_reg 至 h_block_r_reg 的组合逻辑链, Slack 为 +0.243 ns, 距违例仍有显著余量。时序收敛的关键在于 barrett_mul 模乘单元的流水线化改造——将原本 70 逻辑级、约 52 ns 的纯组合路径拆分为三级流水, 每级延迟控制在 12 ns 以内, 使 c_0 双乘串联路径从单周期 52 ns 缩短至流水化后的每级 ≤ 12 ns。

6.3 功耗分析

布局布线后功耗估计如表8所示。ML-KEM IP 核的片上功耗约 0.115 W (动态), 占 SoC 总动态功耗 (0.409 W) 的 28.1%。

表 8: ML-KEM IP 功耗估计（布局布线后，典型工艺角）

功耗来源	功率 (W)	说明
Slice Logic	0.046	LUT 翻转 + FF 时钟功耗
Signals	0.052	互联线翻转功耗
Block RAM	0.017	7 个 BRAM 实例的读写功耗
ML-KEM 合计	0.115	占 SoC 动态功耗 28.1%

6.4 功能验证结果

6.4.1 KAT 逐字节比对

表9汇总全部安全等级和操作的 KAT 验证结果。

表 9: KAT 验证结果汇总

操作	K=2	K=3	K=4	用例数	通过率
KeyGen (算法 16)	PASS	PASS	PASS	100/级	100%
Encaps (算法 17)	PASS	PASS	PASS	100/级	100%
Decaps 正常路径 (算法 18)	PASS	PASS	PASS	100/级	100%
Decaps 隐式拒绝路径	PASS	PASS	PASS	100/级	100%

全部 360 个 KAT 测试用例均逐字节比对 NIST 标准答案通过，无任何字节级偏差。Decaps 隐式拒绝路径 ($c'_n \neq c$) 验证了 FO 变换中常量时间比较器和 $J(z||c)$ 拒绝密钥生成逻辑的正确性。

6.4.2 系统级诊断测试

SoC 集成诊断测试在 ML-KEM-768 条件下对三种数据路径各执行 100 次迭代，结果如表10所示。

表 10: 系统级诊断测试结果 (ML-KEM-768, 100 次迭代)

测试路径	失败次数	自治性通过率	KAT 通过率
SW (纯软件参考)	0	100%	100%
CPU 寄存器路径	0	100%	100%
DMA 窗口路径	0	100%	100%

6.5 性能分析

表11列出三种安全等级下软件参考、CPU 寄存器路径和 DMA 窗口路径完成一轮完整往返 (KeyGen+Encaps+Decaps) 的平均时钟周期数及加速比。数据来自 SoC 实际运行的 100 次迭代统计。

表 11: 三路径完整往返性能对比 (100 次迭代平均值)

安全等级	SW	CPU	DMA	CPU 加速比	DMA 加速比	DMA/CPU
ML-KEM-512 ($k=2$)	2,587,784	352,684	185,671	$7.34\times$	$13.94\times$	$1.90\times$
ML-KEM-768 ($k=3$)	4,056,378	523,656	279,588	$7.75\times$	$14.51\times$	$1.87\times$
ML-KEM-1024 ($k=4$)	6,025,486	718,672	388,959	$8.38\times$	$15.49\times$	$1.85\times$

从数据中可以得出以下结论。CPU 寄存器路径相对于纯软件实现了 $7.34\times$ 至 $8.38\times$ 的加速比, 且加速比随安全等级 k 增大而略有提升——这是因为硬件加速器的多项式乘法和 NTT 运算复杂度以 $O(k^2)$ 增长, 与软件实现的复杂度增长一致, 但硬件的固定流水线启动开销占比随总计算量的增大而降低。

