RapidStream IR：面向 FPGA 高层次物理综合的基础设施

Jason Lau[†1,2], Yuanlong Xiao[†1], Yutong Xie[1], Yuze Chi[1], Linghao Song[2],
Shaojie Xiang[3], Michael Lo[2], Zhiru Zhang[1,3], Jason Cong[1,2], Licheng Guo[1]
[1]RapidStream Design Automation 公司 [2] 加利福尼亚大学洛杉矶分校 [3] 康奈尔大学
{lau,lcguo}@rapidstream-da.com

摘 要

大规模 FPGA 加速器日益提升的复杂度，使得同时兼顾高性能与开发效率变得愈发困难。高层次综合（HLS）因此被广泛采用，但高层描述与物理布局之间的不匹配往往导致工作频率不理想。尽管已有研究提出利用粗粒度设计分区、布局规划和流水化来改善频率的高层次物理综合方法，并取得了一定进展，但这些方法仍缺乏一个能够（1）在任意层次层级上对实际设计进行流水化，（2）整合 HLS 模块、供应商 IP 和手工编写的 RTL 设计，（3）移植到新兴目标 FPGA 器件，以及（4）便于扩展以实现新设计优化工具的框架。

本文提出 RapidStream IR，一套面向复杂 FPGA 设计的组合式表示与物理优化探索的实用高层次物理综合（HLPS）基础设施。该方法引入了灵活的中间表示（IR），可在任意层级表征互连协议、粗粒度流水化结构以及空间信息，从而支持可复用的频率优化流程。RapidStream IR 在多类混合来源设计上带来了 7% 至 62% 的频率提升，涵盖大语言模型和基因测序分析加速器等设计，并可移植到用户自定义的新 FPGA 平台。此外，我们通过案例研究进一步展示了其可扩展性，表明该框架能够支撑后续研究探索。

CCS 概念

• **硬件 → 高层次综合与寄存器传输级综合**; **分区与布局规划**; **EDA 软件工具**.

关键词

高层次综合、多芯粒 FPGA、频率、时序收敛、布局规划、数据流、流水化、延迟不敏感设计

ACM 引用格式:

Jason Lau, Yuanlong Xiao, Yutong Xie, Yuze Chi, Linghao Song, Shaojie Xiang, Michael Lo, Zhiru Zhang, Jason Cong, Licheng Guo. 2024. RapidStream IR: Infrastructure for FPGA High-Level Physical Synthesis. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '24), October 27–31, New York, NY, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3676536.3676649

本文为非正式中文译本，仅供个人学习与参考。如中英文内容存在差异，以英文原版为准。

[†]表示共同第一作者。

ICCAD '24, 2024 年 *10* 月 *27–31* 日, 美国纽约

ACM ISBN 979-8-4007-1077-3/24/10
https://doi.org/10.1145/3676536.3676649

1 引言

FPGA 向更大规模的多芯粒器件演进——例如双芯粒的 Versal VHK158 和三芯粒的 Alveo U280——增强了其加速复杂计算（如大语言模型）的能力，提升了性能与能效 [8, 45]。随着 FPGA 设计日趋复杂，设计者越来越依赖高层次综合（HLS）来管理复杂性，HLS 允许在算法级别描述设计并生成 RTL 代码，从而降低开发工作量 [10]。

然而，HLS 的抽象本身也带来了挑战，主要体现在物理优化方面。由于 HLS 设计描述不包含时序约束，缺少周期级精确性和物理布局信息，前端 HLS 与后端物理实现之间往往存在不匹配，阻碍了时序收敛 [17, 18]。当前 EDA 工具的可扩展性问题加剧了这一矛盾：在缺乏手动布局规划的情况下复制处理逻辑会对结果质量（QoR）产生负面影响 [27, 40]。

相比之下，RTL 专家了解模块的资源密集特性及其可能占用的大面积区域，因此会在相邻模块之间手动添加足够的流水级，以打断跨芯粒长线的延迟。然而，大多数 HLS 工具缺乏此类架构信息，往往无法执行此类优化，导致时序关键路径过长。此外，RTL 专家可以将设计划分为若干组并分配到特定芯粒上以均衡资源利用。遗憾的是，HLS 工具生成的逻辑块之间通常缺乏足够的流水级，迫使下游工具将这些块紧密放置以最小化总布线长度，从而引发局部布线拥塞。

