FPGA 创新设计大赛 AMD 赛道命题式赛道 - 设计报告

1. 项目概述

1.1 项目背景

在 AI 和大数据驱动的计算需求爆炸性增长，而通用 CPU 性能提升乏力的双重压力下，整个行业从追求“通用计算性能”转向了“领域专用计算”，通过对基础“算子”进行极致的软硬件协同优化，成为提升计算速度、打破性能瓶颈的核心手段。

1.2 设计目标

优化 HLS 算子，分别对加速常见安全相关算法的开源软件库 Vitis Security Library，加速大规模线性系统求解和矩阵分解的软件库 Vitis Solver Library，对数据压缩和解压算法进行硬件加速的开源库 Vitis Data Compression Library 进行加速。

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/C++ 行为级仿真：验证用 C/C++ 编写的内核代码功能是否正确，而不关心具体的硬件时序。RTL 行为级仿真：验证综合后产生的 RTL 代码（Verilog/VHDL）的功能是否正确，并开始进行初步的时序分析。

2. 设计原理和功能框图

2.1 算法原理

对称加密：一种置换-置换网络。其核心操作不是在数值上做大的算术运算，而是在数据块（128 位，即 16 个字节）上进行一系列可逆的、混淆和扩散的轮操作。

哈希函数： 将输入消息填充、分块，然后对每个消息块进行一系列逻辑运算和算术加法，迭代地更新一个内部状态（哈希值），直到处理完所有块。

Cholesky 分解：用于求解对称正定矩阵的线性系统。它将矩阵 A 分解为一个下三角矩阵 L 和其转置的乘积：$A = L * L^T$。然后通过前代和回代法快速求解。

2.2 系统架构设计

2.2.1 顶层架构

[描述整体系统架构，包括主要模块划分]

```
[在此插入系统顶层架构图]
┌──────────────────────────────────────────────────────────────────────────┐
│                              顶层设计                                    │
├──────────────┬──────────────┬──────────────┬──────────────┬──────────────┤
│  输入模块    │  计算核心    │  控制模块    │  输出模块    │              │
└──────────────┴──────────────┴──────────────┴──────────────┴──────────────┘
```

2.2.2 核心计算模块设计
模块功能说明：
模块 A：加速算法的核心思路是 “化软为硬，串行改并行” 。它将原本由软件顺序执行的复杂加密算法（如 SHA、SM 等），通过定制化的硬件电路（如在 FPGA 上）来实现。其精髓在于将计算过程分解成多个步骤形成“流水线”，使数据像工厂装配线上的零件一样被连续处理；同时将大量计算任务拆解成多个可同时执行的“并行”操作。这种从通用软件到专用硬件的转变，如同将单车道的普通公路升级为多车道的高速公路，从而实现了吞吐量和效率的数量级提升，特别适用于区块链、金融交易等需要高性能安全计算的场景。
模块 B：算法层面，根据矩阵规模大小采用不同的策略：小矩阵完全展开实现极致并行，中等矩阵部分展开兼顾资源与性能，大矩阵采用分块处理减少内存瓶颈。同时要保证数值稳定性，特别是固定点数运算时使用高精度中间变量避免误差累积。架构设计上，采用数据流式处理让多个计算阶段重叠执行，消除等待时间；使用乒乓双缓冲技术实现计算与数据传输的并行；对于复杂算子如 SVD，设计专门的流水线架构将计算分解为可并行执行的子任务。内存访问优化是关键环节，通过数组分区策略匹配数据访问模式：行主导访问按行分区，列主导访问按列分区，随机访问则完全分区。同时利用 FPGA 的多级存储体系，将频繁访问的小数据放在高速存储器中，大数据块使用大容量存储。计算并行化方面，智能确定循环展开因子，在资源约束内最大化并行度；充分利用 DSP48 单元的级联特性，将乘加操作组合成计算链提高 DSP 利用率；通过指令级并行让多个运算同时进行。数据重用通过分块处理提高数据局部性，减少外部内存访问；将多个计算步骤融合，避免中间结果存储开销；设计数据复用模式让同一数据被多个计算单元共享使用。通信优化着重于减少接口握手开销，采用批量流数据处理；宽数据总线一次传输多个数据元素；优化数据打包格式提高传输效率。
最后还要进行资源感知优化，根据目标 FPGA 平台的 DSP、BRAM、URAM 等资源情况，动态调整并行化策略，在资源约束下达到最佳性能平衡。
模块 C：针对 LZ 系列压缩算法的硬件加速优化，我们主要从并行处理、内存访问、算法效率和系统级设计等方面入手：通过多级流水线和并行匹配引擎提升吞吐量；采用智能预取和多级缓存减少内存延迟；使用自适应匹配策略和分层哈希提高匹配效率；优化数据流架构 with 异步处理和负载均衡；根据资源约束动态调整并行度；利用位级并行和预测执行加速关键操作；并通过内存带宽优化和流水线平衡提升整体性能。这些措施预计可带来 3-5 倍吞吐量提升、30-50%延迟降低和 40%能效改善。
2.2.3 数据流图
2.3 接口设计
接口规格：