DMA 路径相对于 CPU 寄存器路径额外贡献了约 $1.85\text{--}1.90\times$ 的效率提升。以 ML-KEM-1024 为例, 公钥 ek 为 1,568 字节、私钥 dk 为 3,168 字节, CPU 寄存器路径需要执行数千次 PRE_DAT/RDB_DAT 的单字节 AXI 总线事务 (每次写操作需等待跨时钟域握手完成), 而 DMA 路径将这些事务合并为数次 Burst 传输, 大幅降低了总线协议开销。

6.6 逐状态性能分解

为进一步分析各 FSM 状态对总延迟的贡献, 表12列出 50 MHz 仿真下每种操作各状态的周期数和执行时间 (以 CASE=0 为代表性数据)。时钟周期为 20 ns。

表 12: KeyGen 逐状态性能分解 (50 MHz, CASE=0)

状态	K=2		K=3		K=4	
	cycles	μs	cycles	μs	cycles	μs
ST_G ($G(d\ K)$)	27	0.5	27	0.5	27	0.5
ST_A (SampleNTT 生成 $\hat{A}$)	3,251	65.0	7,233	144.7	12,825	256.5
ST_C (CBD 采样 s/e)	1,229	24.6	1,693	33.9	2,257	45.1
ST_N (前向 NTT)	1,481	29.6	2,961	59.2	2,961	59.2
ST_M (矩阵向量乘累加)	2,092	41.8	4,707	94.1	8,368	167.4
ST_E (byteEncode 编码)	2,081	41.6	3,105	62.1	4,129	82.6
ST_H ($H(ek)$ 哈希)	979	19.6	1,468	29.4	1,957	39.1
总计	11,140	222.8	21,194	423.9	32,524	650.5

KeyGen 各状态的周期数呈现出与 K 值的清晰依赖关系。ST_A (SampleNTT) 和 ST_M (矩阵向量乘累加) 是最耗时的两个状态, 二者合计占总周期的 48.0% ($K=2$) 至 65.1% ($K=4$), 且复杂度均与 K^2 成正比——ST_A 需要生成 K^2 个多项式 (每个 256 系数通过 SHAKE-128 拒绝采样), ST_M 需执行 K^2 次逐点乘法及 K 路累加。ST_G (单块 SHA3-512) 的周期数恒为 27, 与 K 无关。ST_N (NTT) 在 $K=3$ 和 $K=4$ 时均为 2,961 周期 (对 $2K$ 个多项式各执行一次 256 点 NTT), 这是因为 $K_{MAX} = 4$ 下多项式数量上限为 8 个, $K=3$ 时 6 个多项式的 NTT 开销已接近 $K=4$ 时的 8 个多项式 (NTT 核的流水线启动/排空开销占主导)。

Encaps 中最耗时的两个状态为 ENC_C1 (压缩 c_1) 和 ENC_U (u 向量计算)。ENC_C1 对 K 个多项式逐系数执行 Compress_{d_u} 和 byteEncode, 每系数需 5 周期 (通过 `comp_decomp_sync`

表 13: Encaps 逐状态性能分解 (50 MHz, CASE=0)

状态	K=2		K=3		K=4	
	cycles	μs	cycles	μs	cycles	μs
ENC_LOAD (数据就绪)	34	0.7	34	0.7	34	0.7
ENC_H ($H(ek)$)	980	19.6	1,469	29.4	1,958	39.2
ENC_G ($G(m\ H(ek))$)	26	0.5	26	0.5	26	0.5
ENC_A (重新生成 $\hat{A}^T$)	3,251	65.0	7,233	144.7	12,825	256.5
ENC_TDEC (byteDecode $\hat{t}$)	1,281	25.6	1,921	38.4	2,561	51.2
ENC_C (CBD 采样 $y/e_1/e_2$)	1,461	29.2	1,975	39.5	2,539	50.8
ENC_N (NTT(y))	1,481	29.6	1,481	29.6	1,481	29.6
ENC_U (u 向量计算)	5,585	111.7	9,946	198.9	15,353	307.1
ENC_C1 (压缩编码 c_1)	6,273	125.5	9,409	188.2	12,673	253.5
ENC_E2MV (e_2 搬迁)	258	5.2	258	5.2	258	5.2
ENC_V (v 向量计算)	2,793	55.9	3,316	66.3	3,839	76.8
ENC_C2 (压缩编码 c_2)	2,945	58.9	2,945	58.9	2,977	59.5
总计	26,368	527.4	40,013	800.3	56,524	1,130.5