为此，研究者提出了一类我们称之为**高层次物理综合（HLPS）**的技术，将粗粒度设计分区、布局规划和流水化相结合，以协同优化 HLS 与物理设计阶段，从而提升频率 [12, 16, 17, 25, 30, 32, 33, 42, 43]。在 HLPS 中，设计被划分为粗粒度的组，然后在 FPGA 上进行粗粒度布局规划，以最小化跨芯粒连接数。基于布局信息，在组间插入流水寄存器以打断长连接，使通信可以在多个时钟周期内完成，而不必受限于单个长延迟周期。此外，还需分析每个设计模块的资源需求，使其在 FPGA 上均衡分布，以避免局部拥塞并缓解有限布线资源引发的时序瓶颈。采用这一方法的代表性工作是 AutoBridge 框架 [17]，它通过减少跨芯粒长连接的影响并均匀分配设计资源，帮助构建跨多芯粒的高频 FPGA HLS 设计。

然而，现有的 FPGA HLPS 方案仍存在若干局限，使其难以应用于真实设计。例如，这些研究通常只关注由 AMD/Xilinx Vitis HLS 生成的一小类设计，且存在以下不足：

（1）不支持任意层次层级的设计优化；因此，所有任务级并行模块都需要在顶层 HLS 函数处互连 [17, 42]。

（2）无法整合手工编写的 RTL 和供应商 IP，尽管实际系统中的 HLS 设计通常都包含这些组件。

（3）仅限于特定 FPGA 器件，如 AMD Alveo U250 和 U280，难以适配满足特定计算或预算需求的硬件。

（4）缺乏可扩展的基础设施，难以便捷地探索分区和流水方案的不同研究方向。

我们提出 RapidStream IR，一套面向 FPGA HLPS 的组合式表示与探索基础设施。它支持混合语言的 FPGA HLS 设计和可定

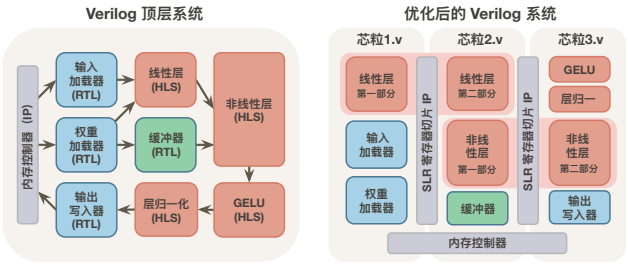

图 1: 大语言模型（LLM）FPGA HLS 加速器设计 [8] 在物理优化前后的对比。

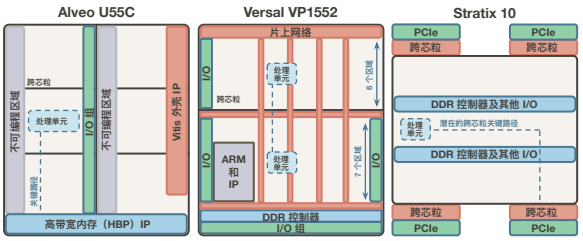

图 2: FPGA 器件的布局。芯粒边界引入显著延迟，间隙区域和 IP 限制了资源利用率。这些限制在 HLS 中未被考虑。

制的 FPGA 器件，并对其进行优化以提升工作频率。该基础设施通过以下关键特性解决了上述技术挑战：

中间表示（IR）——我们的方案提供了灵活且可扩展的设计 IR，可使用任意编程语言进行变换。该 IR 能够有效捕获整个层次结构中的连接关系、组件的流水化能力以及设计的空间信息。RapidStream IR 确保设计功能在 IR 上的变换或操作过程中保持不变。

可复用的设计优化变换——RapidStream IR 提供了一组可复用的变换，用于设计转换，包括层次重建、模块分区和模块插入等。利用这些变换，研究者可以探索不同的优化策略，并轻松针对特定设计目标定制框架。