输入接口：[位宽、时序、协议]
输出接口：[位宽、时序、协议]
控制接口：[控制信号说明]
3. 优化方向选择与原理
3.1 优化目标分析
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：
LUTRAM 优化　：小尺寸数据结构使用 LUTRAM，节省 BRAM 资源
压缩编码存储　：使用霍夫曼编码等压缩格式存储频率表
稀疏数据结构　：对不常用的数据采用稀疏存储方式
运行时优化：
动态缓冲区分配　：根据输入数据特性调整缓冲区大小
流式处理　：避免中间结果的大规模存储，采用流水线处理
内存访问合并　：将多个小访问合并为突发传输，提高效率
具体措施：

数据重用策略：数据重用策略的核心在于最大化利用已处理数据，减少重复计算和内存访问。在 LZ 压缩中，通过滑动窗口机制实现数据重用：维护一个固定大小的窗口（如 4KB），持续将新数据与窗口内的历史数据进行匹配，从而利用局部性原理。在解压缩过程中，采用历史缓冲区存储最近解压的数据，通过偏移量直接访问这些数据用于后续匹配。此外，通过流水线设计和并行处理，多个处理单元可以共享和复用中间结果，如哈希值和匹配位置，进一步降低冗余访问。在可选优化库中，动态缓冲区分配允许根据输入数据特性调整重用策略，例如对重复模式高的数据增大窗口大小，提高重用效率。
存储层次优化：存储层次优化涉及合理利用不同存储资源（寄存器、BRAM、URAM）以平衡性能和面积。最频繁访问的数据（如当前处理窗口、匹配状态）存储在寄存器中，实现零延迟访问；中等规模数据结构（如哈希字典、历史缓冲区）使用 BRAM，通过数组分区和并行访问提高吞吐量；大规模数据（如完整历史记录）则映射到 URAM 或外部内存，通过突发传输优化带宽。优化还包括根据目标设备特性动态调整存储分配，例如在资源受限的 FPGA 上优先使用 BRAM 而非 URAM，并通过数据压缩技术（如位宽压缩）减少存储需求。存储访问模式也经过优化，顺序访问优先，随机访问通过哈希映射和索引压缩减少冲突。
缓存设计：缓存设计旨在减少内存延迟和提高数据访问效率。采用多级缓存架构：第一级是寄存器缓存，用于存储极频繁访问的数据（如当前字节序列）；第二级是片上 BRAM 缓存，作为历史数据和字典的快速缓冲区；第三级是预取机制，提前加载可能需要的数据到缓存中。缓存行大小经过优化，匹配处理单元的数据宽度（如 64 位或 512 位），以最大化总线利用率。此外，智能替换策略（如 LRU 或基于访问频率）确保缓存命中率最高。在解压缩过程中，缓存设计还包括字节对齐优化，支持任意偏移量的快速读取，并通过流水线隐藏缓存访问延迟。可选优化库中的自适应缓存根据数据模式动态调整缓存大小和策略，以适应不同工作负载。
3.2.2 流水线优化
优化原理：流水线优化的根本原理是将复杂的压缩处理过程分解为多个简单阶段，让不同数据块在不同阶段同时处理，实现"多任务并行"的效果，从而大幅提升处理效率。
具体措施：
循环展开：完全展开　：对固定次数的循环（如处理 4 字节窗口）进行完全展开，消除循环控制开销部分展开　：对大循环采用部分展开，每次迭代处理多个数据元素（如 4 个字节）条件展开　：根据数据特征动态选择展开因子，对高重复率数据采用更大展开因子。资源约束展开　：根据目标设备的 DSP、BRAM 资源情况，智能调整展开程度
流水线插入：六阶段划分　：数据输入→哈希计算→字典查找→匹配比较→结果选择→数据输出寄存器隔离　：每个阶段间插入流水线寄存器，确保时序收敛和时钟频率最大化平衡设计　：通过延迟均衡技术确保各阶段处理时间相近，避免瓶颈效应流控机制　：采用 valid/ready 握手协议处理背压和数据流动控制数据依赖处理：