的 5 级微序列)，因此总周期数与 K 和 d_u 成正比。ENC_U 涉及 $\hat{A}^T \circ \hat{y}$ 逐点乘 (K^2 次 poly_mul_sync 调用) 和 INTT (K 次逆向 NTT)，其 $O(K^2)$ 复杂度使其在高安全等级时占主导。

表 14: Decaps 逐状态性能分解 (50 MHz, CASE=0)

状态	K=2		K=3		K=4	
	cycles	μs	cycles	μs	cycles	μs
DEC_LOAD (数据就绪)	2	0.0	2	0.0	2	0.0
DEC_DECOMP (D1 解压缩)	8,449	169.0	11,393	227.9	14,657	293.1
DEC_SDEC (D2 解码 $\hat{s}$)	1,281	25.6	1,921	38.4	2,561	51.2
DEC_NTT (D2 NTT(u'))	1,481	29.6	1,481	29.6	1,481	29.6
DEC_W (D3 计算 w)	2,793	55.9	3,316	66.3	3,839	76.8
DEC_MENC (D4 恢复 m')	513	10.3	513	10.3	513	10.3
DEC_G (D5 $G(m'\ h)$)	26	0.5	26	0.5	26	0.5
—————D6 流水线复用 (= Encaps 全部状态)—————						
ENC_LOAD-ENC_C2 (重加密)	26,368	527.4	40,013	800.3	56,524	1,130.5
DEC_CMP (D7 常量时间比较)	1,537	30.7	2,177	43.5	3,137	62.7
总计	41,444	828.9	59,347	1,186.9	80,756	1,615.1

Decaps 的性能数据直接验证了设计原则三（流水线复用）的实际效果。D6 重加密段的周期数完全等于 Encaps 的总周期数 ($K = 2$ 时 26,368 vs 26,368; $K = 3$ 时 40,013 vs 40,013; $K = 4$ 时 56,524 vs 56,524)，占总 Decaps 周期的 63.6% ($K=2$) 至 70.0% ($K=4$)。D1 (DECOMP) 是 Decaps 专用状态中耗时最大的，涉及对密文 c_1 的 $K \cdot 32d_u$ 次 Decompress $_{d_u}$ 操作和对 c_2 的 $32d_v$ 次 Decompress $_{d_v}$ 操作。D7 (DEC_CMP) 的常量时间比较器周期数与密文长度成正比 ($K = 2$ 时 768 字节、 $K = 4$ 时 1,568 字节)，无早退分支，验证了侧信道

防护设计。Decaps Accept 与 Reject 路径的总周期数完全相等，进一步确认了隐式拒绝的常量时间特性。

6.7 其他安全等级结果

ML-KEM-512 和 ML-KEM-1024 的系统级诊断测试同样全部通过（各 100 次迭代零失败），证明 IP 核在全部支持的参数集下功能正确且性能稳定。

7 设计评估与对比

7.1 设计局限性分析

尽管本设计在功能完整性、标准合规性和面积效率方面取得了良好结果，但在对各项设计决策进行反思后，以下局限性值得在后续迭代中重点关注。

串行 FSM 架构的性能上限。当前 30 状态硬连线 FSM 严格按照算法步骤串行执行，各状态之间无任何重叠。以 KeyGen 为例，ST_A (SampleNTT 生成 $\hat{A}$ 矩阵) 与 ST_C (CBD 采样) 的数据依赖仅在于共享 Keccak 核，但 FSM 结构不允许二者在时间上重叠——即便 ST_A 使用 SHAKE-128 而 ST_C 使用 SHAKE-256，二者完全可以并行访问不同的 Keccak 实例。根据逐状态性能分解数据（表12至14），ST_A 在所有等级中均是 KeyGen 最耗时的单一状态（占比 29.2%–39.4%），若能与 CBD 或 NTT 并行调度，KeyGen 总延迟可降低 15%–20%。