支持多种设计格式——我们提供了针对不同设计格式的分析器，如 Verilog、Xilinx 编译 IP（XCI）和 Vitis HLS 生成的设计。该框架还可扩展支持其他源格式，例如 Dynamatic HLS [22, 23] 和 Catapult HLS [37]，用户只需编写一个简单的信息提取器即可。

跨平台可移植性——RapidStream IR 通过提供定义新器件的接口确保在不同 FPGA 平台间的可移植性，无需修改分析器或优化变换。

以图 1 中的大语言模型（LLM）FPGA 加速器 [8] 为例。该设计整合了多种源格式，包括 HLS、RTL 和 Xilinx IP 模块，并通过 Verilog 进行层次化连接。具体而言，“输入加载器”和“缓冲器”采用手写 RTL 实现，计算内核由 HLS 生成，并通过 Xilinx IP 模块与外部存储器交互。

该设计最初在 Xilinx Alveo U280 FPGA 上以 150 MHz 运行。Chen 等人 [8] 通过手动将模块分配到不同 FPGA 芯粒并添加寄存器以打断关键路径（如图 1 所示），将频率提升至 245 MHz。该方法涉及将模块（如线性层）分区到多个 FPGA 区域，这增加了代码管理的复杂性。由于缺乏支持混合语言集成的 HLPS 框架，自动化这一耗时且易出错的过程十分困难。此外，将设计适配到新的或定制的硬件需要对 RTL 层次结构和布局约束进行大量修改以获得最优性能。

RapidStream IR 通过组合多种设计源格式并探索分区与流水策略，实现了 LLM FPGA 设计优化的自动化。使用 RapidStream IR，在 U280 FPGA 上无需修改代码即可达到与原工作手动优化结果相符的 243 MHz 频率，并可无缝移植到其他 FPGA 平台。在六种 FPGA 器件上的评估显示，频率提升范围为 30% 至 62%，平均工作频率达到 244 MHz。

本文的技术贡献如下：

(1) 我们提出了 RapidStream IR，这是首个支持从多种来源（如 HLS 生成的模块、RTL 和供应商 IP）层次化组合 FPGA 设计的 HLPS 基础设施。该框架支持在复杂 FPGA 设计中探索物理优化，旨在提升工作频率的同时保持设计开发效率。
(2) 我们引入了面向 HLPS 的灵活且可扩展的 IR。该方法允许创建可复用的变换以适应各种设计格式和目标器件；为实现这一目标，用户只需编写一个简单的信息提取器。
(3) 通过案例研究——包括布局规划探索、并行综合和设计调试——我们展示了框架的可扩展性。这些研究突显了 RapidStream IR 促进研究与探索的能力，并提出了框架的未来改进和应用方向。

RapidStream IR 支持对层次化组合的 HLS 设计进行全局时序优化探索，并能适配 Versal VP1552 等新平台。实验结果表明，在已有研究和新引入的 FPGA 平台上，平均频率提升达 40%，处于领先水平。

2 背景

2.1 大规模 FPGA 设计面临的挑战

FPGA HLS 设计的优化之所以困难，在于不同器件乃至同一器件内部的资源分布和连线延迟都存在明显的架构差异。因此，设计者在时序优化和设计可移植性方面都会面临困难。图 2 以三种代表性器件为例说明了这一问题 [1, 17]：

(1) AMD Alveo U55C FPGA 拥有三个芯粒，每个芯粒的部分资源专用于 Vitis 外壳。其底部连接着 32 个高带宽内存（HBM）通道，中心存在不可编程的间隙区域。在 Vitis HLS 中，HBM 通道通过指针访问，既无法控制访问模块的放置位置，也无法控制通往 HBM 控制器的流水级数。
(2) AMD Versal VP1552 FPGA 由两个芯粒组成，高度分别为六个和七个区域。它集成了片上网络和 ARM 处理器。IP 导致的不连续性可能延长附近信号的布线路径，而跨芯粒连接引入的延迟明显更高，但在 HLS 中未被反映。
(3) Intel Stratix 10 FPGA 的 I/O 组位于可编程逻辑的中心，多芯粒互连桥和 PCIe 模块位于两侧，这些在 HLS 中均未建模。