3.2.3 并行化优化
优哈希计算并行 ：多个哈希单元同时计算不同位置的哈希值
匹配查找并行 ：并行比较多个候选位置，加速最佳匹配查找
编码输出并行 ：同时进行长度编码和偏移编码，减少处理延迟化原理：
具体措施：
任务级并行：[描述]
数据级并行：[描述]
指令级并行：[描述]
3.3 HLS 指令优化
#pragma HLS latency min=1 max=5
#pragma HLS UNROLL factor=4
#pragma HLS LOOP_TRIPCOUNT min=1024 max=65536
#pragma HLS DATAFLOW
#pragma HLS PIPELINE II=1
#pragma HLS UNROLL factor=4
#pragma HLS ARRAY_PARTITION variable=array type=block factor=2

4. LLM 辅助优化记录

大模型辅助使用记录

- **模型名称**：deepseekR1
- **提供方 / 访问 API**：豆包
- **主要用途**：代码重构,优化建议
- **完整 Prompt 内容**：

```
  /*
 * Copyright 2019 Xilinx, Inc.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *        http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

/**
 * @file hmac.hpp
```

```
 * @brief header file for HMAC.
 * This file part of Vitis Security Library.
 * TODO
 * @detail .
 */

#ifndef _XF_SECURITY_HMAC_HPP_
#define _XF_SECURITY_HMAC_HPP_

#include <ap_int.h>
#include <hls_stream.h>
#include <xf_security/types.hpp>

#if !defined(__SYNTHESIS__) && XF_SECURITY_DECRYPT_DEBUG == 1
#include <iostream>
#endif
namespace xf {
namespace security {

namespace internal {
// typedef ap_uint<64> u64;

template <int lW, int keyLen>
void expandStrm(hls::stream<bool>& eInStrm, hls::stream<bool>& eOutStrm,
hls::stream<ap_uint<lW> >& lenStrm) {
    while (!eInStrm.read()) {
        eOutStrm.write(false);
        lenStrm.write(ap_uint<lW>(keyLen));
    }
    eOutStrm.write(true);
}

template <int dataW, int hshW, int blockSize>
void genPad(hls::stream<ap_uint<hshW> >& keyHashStrm,
            hls::stream<bool>& ekeyHashStrm,
            hls::stream<ap_uint<blockSize * 8> >& kipadStrm,
            hls::stream<ap_uint<blockSize * 8> >& kopadStrm,
            hls::stream<bool>& eKipadStrm) {
    while (!ekeyHashStrm.read()) {
#pragma HLS pipeline II = 1
        ap_uint<blockSize* 8> kipad = 0;
        ap_uint<blockSize* 8> kopad = 0;
        ap_uint<blockSize* 8> k1 = 0;
```

```
            ap_uint<hshW> keyHash = keyHashStrm.read();
            for (int i = 0; i < hshW / dataW; i++) {
#pragma HLS unroll
                k1.range(blockSize * 8 - i * dataW - 1, blockSize * 8 - (i + 1) * dataW) =
                    keyHash.range(i * dataW + dataW - 1, i * dataW);
            }
            for (int i = 0; i < blockSize; i++) {
#pragma HLS unroll
                kipad.range(i * 8 + 7, i * 8) = k1.range(i * 8 + 7, i * 8) ^ 0x36;
                kopad.range(i * 8 + 7, i * 8) = k1.range(i * 8 + 7, i * 8) ^ 0x5c;
            }
            kipadStrm.write(kipad);
            kopadStrm.write(kopad);
            eKipadStrm.write(false);
        }
        eKipadStrm.write(true);
}

template <int dataW, int lW, int hshW, int keyLen, int blockSize, template <int iW, int ilW, int oW>
class F>
void kpadHash(hls::stream<ap_uint<dataW> >& keyStrm,
                    hls::stream<bool>& eStrm,
                    hls::stream<ap_uint<blockSize * 8> >& kipadStrm,
                    hls::stream<ap_uint<blockSize * 8> >& kopadStrm,
                    hls::stream<bool>& eKipadStrm) {
#pragma HLS dataflow

    hls::stream<bool> eKeyStrm;
#pragma HLS stream variable = eKeyStrm depth = 4
#pragma HLS resource variable = eKeyStrm core = FIFO_LUTRAM
    hls::stream<ap_uint<lW> > keyLenStrm;
#pragma HLS stream variable = keyLenStrm depth = 4
#pragma HLS resource variable = keyLenStrm core = FIFO_LUTRAM
    hls::stream<ap_uint<hshW> > keyHashStrm;
#pragma HLS stream variable = keyHashStrm depth = 4
#pragma HLS resource variable = keyHashStrm core = FIFO_LUTRAM
    hls::stream<bool> ekeyHashStrm;
#pragma HLS stream variable = ekeyHashStrm depth = 4
#pragma HLS resource variable = ekeyHashStrm core = FIFO_LUTRAM

    expandStrm<lW, keyLen>(eStrm, eKeyStrm, keyLenStrm);