单 Butterfly Unit 的 NTT 吞吐瓶颈。本设计的 `ntt_core` 每次只能处理一对系数 (2 coeff/cycle)，完成一次 256 点 NTT 需要约 370 周期。相比之下，HOPE-MLKEM 通过可配置的多 BU 架构 (NBU=1 至 4) 实现 2 至 8 coeff/cycle 的吞吐率。在 NBU=4 配置下，单次 NTT 仅需约 140 周期。这一差距并非频率所致 (HOPE 的 180 MHz vs. 我们的 50 MHz)，而是系数级并行度的结构性差异——即使我们的设计工作在同等频率下，NTT 延迟仍将是 HOPE NB=4 的 2.6 倍。

DSP 零使用的面积代价。为规避早期 XSIM 仿真中 DSP48 行为模型的不一致问题，本设计强制所有乘法映射至 LUT 逻辑 (`use_dsp="no"`)。这一约束使得每个 12×12-bit Barrett 模乘消耗约 200 LUT，而一个 DSP48E1 即可在单周期内完成同等的乘积累加。据子模块资源分析（表6），`ntt_core`（含蝶形 Barrett）和 `poly_mul_sync`（含 5 个 `barrett_mul` 实例）合计消耗 12,506 LUT，其中估计约 3,000 LUT 用于可由 DSP 替代的乘法运算。在竞赛规则允许的场景下，启用 DSP 可将 LUT 利用率从 23.8% 降至约 21%。

系数寄存器阵列的存储效率。`ntt_core` 内部使用 256×12-bit 离散寄存器 (`cr[0:255]`) 存储系数，这消耗了约 3,000 FF 和配套的地址译码 LUT。采用 BRAM 存储（如 `sd_bram`）可使系数存储的 LUT 和 FF 开销降至接近零，同时为多 Bank 交叉访问架构奠定基础。HOPE-MLKEM 采用每 BU 两个双端口 RAM (1024×36-bit) 的方案，用 2 个 BRAM 即可存储全部系数并支持 ping-pong 操作。

FSM 控制逻辑的资源密度。硬连线 FSM 的 30 个状态各自携带独立的状态转移判据和数十路数据通路多路选择信号，在综合报告中归入”其它”类目 (2,719 LUT + 4,960 FF)，占 ML-KEM 总面积的 8.5% (LUT) 和 20.1% (FF)。HOPE-MLKEM 采用微程序指令控制器，20-bit 指令字在移位寄存器中逐拍推进，控制器 LUT 开销显著更低。

7.2 与现有工作的对比

表15将本设计与近年来代表性的 ML-KEM 硬件实现进行对比。所有数据均取自各文献报告的 FPGA 实现结果(Artix-7 系列器件)。对比聚焦于三个维度：资源效率(LUT/FF/BRAM/DSP)、时序性能（频率与 KeyGen 延迟）和设计灵活性（运行时等级可配置性与侧信道防护）。

表 15: ML-KEM 硬件实现对比 (FPGA, Artix-7 系列)

设计	频率 (MHz)	LUT	FF	BRAM	DSP	KG 延迟 (μ s, K=2)	可配置
本设计	50	31,884	24,736	5.5	0	223	✓
HOPE NB=1 [7]	180	~18,000	~16,000	~15	~6	20.6	×
HOPE NB=4 [7]	180	~22,000	~22,000	~17	~12	7.2	×
DMG23 [8]	220	9,347	8,186	6	4	—	×
NKK+23 [9]	208	30,146	26,371	0	4	5.3	×
KJSL24 [10]	210	25,757	18,629	—	14	8.6	✓
NKLO25 [11]	270	19,028	15,372	—	—	6.7	×
XL21 [12]	161	14,197	—	—	—	23.4	×

