EDA作业2

2- Ch3.高层次综合

1. 简述高层次综合的*主要技术问题*有哪些？（开放题）

高层次综合（HLS，High-Level Synthesis）是将高级语言描述（如C、C++、SystemC等）转换为硬件描述语言（HDL，如Verilog或VHDL）的过程。HLS 包括前端编译阶段 与 后端调度与绑定，以及最后 RTL 生成阶段。

HLS前端

- 输入：形式化高层次语言
- 语法及词法分析
- 中间表示
- 优化
- RTL代码生成

HLS后端

将前端生成的IR进一步转换为CDFG，根据CDFG生成满足时序和资源约束的硬件，最后再将硬件框架转为RTL代码。

HLS后端的关键步骤如下：

- 调度
- 绑定（分配）

调度，将CDFG中的数据流图（DFG）中的各个运算分配至一个确定的时钟周期中执行。将每一个运算的起始之行时间赋值给一个确定的控制步。控制步是调度算法中最基本的时序单位，通常为一个时钟周期。调度考虑硬件的可实现性外，还需要使使生成的硬件满足给定的约束条件（时序、面积），并在满足约束条件下优化。
调度算法分为静态调度和动态调度。

1. *静态调度*：在硬件生成前将所有运算固定到确定控制步
    1. 目标
        1. 在给定的控制步数内，使得运算资源总量最小化，即实现面积最优
        2. 在给定的运算资源总量限制下，使得所需的控制步数量最小化，即实现延时最优
    2. 静态调度算法
       CDFG以图的形式表示，节点表示运算，边表示运算之间的依赖。静态调度算法将图上的节点固定到某个控制步。
        1. ASAP

    2. ALSP
    3. 列表调度
    4. 整数线性规划调度
2. *动态调度*：根据实时硬件执行条件，动态确定一部分数据流的控制步
    1. 与静态调度对比
       静态调度往往针对硬件运行最坏的情况，即控制步最长路径，相对保守
    2. 动态调度怎么做
       引入延迟不敏感的通信模式，根据实时的数据执行情况动态确定后续执行所需的控制步，从而只有当数据依赖关系出现时才需要进行数据等待，而其他时候不需要遵守最坏的情况，提高了执行效率

绑定&分配，将运算与硬件实例进行一一映射以及融合。

1. 分配，是指在设计中确定所需的硬件资源类型和数量。
    1. 目标
        1. 提高资源利用率
        2. 性能优化
    2. 分配算法
        1. 静态分配
        2. 动态分配
        3. 启发式分配 如贪心算法
        4. 基于图的分配，如图着色
        5. 线性规划
        6. 动态规划
2. 绑定，是在已分配的资源中为调度好的操作分配具体的硬件资源。
    1. 优化目标
       资源共享，同类型但处于不同控制步的运算单元采用同一硬件实体进行实现
    2. 算法
        1. 图着色
           目标是使用最少数量颜色对图上节点着色
            1. 根据DFG构建冲突图，节点为DFG中同类型节点，边表示节点之间资源冲突关系
            2. 约束条件是图上相连的节点不能使用相同颜色，最终着色图所需的颜色数则为该类型运算单元所需的个数
        2. 左边算法
           根据运算的起始和结束时间，将运算转化为方块表示，优先选择起始时间最早的方块按行排列，遇到无法排列的方块则新开一行，循环算法流程

\2. 请调研一下（IEEE Library），最近2-5年内，针对以下三个方向，高层次综合HLS的主要研究热点？请简单汇总下，包括研究点、paper title、年份（开放题）

-- 方向1：面向HLS的编译优化。关键词：loop pipelining, LLVM, HLS, array partitioning, load-store optimization
-- 方向2：HLS的调度算法等。关键词：HLS, scheduling, binding
-- 方向3：HLS的形式验证。关键词：HLS, formal verification, model checking

> HLS形式验证，是指用数学方法验证HLS生成硬件描述的正确性，而无需依赖传统仿真测试。
> 形式验证能够证明设计的某些属性在所有可能的输入情况下都满足，而仿真只能在特定输入下进行测试