    F<dataW, lW, hshW>::hash(keyStrm, keyLenStrm, eKeyStrm, keyHashStrm, ekeyHashStrm);
```

```
    genPad<dataW, hshW, blockSize>(keyHashStrm, ekeyHashStrm, kipadStrm, kopadStrm, eKipadStrm);
}

template <int dataW, int lW, int hshW, int keyLen, int blockSize, template <int iW, int ilW, int oW> class F>
void kpad(hls::stream<ap_uint<dataW> >& keyStrm,
          hls::stream<bool>& eStrm,
          hls::stream<ap_uint<blockSize * 8> >& kipadStrm,
          hls::stream<ap_uint<blockSize * 8> >& kopadStrm,
          hls::stream<bool>& eKipadStrm) {
    if (keyLen > blockSize) {
        kpadHash<dataW, lW, hshW, keyLen, blockSize, F>(keyStrm, eStrm, kipadStrm, kopadStrm, eKipadStrm);
    } else {
        while (!eStrm.read()) {
            ap_uint<blockSize* 8> k1 = 0;
            for (int i = 0; i < ((keyLen * 8 + dataW - 1) / dataW); i++) {
#pragma HLS pipeline II = 1
                ap_uint<dataW> tmp = keyStrm.read();
                k1 <<= dataW;
                k1.range(dataW - 1 + ((blockSize - keyLen) * 8), ((blockSize - keyLen) * 8)) = tmp;
                // k1.range(blockSize * 8 - 1 - i * dataW, blockSize * 8 - (i + 1) * dataW) = tmp;
            }
            ap_uint<blockSize* 8> kipad = 0;
            ap_uint<blockSize* 8> kopad = 0;
            for (int i = 0; i < blockSize; i++) {
#pragma HLS unroll
                kipad(i * 8 + 7, i * 8) = 0x36 ^ k1.range(i * 8 + 7, i * 8);
                kopad(i * 8 + 7, i * 8) = 0x5c ^ k1.range(i * 8 + 7, i * 8);
            }
            kipadStrm.write(kipad);
            kopadStrm.write(kopad);
            eKipadStrm.write(false);
        }
        eKipadStrm.write(true);
    }
}

template <int dataW, int lW, int hshW, int blockSize>
void mergeKipad(hls::stream<ap_uint<blockSize * 8> >& kipadStrm,
                hls::stream<ap_uint<blockSize * 8> >& kopadInStrm,
                hls::stream<ap_uint<dataW> >& msgStrm,
```

```
                    hls::stream<ap_uint<lW> >& msgLenStrm,
                    hls::stream<bool>& eLenStrm2,
                    hls::stream<ap_uint<dataW> >& mergeKipadStrm,
                    hls::stream<ap_uint<lW> >& mergeKipadLenStrm,
                    hls::stream<bool>& eMergeKipadLenStrm,
                    hls::stream<ap_uint<blockSize * 8> >& kopadOutStrm) {
    while (!eLenStrm2.read()) {
        eMergeKipadLenStrm.write(false);

        ap_uint<lW> ml = msgLenStrm.read();
        ap_uint<lW> mergeKipadLen = ml + blockSize;

        mergeKipadLenStrm.write(mergeKipadLen);

        ap_uint<blockSize* 8> kipad = kipadStrm.read();

        for (int i = 0; i < ((blockSize * 8 + dataW - 1) / dataW); i++) {
#pragma HLS pipeline II = 1
            // mergeKipadStrm.write(kipad.range(blockSize * 8 - 1 - i * dataW, blockSize * 8 - (i
+ 1) * dataW));
            mergeKipadStrm.write(kipad.range(blockSize * 8 - 1, blockSize * 8 - dataW));
            kipad <<= dataW;
        }

        kopadOutStrm.write(kopadInStrm.read());

        for (int i = 0; i < ((ml * 8 + dataW - 1) / dataW); i++) {
#pragma HLS pipeline II = 1
            mergeKipadStrm.write(msgStrm.read());
        }
    }
    eMergeKipadLenStrm.write(true);
}

template <int dataW, int lW, int hshW, int keyLen, int blockSize, template <int iW, int ilW, int oW>
class F>
void msgHash(hls::stream<ap_uint<blockSize * 8> >& kipadStrm,
             hls::stream<ap_uint<blockSize * 8> >& kopadInStrm,
             hls::stream<ap_uint<dataW> >& msgStrm,
             hls::stream<ap_uint<lW> >& msgLenStrm,
             hls::stream<bool>& eLenStrm,
             hls::stream<ap_uint<blockSize * 8> >& kopadOutStrm,
             hls::stream<ap_uint<hshW> >& msgHashStrm,
             hls::stream<bool>& eMsgHashStrm) {
```

```
#pragma HLS dataflow

    hls::stream<ap_uint<dataW> > mergeKipadStrm;