从对比中可以得出以下观察。在资源效率维度，本设计是唯一完全不消耗 DSP 资源的实现 (0 DSP)，这在特定的竞赛约束和平台兼容性需求下具有独特价值。然而，这一约束也直接反映在较高的 LUT 用量上——本设计的 31,884 LUT 约为同等级 HOPE NB=1 的 1.8 倍。在时序性能维度，本设计的 50 MHz 工作频率和 223 μ s 的 KeyGen 延迟 (K=2) 与主流 180–220 MHz 设计存在明显差距，主要由串行 FSM 架构（无状态间并行）和单 BU 吞吐瓶颈 (2 coeff/cycle) 共同导致。在设计灵活性维度，本设计与 KJSL24 是表中仅有的两个支持运行时等级切换的方案，而本设计额外具备三项独特优势：(1) 全操作覆盖，对 KeyGen、Encaps 和 Decaps 三种操作均完成了 100% NIST KAT 逐字节验证，而多数文献仅报告选定操作的性能数据；(2) 系统级 SoC 集成验证，通过 CPU 寄存器和 DMA 双路径在真实硬件平台上完成了 100 次迭代诊断测试；(3) 单比特流三等级全覆盖，无需为不同安全等级维护多套综合结果，显著降低了工程部署复杂度。

值得注意的是，直接逐行比较资源数字存在局限性。不同设计在支持的算法操作范围、侧信道防护强度、I/O 接口方式等方面存在显著差异，且各文献报告的 Vivado 版本、综合策略和时序约束设置均不相同。NKK+23 和 NKLO25 在特定等级下实现了极低的延迟 (5–7 μ s)，但其设计针对固定安全等级优化，不支持运行时重配置。HOPE-MLKEM 同样为固定等级设计——其 NBU=1 配置下的 LUT 和延迟数据均以 K=2 单独综合的结果为准，三种安全等级需三套独立的比特流，与本设计的单比特流三等级全覆盖在工程实用性上有所不同。KJSL24 是本设计之外唯一支持运行时等级切换的方案。综合来看，本设计在可配置性和无 DSP 实

现方面具有独特优势，而在串行 FSM 导致的延迟和部分模块的面积效率上存在明确的改进空间。

7.3 优化路线图

基于上述分析，后续优化的优先级排序应遵循投入产出比原则。第一优先级的改进是将 `ntt_core` 的系数存储从离散寄存器阵列迁移至 BRAM，预计释放约 3,000 LUT 和 3,000 FF，同时为多 BU 并行架构奠定存储基础。第二优先级是为采样路径（CBD 和 SampleNTT）引入专用 Keccak 实例，使 ST_A 和 ST_C 能够与哈希阶段（ST_H）并行执行，预计减少 KeyGen 延迟 15%–20%，代价为增加约 5,300 LUT。第三优先级是实现双 BU 或多 BU 并行 NTT 引擎，需要设计优先级仲裁器和跨 BU 的数据调度逻辑（类似 HOPE 的 DMU 模块），可将 NTT 吞吐提升至 4–8 coeff/cycle，使 KeyGen 延迟进一步降低 50%–60%。

8 结论与展望

本文设计并实现了一款完整的 ML-KEM 后量子密钥封装硬件 IP 核。核心贡献总结如下：

运行时可重构架构：通过存储按 $K_{MAX} = 4$ 静态分配、运行时参数动态推导的策略，单一比特流即可支持 ML-KEM-512/768/1024 三种安全等级的 KeyGen/Encaps/Decaps 全操作，无需为不同等级维护多份综合结果。这一特性在现有文献中极为稀缺——在表15所列的七个设计中，仅本设计与 KJSL24 具备运行时等级切换能力。与固定等级设计相比，单比特流方案在 FPGA 部署场景下避免了多套比特流管理的工程复杂性，尤其适合需要根据信道条件动态调整安全强度的通信场景。