2.2 高层次物理综合（HLPS）

高层次物理综合（HLPS）旨在弥合 HLS 与物理设计之间的鸿沟。通过向 HLS 提供 FPGA 的物理布局信息，可以更有效地对模块进行分区和布局规划，并识别需要插入流水级的长连线。图 3 以 LLM 加速器的前几个阶段为例说明了 HLPS 流程。该过程可概括为以下阶段：

(1) **通信分析**。对高层设计描述（如 C++ 代码）进行分析，以识别模块单元之间可容忍延迟的连接，如握手协议、“有效-就绪”协议和互连总线。这些通信在 C++ 中通常表示为流、数据流区域、全局指针和函数参数。

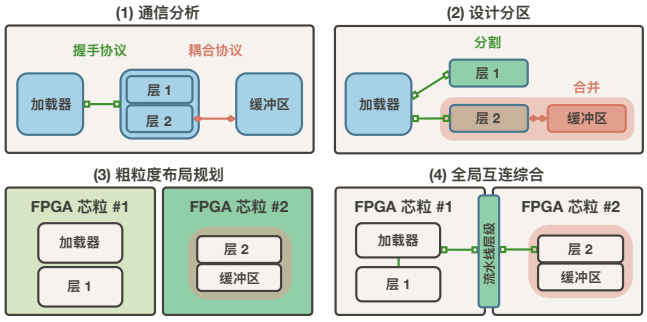

图 3: 针对 LLM 设计 [8] 前三个阶段的 HLPS 流程。

(2) **设计分区。**根据通信模式将设计划分为若干分区，仅允许分区间存在可容忍延迟的连接。这些分区可以分布在远距离区域或不同芯粒上，分区间的连接可以被流水化，从而打断全局关键路径。
(3) **粗粒度布局规划。**将分区分配到 FPGA 上的粗粒度区域，优化多个目标，如最小化跨区域的布线连接数、管理可用资源有限的区域，以及均衡资源分布以防止局部布线拥塞。
(4) **全局互连综合。**确定每个分区的位置后，根据估计的延迟在分区间建立互连以打断关键路径，实现时序收敛。

多项研究 [12, 16, 17, 25, 30, 32, 33] 探讨了 HLPS 的方法论，并证明了其在自动优化 HLS 设计频率方面的有效性。

2.3 示例分析

HLPS 的实际应用仍受制于以下几方面的不足：（1）跨层次层级的全局优化，（2）多种源格式的集成，以及（3）对新器件的适配。这些基础设施层面的缺失不仅限制了 HLPS 的落地，也阻碍了后续研究的推进，因为研究者必须为每个新设计或器件开发不可复用的定制工具。

HLPS 基础设施的需求可通过图 4a 所示的 LLM FPGA 设计 [8] 来说明。该图展示了第 1 节讨论的 LLM 加速器的一个简化片段，其中使用 Verilog 互连模块以集成 RTL、HLS 和 IP 组件。本文将反复引用该示例以突出挑战并讨论我们提出的解决方案。

在图 4a 中，该 LLM 加速器由三个模块组成：（1）一个用 Verilog 编写的“输入加载器”，（2）一个用于数据缓冲的 Verilog FIFO，以及（3）一个由前两个线性 LLM 层组成的层次化 HLS 内核。这两个层彼此协同，并被组织为一个 HLS 函数，由其调用两个子函数完成相应计算。顶层互连采用 Verilog 编写，不仅用于实例化各个子模块，还使用 `assign` 和 `always` 语句来整合控制逻辑。

现有 HLPS 方法不足以优化该 LLM 设计，因为它们缺乏对“输入加载器”和 FIFO 等 RTL 组件的支持。此外，它们无法解析顶层 Verilog 逻辑，阻碍了通信分析（例如通过 `assign` 语句建立的连接）。即使支持 RTL，结果仍然不理想。如图 4b 所示，由于缺乏层次化流水化，资源密集的“层 1”和“层 2”被迫放置在同一芯粒上，导致局部拥塞。此外，由于无法理解 Verilog 代码，顶层控制逻辑被视为一个单体模块，导致与各层的连接无法流水化，形成全局关键路径。