#pragma HLS stream variable = mergeKipadStrm depth = 128
#pragma HLS resource variable = mergeKipadStrm core = FIFO_BRAM
    hls::stream<ap_uint<lW> > mergeKipadLenStrm;
#pragma HLS stream variable = mergeKipadLenStrm depth = 4
#pragma HLS resource variable = mergeKipadLenStrm core = FIFO_LUTRAM
    hls::stream<bool> eMergeKipadLenStrm;
#pragma HLS stream variable = eMergeKipadLenStrm depth = 4
#pragma HLS resource variable = eMergeKipadLenStrm core = FIFO_LUTRAM

    mergeKipad<dataW, lW, hshW, blockSize>(kipadStrm, kopadInStrm, msgStrm, msgLenStrm, eLenStrm, mergeKipadStrm,
                                           mergeKipadLenStrm, eMergeKipadLenStrm, kopadOutStrm);

    F<dataW, lW, hshW>::hash(mergeKipadStrm, mergeKipadLenStrm, eMergeKipadLenStrm, msgHashStrm, eMsgHashStrm);
}

template <int dataW, int lW, int hshW, int keyLen, int blockSize>
void mergeKopad(hls::stream<ap_uint<blockSize * 8> >& kopadStrm,
                hls::stream<ap_uint<hshW> >& msgHashStrm,
                hls::stream<bool>& eMsgHashStrm,
                hls::stream<ap_uint<dataW> >& mergeKopadStrm,
                hls::stream<ap_uint<lW> >& mergeKopadLenStrm,
                hls::stream<bool>& eMergeKopadLenStrm) {
    while (!eMsgHashStrm.read()) {
        eMergeKopadLenStrm.write(false);
        mergeKopadLenStrm.write(ap_uint<lW>(blockSize + hshW / 8));

        ap_uint<blockSize* 8> kopad = kopadStrm.read();
        ap_uint<hshW> msgHash = msgHashStrm.read();

        for (int i = 0; i < ((blockSize * 8 + dataW - 1) / dataW); i++) {
#pragma HLS pipeline II = 1
            mergeKopadStrm.write(kopad.range(blockSize * 8 - 1, blockSize * 8 - dataW));
            kopad <<= dataW;
        }

        for (int i = 0; i < ((hshW + dataW - 1) / dataW); i++) {
#pragma HLS pipeline II = 1
            mergeKopadStrm.write(msgHash.range(dataW - 1, 0));
```

```
                msgHash >>= dataW;
            }
        }
        eMergeKopadLenStrm.write(true);
}

template <int dataW, int lW, int hshW, int keyLen, int blockSize, template <int iW, int ilW, int oW>
class F>
void resHash(hls::stream<ap_uint<blockSize * 8> >& kopadStrm,
                    hls::stream<ap_uint<hshW> >& msgHashStrm,
                    hls::stream<bool>& eMsgHashStrm,
                    hls::stream<ap_uint<hshW> >& hshStrm,
                    hls::stream<bool>& eHshStrm) {
#pragma HLS dataflow

    hls::stream<ap_uint<dataW> > mergeKopadStrm;
#pragma HLS stream variable = mergeKopadStrm depth = 4
#pragma HLS resource variable = mergeKopadStrm core = FIFO_LUTRAM
    hls::stream<ap_uint<lW> > mergeKopadLenStrm;
#pragma HLS stream variable = mergeKopadLenStrm depth = 4
#pragma HLS resource variable = mergeKopadLenStrm core = FIFO_LUTRAM
    hls::stream<bool> eMergeKopadLenStrm;
#pragma HLS stream variable = eMergeKopadLenStrm depth = 4
#pragma HLS resource variable = eMergeKopadLenStrm core = FIFO_LUTRAM

    mergeKopad<dataW, lW, hshW, keyLen, blockSize>(kopadStrm, msgHashStrm,
eMsgHashStrm, mergeKopadStrm,
                                                   mergeKopadLenStrm,
eMergeKopadLenStrm);

    F<dataW, lW, hshW>::hash(mergeKopadStrm, mergeKopadLenStrm, eMergeKopadLenStrm,
hshStrm, eHshStrm);
}

template <int dataW, int lW, int hshW, int keyLen, int blockSize, template <int iW, int ilW, int oW>
class F>
void hmacDataflow(hls::stream<ap_uint<dataW> >& keyStrm,
                  hls::stream<ap_uint<dataW> >& msgStrm,
                  hls::stream<ap_uint<lW> >& msgLenStrm,
                  hls::stream<bool>& eLenStrm,
                  hls::stream<ap_uint<hshW> >& hshStrm,
                  hls::stream<bool>& eHshStrm) {
#pragma HLS dataflow
    hls::stream<bool> eKipadStrm;
```

```
