大模型辅助使用记录
基本信息

模型名称：Deepseek

使用场景
主要用途
修改代码，完善代码。
完整 Prompt 内容
优化目标:

确定高性能 HMAC-SHA256 硬件架构

完整 Prompt 设计

用户输入：我正在 Xilinx Zynq-7020 上设计 HMAC-SHA256 硬件加速器，要求：

1.支持任意长度密钥和消息
2.资源使用优化
3.高吞吐量流水线设计
当前代码如下：

```
pragma HLS pipeline II=1
  uint32_t l = msg_strm.read()
     l = ((0x000000ffUL & l) << 24) | ((0x0000ff00UL & l) << 8) |
                  ((0x00ff0000UL & l) >> 8) | ((0xff000000UL & l) >> 24);
              b0.M[i] = l;
           }
       blk_strm.write(b0);
       }


       char left = (char)(len & 0x3fULL);


       if (left == 0) {
           SHA256Block b;pragma HLS array_partition variable = b.M complete
           b.M[0] = 0x80000000UL;
           for (int i = 1; i < 14; ++i) {pragma HLS unroll
               b.M[i] = 0;
           }
           b.M[14] = (uint32_t)(0xffffffffUL & (L >> 32));
           b.M[15] = (uint32_t)(0xffffffffUL & (L));
           blk_strm.write(b);
       } else if (left < 56) {
           SHA256Block b;
```

4.1.3 LLM 回答

建议三级数据流架构：

密钥处理级：动态填充 + ipad/opad 生成
内层哈希级：完全流水线 SHA-256 核心

外层哈希级：优化 SHA-256 计算

优化实施

- 采用的建议
- 数组循环分区：4 路分区改善内存访问并行性
- 深度流水线重构：3 级子流水线分解关键路径

代码修改

- 优化前代码：

串行消息调度

```
for (int t = 16; t < 64; t++) {
W[t] = SSIG1(W[t-2]) + W[t-7] + SSIG0(W[t-15]) + W[t-16];
}
```

单周期轮函数

```
T1 = h + BSIG1(e) + CH(e, f, g) + Kt + Wt;
T2 = BSIG0(a) + MAJ(a, b, c);
a = T1 + T2;
```

基础流接口

```
hls::stream<ap_uint<32> > w_strm;
```

优化后代码：

并行消息调度

```
pragma HLS array_partition variable=W cyclic factor=4
for (int t = 16; t < 64; t += 2) {
pragma HLS pipeline II=1
pragma HLS dependence variable=W inter false
W[t] = SSIG1(W[t-2]) + W[t-7] + SSIG0(W[t-15]) + W[t-16];
W[t+1] = SSIG1(W[t-1]) + W[t-6] + SSIG0(W[t-14]) + W[t-15];
}
```

3 级流水线轮函数

第 1 级：预计算

```
ch_pre = CH(e, f, g);
maj_pre = MAJ(a, b, c);
bsig1_pre = BSIG1(e);
bsig0_pre = BSIG0(a);
```

第 2 级：中间计算

```
T1_part = h + bsig1_pre + ch_pre;
T2_part = bsig0_pre + maj_pre;
```

第 3 级：最终计算

```
T1 = T1_part + Kt + Wt;
```

```
T2 = T2_part;
a = T1 + T2;
```

深度优化流接口

pragma HLS stream variable=w_strm depth=64

pragma HLS stream variable=blk_strm depth=32

pragma HLS stream variable=mergeStrm depth=128

实施效果：

II 改善：消息调度模块 II 从 64 降低到 32（提升 50%），关键路径 II 目标从 64 降低到 4
BRAM 使用改善：从预估 8 个 BRAM 降低到 3 个 BRAM（减少 62.5%），利用率从 5.7%降低到 2.1%

模型输出摘要

问题 1：时序分析能力有限

问题描述：LLM 难以准确预测具体综合工具下的时序结果，建议的优化方案有时效果不佳
解决方法：采用迭代优化策略，先进行小范围试验验证，再推广应用有效方案
学习收获：建立"LLM 建议→小规模验证→大规模应用"的稳健优化流程

问题 2：LLM 生成的架构方案过于理想化

问题描述：初始 LLM 建议的 300MHz 目标频率和 II=1 的流水线设计在实际综合中无法实现

解决方法：结合工程经验对 LLM 建议进行可行性评估，调整为现实的 100MHz 目标，重点关注可实现的优化点

学习收获：认识到 LLM 在理论优化方面的优势，但需要人工结合具体硬件约束进行修正
8. 结论与展望

◆ 模型输出摘要
主要优化改进：
➢ 头文件更新：
使用 C++标准头文件（cstdlib、cstdint 等）
添加 hls_vector.h 用于更好的向量化支持
➢ 接口优化：
使用 s_axilite 接口替代旧的 ap_ctrl_hs
为控制信号添加适当的接口绑定
➢ 流处理改进：
使用 write()和 read()方法替代和>>操作符
改进流空检查和数据可用性检查
➢ 错误处理增强：
添加文件操作错误检查
➢ 改进空文件处理
添加读取操作的状态检查
➢ 性能优化：

增大缓冲区大小（4KB）
改进数据流控制逻辑
添加更详细的进度输出

- 代码健壮性：

使用 static const 替代宏定义常量，改进变量初始化，添加边界检查，编译兼容性。
使用 Vitis HLS 2024.2 推荐的编译指令
改进数据类型一致性
人工审核与采纳情况
增大缓冲区大小，使用 s_axilite 接口替代旧的 ap_ctrl_hs，让变量初始化。
学习收获
通过与大模型交互，您学到了哪些新知识或优化技巧？
问法很重要。