RapidStream IR 通过先将设计转换为等效的中间表示（IR）来解决这些挑战，如图 4c 所示。随后，再通过一系列变换将该 IR 优化为图 4d 所示的形式。RapidStream IR 在以下三个方面增强了 HLPS：**（1）层次化优化**，允许“层 1”和“层 2”独立放置并分别流水化，以均衡资源需求；**（2）格式无关的分析**，支持“输入加载器”和 FIFO 的集成，并允许将 Verilog 控制逻辑划分为两个独立单元“辅助 1”和“辅助 2”（将在 §3.1 中讨论）；以及 **（3）跨输入格式和 FPGA 平台的可移植性**，通过在 IR 中统一接口和流水化能力等信息实现。

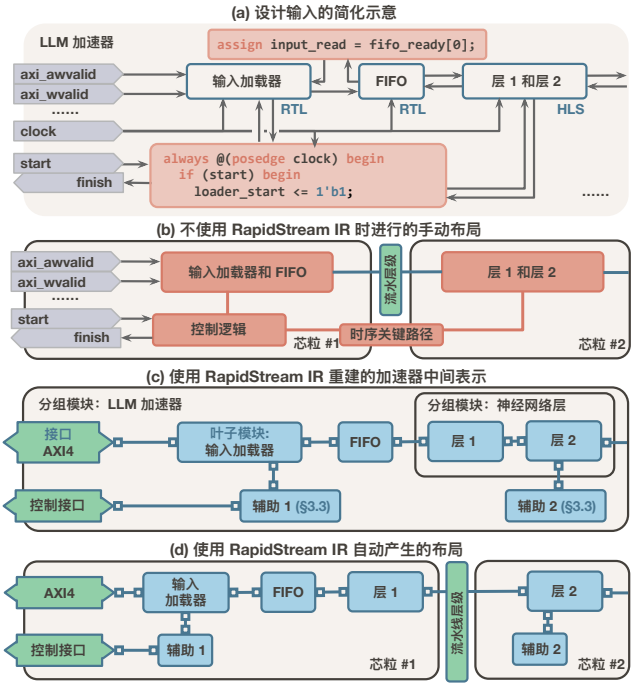

图 4: LLM 加速器 [8] 的前三个模块及使用或不使用 RapidStream IR 时的优化效果。

3 RAPIDSTREAM IR 框架

RapidStream IR 由三个组件构成：（1）一个逐步细化的**中间表示（IR）**，对特定 HLS 框架、EDA 工具或编码风格保持无关性；（2）用于辅助设计规格输入和 EDA 工具输出的**实用插件**；以及（3）一组可复用的**变换**，用于组合设计优化。每个要素在增强 HLPS 流程中发挥关键作用。若没有 IR，HLPS 研究者将不得不分析 Verilog 代码以确定模块通信并直接在代码中插入流水线。若没有实用插件，研究者必须手动调用 EDA 工具进行资源分析和设计约束设置。

总体架构。RapidStream IR 的整体架构如图 5 所示。它接受三类**输入**：FPGA 设计（如 Verilog、网表）、高层接口信息（如 HLS 报告、编译指示）以及用于器件信息和 EDA 工具交互的 Python 指令。这些输入由**插件**处理后导入 *IR*。变换 *pass* 随后修改 IR 以执行 HLPS 流程。最终 IR 由**插件**转换回设计代码和布局提示，供 EDA 工具实现。用户也可以编写工具修改输出 IR 以进行定制。将这些集成后，我们在第 3.4 节实现了一个完整的 HLPS 系统，用于优化实际设计。

设计原则。我们遵循以下设计理念：

(1) **支持增量式分析与变换。**我们特意将 IR 设计为轻量级且健壮的。通过提供一组可组合的核心变换，将复杂设计简化为规范形式并逐步获取所需信息，使变换和插件保持简洁。