#pragma HLS stream variable = eKipadStrm depth = 4
#pragma HLS resource variable = eKipadStrm core = FIFO_LUTRAM

    hls::stream<ap_uint<blockSize * 8> > kipadStrm;
#pragma HLS stream variable = kipadStrm depth = 4
#pragma HLS resource variable = kipadStrm core = FIFO_LUTRAM
    hls::stream<ap_uint<blockSize * 8> > kopadStrm;
#pragma HLS stream variable = kopadStrm depth = 4
#pragma HLS resource variable = kopadStrm core = FIFO_LUTRAM
    hls::stream<ap_uint<blockSize * 8> > kopad2Strm;
#pragma HLS stream variable = kopad2Strm depth = 4
#pragma HLS resource variable = kopad2Strm core = FIFO_LUTRAM

    hls::stream<ap_uint<hshW> > msgHashStrm;
#pragma HLS stream variable = msgHashStrm depth = 4
#pragma HLS resource variable = msgHashStrm core = FIFO_LUTRAM
    hls::stream<bool> eMsgHashStrm;
#pragma HLS stream variable = eMsgHashStrm depth = 4
#pragma HLS resource variable = eMsgHashStrm core = FIFO_LUTRAM

    kpad<dataW, lW, hshW, keyLen, blockSize, F>(keyStrm, eLenStrm, kipadStrm, kopadStrm,
eKipadStrm);

    msgHash<dataW, lW, hshW, keyLen, blockSize, F>(kipadStrm, kopadStrm, msgStrm,
msgLenStrm, eKipadStrm, kopad2Strm,
                                                   msgHashStrm, eMsgHashStrm);

    resHash<dataW, lW, hshW, keyLen, blockSize, F>(kopad2Strm, msgHashStrm, eMsgHashStrm,
hshStrm, eHshStrm);
}

} // end of namespace internal

/**
 * @brief Compute HMAC value according to specified hash function and input data.
 *
 *    keyW, keyStrm, keyLenStrm, msgW, msgStrm, and msgLenStrm would be used as
 *    parameters or input for the hash function, so they need to align with the API
 *    of the hash function.
 *
 *    Hash function is wrapped to a template struct which must have a static function named
`hash`.
 *
 *    Take md5 for example::
```

```
 *
 *    template <int msgW, int lW, int hshW>
 *    struct md5_wrapper {
 *          static void hash(hls::stream<ap_uint<msgW> >& msgStrm,
 *                                   hls::stream<lW>& lenStrm,
 *                                   hls::stream<bool>& eLenStrm,
 *                                   hls::stream<ap_uint<hshW> >& hshStrm,
 *                                   hls::stream<bool>& eHshStrm) {
 *                xf::security::md5(msgStrm, lenStrm, eLenStrm, hshStrm, eHshStrm);
 *          }
 *    };
 *
 *    then use hmac like this,
 *
 *     xf::security::hmac<32, 32, 64, 128, 64, md5_wrapper>(...);
 *
 * @tparam dataW the width of input stream keyStrm and msgStrm.
 * @tparam lW the with of input msgLenstrm.
 * @tparam blockSize    the block size (in bytes) of the underlying hash function (e.g. 64 bytes for
md5 and SHA-1).
 * @tparam hshW the width of output stream hshStrm.
 * @tparam keyLen lenght of key (in bytes)
 * @tparam F a wrapper of hash function which must have a static fucntion named `hash`.
 *
 * @param keyStrm    input key stream.
 * @param msgStrm    input meassge stream.
 * @param msgLenStrm    the length stream of input message stream.
 * @param eLenStrm    the end flag of length stream.
 * @param hshStrm output stream.
 * @param eHshStrm end flag of output stream hshStrm.
 *
 */
template <int dataW, int lW, int hshW, int keyLen, int blockSize, template <int iW, int ilW, int oW>
class F>
void hmac(hls::stream<ap_uint<dataW> >& keyStrm,
                hls::stream<ap_uint<dataW> >& msgStrm,
                hls::stream<ap_uint<lW> >& msgLenStrm,
                hls::stream<bool>& eLenStrm,
                hls::stream<ap_uint<hshW> >& hshStrm,
                hls::stream<bool>& eHshStrm) {
      internal::hmacDataflow<dataW, lW, hshW, keyLen, blockSize, F>(keyStrm, msgStrm,
msgLenStrm, eLenStrm, hshStrm,
                                                                                  eHshStrm);
}
```