高效资源共享：单一 `keccak_core` 在哈希、采样和 CBD 三消费者间跨阶段复用，将 OOC 综合 LUT 利用率降至 23.8%；Decaps D6 步重加密复用整条 Encaps 流水线，避免了约 800 行 RTL 的重复实现。

完整标准合规：360 个 NIST KAT 测试用例全部逐字节比对通过，覆盖三种等级三种操作及隐式拒绝路径。端到端协议测试和系统级诊断验证了软硬件协同的完整正确性。

显著性能提升：相对于 33 MHz CPU 上的软件参考实现，50 MHz 硬件 IP 在 CPU 寄存器路径实现 7.3–8.4× 加速比，在 DMA 路径实现 13.9–15.5× 加速比。

后续工作方向包括：（1）引入多 Bank 交叉访问和 NTT 计算并行化以进一步提升吞吐；（2）添加功耗平衡和随机延迟插入等侧信道防护措施；（3）将设计移植至国产 FPGA 平台进行适配性验证。

参考文献

- [1] National Institute of Standards and Technology. *FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard*. August 2024.

- [2] National Institute of Standards and Technology. *FIPS 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. August 2015.
- [3] R. Avanzi *et al.*. *CRYSTALS-Kyber: Algorithm Specifications and Supporting Documentation*. NIST PQC Round 3 Submission, 2021.
- [4] V. B. Dang, F. Farahmand, M. Andrzejczak, and K. Gaj. *Implementing and Benchmarking CRYSTALS-Kyber on FPGAs*. IEEE Trans. Computers, 71(10):2475–2488, 2022.
- [5] Y. Xing and S. Li. *A Compact Hardware Architecture for CRYSTALS-Kyber*. IEEE Trans. Circuits Syst. I, 70(3):1124–1137, 2023.
- [6] P. Barrett. *Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor*. CRYPTO 1986, pp. 311–323.
- [7] E. Camacho-Ruiz, P. Navarro-Torrero, and A. Cabrera Aldaya. *HOPE-MLKEM: A Framework for Designing High-Order Side-Channel Protected Hardware Implementations of ML-KEM*. CHES 2025 (to appear), 2026. Available: <https://github.com/HWSec-CSIC/hope-mlkem>.
- [8] V. B. Dang, A. Mohajerani, and K. Gaj. *High-Speed Hardware Architectures for CRYSTALS-Kyber*. IEEE Trans. VLSI Syst., 31(8):1185–1198, 2023.
- [9] M. J. Nguyen, B. Kieu-Do, and C. K. Koc. *A High-Performance Hardware Implementation of CRYSTALS-Kyber*. In Proc. HOST, pp. 123–134, 2023.
- [10] M. Krausz, G. Jati, S. Saha, and D. Lee. *A Configurable and Efficient Hardware Implementation of ML-KEM*. IACR Trans. CHES, 2024(2):213–236, 2024.
- [11] P. Navarro-Torrero, M. Krausz, D. Lee, and A. O. Otero. *Optimized Hardware Accelerator for ML-KEM*. In Proc. DATE, pp. 1–6, 2025.
- [12] Y. Xing and S. Li. *Hardware Implementation of CRYSTALS-Kyber*. In Proc. FCCM, pp. 144–152, 2021.
- [13] Xilinx Inc. *Vivado Design Suite User Guide: Synthesis (UG901)*. v2019.2, 2019.
- [14] Xilinx Inc. *7 Series FPGAs Data Sheet: Overview (DS180)*. v2.6, 2020.