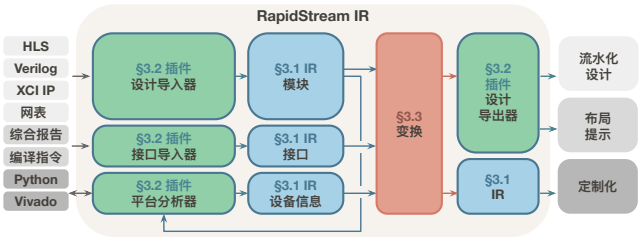

图 5: RapidStream IR 的整体架构，由 IR（蓝色，§3.1）、插件（绿色，§3.2）和变换（红色，§3.3）组成。

(2) **灵活的适用范围。**我们聚焦于 HLPS 方法论的实际需求，而非创建类似 MLIR [26] 的全能解决方案。这种方法使我们能够优先处理粗粒度模块交互，同时维护对无法轻易转换为 IR 的细粒度逻辑的支持，例如 Xilinx 供应商 IP [3] 和设计网表。

(3) **无语言"锁定"。**并非所有计算都值得付出 C++ 的开发开销。为支持所有主流语言，我们将 IR 设计为尽可能简单的 JSON Schema [24] 子集，并提供自动化语言绑定生成器。事实上，我们使用 Rust、Python 和 Java 编写了许多变换，并使用 TypeScript 开发了可视化和调试工具。

此外，我们提供"设计规则检查（DRC）"变换以确保设计信息的一致性。我们还在整个优化过程中维护原始设计组件与其变换后对应物之间的映射关系，确保人类可读性和可调试性。

3.1 逐步细化的 IR

RapidStream IR 是一种逐步纳入设计粗粒度信息的 IR。每个变换逐步推断设计的层次结构、连接和端口接口属性。如果细粒度逻辑在变换中未被使用，则保持其原始形态不变。

设计元素。RapidStream IR 捕获设计的以下要素：

(1) **模块。**一个设计实体，分为**分组模块**和**叶子模块**。每个模块由名称标识，包含多个端口，每个端口具有方向和位宽属性，与其他模块互连。模块可以包含**接口**，用于标识端口的潜在流水方法。

(2) **叶子模块。**HLPS 将其视为原子单元的基本设计单元，保持其完整不变。叶子模块可以采用任意格式，如 RTL 或 IP，只要后续 EDA 工具支持即可。RapidStream IR 提供多种实用插件来获取叶子模块所需的属性。叶子模块可以通过 RapidStream IR 的变换被逐步重构为分组模块或分区为多个叶子模块。

(3) **分组模块。**从叶子模块重构出的层次结构，用于组织子模块。分组模块仅作为容器，不添加逻辑，这意味着每个子模块连接必须通过单一标识符实现。RapidStream IR 变换在遵循此规则的前提下逐步分区其子模块。

(4) **接口。**一种可应用于一组端口的流水策略。接口的**类型**指导流水策略，如握手或前馈。当端口被纳入接口时，可以通过引入额外的流水级来实现流水化。例如，仅携带标量信号的**前馈**接口可以通过插入触发器来打断关键路径。涉及有效、就绪和数据端口的**握手**接口可以通过添加中继站 [6] 或近满 FIFO [18] 来流水化。图 6 说明了这两种最常见的接口及其流水方法。

(5) **附加元数据。**IR 可以包含额外数据，如布局约束、资源利用率和时序特性，附加到任意 IR 节点作为附加字段，并由分析变换根据需要逐步推断和更新。

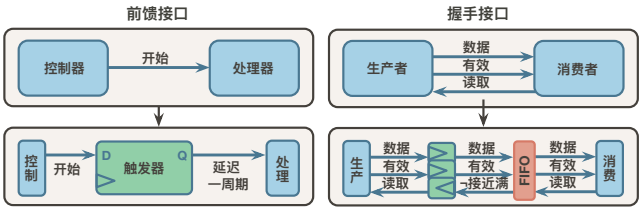

图 6: 前馈接口使用触发器寄存器进行流水化，握手接口使用近满 FIFO 和寄存器进行流水化。接近满表示 FIFO 接近满状态，防止因触发器延迟导致溢出。

不变假设。在变换过程中，IR 维护若干假设：（1）分组模块中的每条线必须精确连接两个模块，禁止扇出。（2）分组模块中每个子模块端口只能连接到一个标识符或常量，不允许拼接或位选择等操作。（3）接口上的所有非常量端口应完全连接到另一个模块，不允许信号的拆分或遗漏。这些限制保持了 IR 的简洁性和易操作性。尽管有此限制，我们的核心变换能够将复杂设计转换为此形式。