} // end of namespace security
} // end of namespace xf

#endif // _XF_SECURITY_HMAC_HPP_给出代码优化建议，以减少延迟为目标

- **模型输出摘要**：
1. 展开固定次数循环，消除迭代延迟
代码中存在大量基于固定块大小（blockSize）的循环（如 genPad、mergeKipad、mergeKopad 中按字节或数据宽度处理的循环），这些循环的迭代次数是编译时常量（由 blockSize 和 dataW 决定），可通过完全展开消除循环控制开销。
2. 消除数据依赖，提升流水线 II（启动间隔）
代码中部分循环存在数据依赖（如 kpad 函数中 k1 <<= dataW 的移位操作），导致流水线无法达到 II=1，成为延迟瓶颈。需重构逻辑，用并行位置计算替代串行移位。
3. 优化数据流缓冲，避免阻塞延迟
dataflow 模式下，前级函数的输出流若缓冲不足，会导致前级阻塞等待后级读取，增加延迟。需根据各阶段处理速度调整 stream 的 FIFO 深度。
4. 匹配数据宽度（dataW）与处理粒度
dataW 决定了每次次流传输的数据量，增大 dataW 可减少传输次数，降低延迟（需确保接口支持）。
5. 平衡数据流各阶段处理时间
hmacDataflow 包含 kpad→msgHash→resHash 三个阶段，若某阶段耗时过长（如 kpadHash 中的哈希计算），会成为瓶颈。需优化哈希函数 F::hash 的内部流水线。
优化哈希函数：
确保 F::hash（如 MD5、SHA）内部循环完全展开，流水线 II=1；
减少哈希计算中的共享资源冲突（如通过资源复制拆分关键路径）；
若哈希函数支持，采用预计算常量（如 SHA 的 K 值）减少实时计算延迟。
6. 合并串行操作，重叠处理
mergeKipad 和 mergeKopad 中，“传输 kpad/kipad” 与 “传输消息 / 消息哈希” 是串行的，可通过双缓冲（Double Buffering）重叠处理。
优化建议：
- **人工审核与采纳情况**：
采纳了 156 三点信息，有效降低了延迟。### 4.2 优化阶段二：[阶段名称]

4.2.1 优化目标

[描述第二阶段的优化目标]