HLS 编译优化

> loop pipelining, LLVM, HLS, array partitioning, load-store optimization

年份	Paper title	研究点	贡献
2022	# Reinforcement Learning Strategies for Compiler Optimization in High level Synthesis	强化学习指导编译优化	为编译器优化提供了新的视角和方法
2024	# Array Partitioning Method for Streaming Dataflow Optimization in High-level Synthesis	array partitioning	提出了一种有效的数组分区方法来缓解流数据 流的约束。 结论：在CNN案例上，性能增加 28.6%，功耗增加 7.2%

Reinforcement Learning Strategies for Compiler Optimization in High level Synthesis
高层次综合中编译优化的强化学习策略

背景：
随着硬件设计复杂性的增加，传统的编译器优化方法已难以满足日益增长的性能需求。研究现状表明，现有的编译器优化策略往往依赖于固定的优化顺序和重复模式，缺乏灵活性和适应性。

在高级综合编译器优化中使用强化学习（Reinforcement Learning, RL）策略来自动化发现最优策略的过程。训练一个RL模型来学习LLVM优化传递的顺序，以提高硬件质量。作者构建了一个基于Open AI Gym、Ray和TensorFlow的统一环境和API，用于强化学习框架。研究问题是如何通过强化学习提高编译器优化的效率。为此，作者提出了两种新的学习策略：
策略1考虑了编译优化过程中的传递顺序，通过改变观察空间和奖励频率来影响学习策略；
策略2则通过行动元组（Action Tuples）来考虑传递顺序，即在每一步中预测多个传递而不是单个传递。通过这些策略，作者分析了学习速度、性能潜力、波动带和相对于-O3标志的加

速比等指标。实验结果表明，选择合适的学习策略可以显著提高学习速度和性能潜力，同时减少波动。

Array Partitioning Method for Streaming Dataflow Optimization in High-level Synthesis

在 HLS 中，数据流是实现高并行性的关键微架构。然而，潜在通道上的顺序访问等严格条件往往会限制流数据流
本文提出了一种用于流数据流推理的有效阵列分区方法。关键是探索与流通道顺序访问要求相匹配的潜在阵列分区模式。
对卷积神经网络 (CNN) 的推理进行了实验案例研究。结果表明，与默认数据流相比，所提出的方法可以实现约 28.6% 的性能提升，而功耗增加 7.2%。

HLS 调度算法

> HLS, scheduling, binding

年份	Paper title	研究点	主要贡献
2020	Using Reduced Graphs for Efficient HLS Scheduling	HLS调度优化	提出 Reduced DFG操作被组织成图的层级，操作之间依赖关系仅部分通过层级排序保留，大大降低调度的复杂性 结论：调度时间x16，内存x(1/5)、硬件执行时间增加（ 0 到 6%）
2021	DASS: Combining Dynamic & Static Scheduling in High-Level Synthesis	动态调度与静态调度结合	结合静态调度与动态调度，实现获得双方好处的调度。 结论：在DASS的基准程序中，与动态调度相比，节省45%面积；与静态调度相比，x1.98加速
2024	Subgraph Extraction-Based Feedback-Guided Iterative Scheduling for HLS	迭代调度	提出了一种基于子图提取的反馈引导迭代调度算法（ISDC），在传统的静态依赖计算（SDC）方法基础上，通过迭代细化和下游工具反馈，显著减少了寄存器的使用量。 结论：与商业HLS相比，寄存器使用量减少28.5%

Using Reduced Graphs for Efficient HLS Scheduling

大大降低了调度的复杂性，从而提高了内存使用率并降低了计算工作量。结果表明，确定调度所需的时间平均加快了 16 倍，而内存使用量仅为其一小部分（平均 1/5），最终硬件执行时间仅增加 0 到 6% 的成本。
创新：

- 提出RDFG（CDFG的简化表示形式），操作被组织成图的层级，操作之间依赖关系仅部分通过层级排序保留
- 位于i+1层的操作只能在i层的操作后调度
  因此，处理RDFG比传统调度计算复杂度低，内存占用小

DASS: Combining Dynamic & Static Scheduling in High-Level Synthesis

- 静态调度 static scheduling，可以简化电路并实现更多的资源共享
  - 可以从优化中受益，最大限度减少生成电路的关键路径和资源需求。
- 动态调度 dynamic schedulin，在计算具有*非平凡的控制流*时提高硬件速度
  - 适合于处理不规则和控制主导的应用程序
  本文希望实现结合SS与DS优点的调度。