虚拟器件定义。RapidStream IR 通过存储在 IR 中的虚拟器件描述支持多种 FPGA 器件，其中包含器件内的资源分布和跨芯粒线数量。虚拟器件描述将物理 FPGA 器件划分为槽位。在布局规划期间，设计模块被映射到这些槽位。RapidStream IR 包含基于经验数据的 UltraScale+ 和 Versal 预定义虚拟器件。用户也可以通过指定 FPGA 器件部件号和槽位形状等参数自定义虚拟器件。RapidStream IR 随后使用供应商工具提取必要的资源信息并自动生成虚拟器件描述。图 7 展示了使用我们的 Python API 为 Versal VP1552 描述的虚拟器件，该器件被划分为两列四行，每个槽位包含一个 FPGA 芯粒的四分之一，通过在 Vivado 中指定名为 `pblocks` 的布局规划矩形实现。

IR 格式示例。RapidStream IR 采用 JSON Schema [24] 可验证的格式进行结构化，包括字典、列表、字符串和数字等数据类型。IR 的存储和交换格式可根据所使用的编程语言选择性地采用 YAML [5]、JSON [7] 或 XML [38] 等。图 8 展示了第 2.3 节讨论的 LLM 加速器 IR 的一个片段，以 YAML 格式呈现以便阅读，并附有对应的模块图。顶层分组模块 `LLM`（第 1–12 行）实例化三个子模块：`InputLoader`、`FIFO` 和 `Layers`（第 5–12 行），它们通过握手接口互连。`InputLoader` 从内存读取文本输入，`FIFO` 缓冲数据，`Layers` 对缓冲输入执行线性层计算。IR 捕获粗粒度信息，如模块名称（第 1、14 行）、端口（第 2–3、15–19 行）和线（第 4 行）。`FIFO` 模块的实例化记为 `FIFO_inst`（第 9–11 行），将 `I_wire` 连接到其 `I` 端口（第 11 行）。在叶子模块 `FIFO` 内部，IR 保留其原始形式，如 Verilog 源代码（第 20 行）。流水线细节在接口部分（第 21–24 行）中指定，定义了握手接口及其关联端口。每个对象可选择性地包含特定于不同变换的附加元数据，如资源利用率和布局约束（第 25–27 行）。器件信息可嵌入 IR 以辅助针对特定 FPGA 目标优化设计的变换或生成 EDA 工具约束。

对比。RapidStream IR 聚焦于对**已有**各种格式设计的 HLPS，不同于 Xilinx IP Integrator（IPI）将 IP 组装成系统。RapidStream IR 提供灵活的表示，通过变换支持增量式分析。例如，所有模块最初被视为不可分割的叶子模块，按需分区。因此，大型 HLS 生成的模块可以在 RapidStream IR 中被分区，而 IPI 将其视为整体。