4.2.2 Prompt 设计

用户输入：

```
[第二轮优化的 prompt]
```

4.2.3 LLM 回答

模型回答：

```
[LLM 的回答]
```

4.2.4 优化实施

[按照 4.1.4 的格式填写]

4.3 优化阶段三：[如有更多阶段，继续添加]

4.4 LLM 辅助优化总结

总体收益：

- 性能提升：[具体数值]
- 资源节省：[具体数值]
- 开发效率：[描述 LLM 如何提高开发效率]

经验总结：

- 有效的 prompt 设计要点：[总结]
- LLM 建议的可行性分析：[总结]
- 需要人工验证的关键点：[总结]

---

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
测试数据集：[描述测试数据]
评估指标： [列出所有评估指标]

5.2 综合结果对比

5.2.1 资源使用对比

资源类型	优化前	优化后	改善幅度	利用率(优化前)	利用率(优化后)
BRAM	[数量]	[数量]	[%]	[%]	[%]
DSP	[数量]	[数量]	[%]	[%]	[%]
LUT	[数量]	[数量]	[%]	[%]	[%]
FF	[数量]	[数量]	[%]	[%]	[%]

5.2.2 性能指标对比

性能指标	优化前	优化后	改善幅度
初始化间隔(II)	[周期]	[周期]	[%]
延迟(Latency)	[周期]	[周期]	[%]
吞吐率(Throughput)	[ops/s]	[ops/s]	[%]
时钟频率	[MHz]	[MHz]	[%]

5.2.3 复合性能指标

复合指标	优化前	优化后	改善幅度
性能/DSP 比 (MACs/DSP)	[值]	[值]	[%]
吞吐量/BRAM 比 (Throughput/BRAM)	[值]	[值]	[%]
能效比 (GOPS/W)	[值]	[值]	[%]

5.3 详细分析

5.3.1 资源优化分析

BRAM 优化效果：
[详细分析 BRAM 使用的优化效果，包括优化前后的存储映射策略变化]

DSP 优化效果：
[分析 DSP 使用效率的提升]

逻辑资源优化效果：**
[分析 LUT 和 FF 使用的变化]

5.3.2 性能优化分析

流水线效率提升：
[分析 II 降低带来的性能提升]

延迟优化效果：
三个算子都有不同程度的降低

吞吐率提升分析：

5.4 功耗分析

功耗类型	优化前	优化后	改善幅度
静态功耗(W)	[值]	[值]	[%]
动态功耗(W)	[值]	[值]	[%]
总功耗(W)	[值]	[值]	[%]

5.5 正确性验证

5.5.1 C 代码仿真结果

仿真配置：

- 测试用例数量：[数量]
- 测试数据类型：[描述]
- 精度要求：[描述]

仿真结果：

- 功能正确性：☑ 通过
- 输出精度：[具体精度指标]
- 性能验证：[描述]

5.5.2 联合仿真结果

仿真配置：

- RTL 仿真类型：Verilog
- 时钟周期：[ns]
- 仿真时长：[周期数]

仿真结果：

- 时序正确性：☑ 通过
- 接口兼容性：☑ 通过
- 性能匹配度：[%]

5.5.3 硬件验证（如适用）

[如果进行了板级验证，在此描述验证过程和结果]

---

6. 创新点总结

6.1 技术创新点

[列出本设计的主要技术创新点]

1. **创新点 1：** [描述]
2. **创新点 2：** [描述]
3. **创新点 3：** [描述]

6.2 LLM 辅助方法创新

通过详细向 AI 询问问题
6.3 工程实现创新

[描述在工程实现方面的创新点]

---

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
[问题 1]	[解决方案 1]	[效果描述]
[问题 2]	[解决方案 2]	[效果描述]

7.2 LLM 辅助过程中的问题

AI 对提出的问题不够清晰，产生了一些 AI 幻觉
8. 结论与展望

8.1 项目总结
三个算子的运行延迟都得到了大幅度的降低，程序运行更加流畅。
8.2 性能达成度
基本达到了降低延迟的优化目标
8.3 后续改进方向
继续进一步降低延迟
9. 参考文献

[1] [参考文献 1]
[2] [参考文献 2]

[3] [参考文献 3]

---

10. 附录

10.1 完整代码清单

[如需要，可以在此提供关键代码的完整清单]

10.2 详细仿真报告

[如需要，可以附上详细的仿真报告截图]

10.3 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]