1. 确定输入程序中使用动态调度不会带来任何性能优势的部分，并对这些部分使用静态调度。
2. 然后，在为程序的其余部分创建数据流电路时，将这些静态调度的部分视为黑盒

Subgraph Extraction-Based Feedback-Guided Iterative Scheduling for HLS

基于子图提取的 HLS 反馈引导迭代调度
背景：

- 聚焦HLS的迭代调度问题，如何进行有效迭代以充分利用底层优化信息，同时保持合理的计算复杂度
- 现有调度方法依赖于初步的时序估计，忽略下游中低层次优化反馈信息，无法得到最优解。

提出了一种基于子图提取的反馈引导迭代调度算法（ISDC），用于硬件描述语言（HLS）的低层次综合。ISDC算法在传统的静态依赖计算（SDC）方法基础上，通过迭代细化和下游工具反馈，显著减少了寄存器的使用量。实验结果表明，与传统SDC调度算法相比，ISDC算法在寄存器使用量上有显著降低，证明了反馈引导优化在HLS中的有效性，并为未来的研究提供了新的方向。

创新：

1. 提出了一种新型的迭代调度算法ISDC，该算法通过整合下游工具的反馈信息，实现了对HLS设计更精细的调度优化；
2. 引入了基于子图提取的策略，通过关注关键寄存器的使用情况，提高了调度的效率和准确性

HLS 形式验证

> HLS, formal verification, model checking

年份	Paper title	研究点	主要贡献
2023	Synthesis-Embedded Verification	形式化验证	提出了一种新颖的Synthesis-Embedded Verification（综合嵌入式验证）流程
2023	Automatic Inductive Invariant Generation for Scalable Dataflow Circuit Verification	形式化验证	提出了一个全自动的框架，能够为数据流电路验证生成合适的归纳不变式。

Synthesis-Embedded Verification

背景

随着硬件系统复杂性的增加，传统的验证方法已难以满足高效、准确的验证需求。目前，形式化验证方法因其能够系统性地检测硬件设计中的缺陷而受到重视，但这些方法通常需要复杂的数学模型检查或等价性检查技术，且自动化程度不高

提出了一种将形式验证嵌入到综合流程中的方法，以及编译的FSM（有限状态机）模型。该方法通过使用CAS（C语言的仿真器）来验证硬件设计的正确性，无需复杂的数学模型检查或等价性检查技术。

创新

1. 首先，提出了一种将形式化验证嵌入到综合流程中的新方法，这使得验证过程自动化并且与综合过程紧密集成。
2. 其次，使用逻辑Horn谓词进行综合转换优化，并采用编译器编译器技术开发前端编译器，保证了从软件到硬件的转换过程的形式化。
3. 此外，该流程能够自动快速生成周期精确的模拟器（CAS），而无需用户具备复杂的数学模型检查或等价性检查技术知识。最后，该方法的自动化程度高，甚至CAS编译也是通过自动生成的批处理命令行文件完成的。

Automatic Inductive Invariant Generation for Scalable Dataflow Circuit Verification

背景：

高级综合（High-Level Synthesis, HLS）生成的数据流电路的验证问题。

提出了一个全自动框架，用于生成适合于可扩展数据流电路验证的归纳不变式。该框架利用高级综合（HLS）的多种洞见，系统地为任何从C代码生成的数据流电路传递相关的不变式信息。在一组代表性基准测试中，该方法显著优于先前的基于BDD的方法，能够在几分钟内证明BDD检查器无法在几天内证明的属性，同时只对验证能力进行了轻微的减少。此外，该方法还展示了其在先前的基于归纳的技术之上的有效性，能够在相同的运行时间内证明多达4倍的属性。

创新：

1. 本研究的创新性在于提出了一个全自动的框架，能够为数据流电路验证生成合适的归纳不变式。

2. 这一框架不仅适用于任何从C代码生成的数据流电路，而且能够显著提高验证的可扩展性和效率。此外，该框架通过系统地利用HLS的洞见，有效地传递相关的不变式信息，从而在保持高验证能力的同时，显著减少了验证所需的时间和资源。