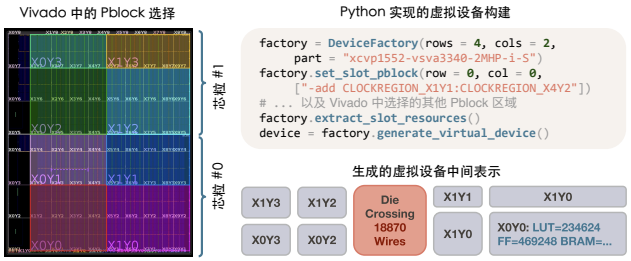

图 7: VP1552 的布局块（Pblocks）；Python 中的虚拟器件描述；推断的资源和跨芯粒线容量。

```
- module_name:       LLM
  module_ports:
  - { name: ap_clk, direction:   in,  width:      1 } # ..
  module_wires:      [{ name: I_wire,  width:     64 },  ..]
  module_submodules:
  - instance_name:  InputLoader_inst
    module_name:    InputLoader
    connections:    [{ port:      I,  value: I_wire },  ..]
  - instance_name:  FIFO_inst
    module_name:    FIFO
    connections:    [{ port:      I,  value: I_wire },  ..]
  - instance_name:  Layers_inst                       # ..

- module_name:       FIFO
  module_ports:
  - { name: I,       direction:    in, width:     64 }
  - { name: I_rdy,   direction:   out, width:      1 }
  - { name: I_vld,   direction:    in, width:      1 }
  - { name: ap_clk, direction:    in, width:      1 } # ..
  module_verilog:   "module FIFO (I, ..); ..; endmodule"
  module_interfaces:
  - iface_type:     handshake
    iface_ports:  { data:      [ I ], clk:   ap_clk ,
                    ready:     I_rdy, valid:  I_vld }
  module_metadata:
    resource:     { FF: 10, LUT: 39, DSP: 0, BRAM: 0,  ..}
    floorplan:      "SLOT_X1Y1"
```

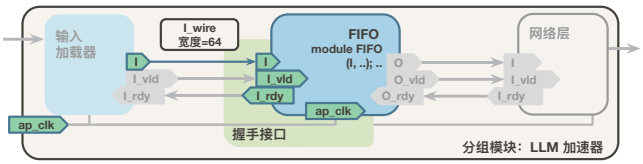

图 8: LLM 的部分 IR 及对应的模块图。

加速器描述语言如 Chisel [4] 和 Calyx [34] 支持细粒度硬件设计的高层规格。这些语言与 RapidStream IR 正交——RapidStream IR 专为捕获与 HLPS 相关的粗粒度信息而设计，包括模块层次、接口和资源利用率指标。鉴于 RapidStream IR 的语言无关设计，它可以直接将这些表示作为叶子模块纳入，并使用可复用变换对其进行转换。

另一方面，MLIR [26] 作为跨抽象层级的通用 IR，而 RapidStream IR 专为聚焦粗粒度优化的 HLPS 量身定制。RapidStream IR 允许叶子模块采用任意格式，而 MLIR 要求每个层级都有详细信息。尽管 RapidStream IR 可以作为 MLIR 的自定义方言来表示，但这样的表示要求变换使用 C++ 编写，而这正是 RapidStream IR 刻意规避的。此外，MLIR 缺乏面向 HLPS 的可复用变换，降低了在此领域使用它的动机。

```
module InputLoader (
  output wire m_axi_AWVALID,  input wire m_axi_AWREADY,
  output wire m_axi_WVALID,   input wire m_axi_WREADY,
  // ... 33 other AXI ports
);
  // pragma handshake pattern=m_axi_{bundle}{role} \
     role.valid=VALID role.ready=READY role.data=.*
endmodule
```

图 9: Verilog 中的接口编译指令（pragma），将带有 m_axi_ 前缀的端口映射为握手接口，并将具有相同前缀的端口绑定（如 m_axi_AW）。后缀 VALID 和 READY 指示端口角色，其他后缀表示数据。

3.2 实用插件

RapidStream IR 拥有一套实用插件，用于桥接抽象 IR 与具体实现。这些插件本质上是模块化的，可根据需要支持额外的源格式和 EDA 工具。插件分为**导入器**、**分析器**和**导出器**三类。

叶子模块导入器。从模块的源格式中提取元数据以构建 IR 中对应的叶子模块。解析的数据包括模块名称、端口等。为保持设计完整性，源代码或其二进制文件直接嵌入 IR。我们已开发了针对 Verilog、VHDL、网表和 Xilinx 编译 IP（XCI）等格式的导入器。例如，对于 Verilog，我们使用 Slang [35] 从语法树中提取模块信息。其他格式（如 VHDL）通过适当的解析器或使用 EDA 工具将模块签名转换为 Verilog 存根文件后使用 Verilog 导入器来处理。

接口导入器。HLPS 所需的高层接口信息可从多种来源提取。Vitis HLS 在报告文件中提供接口信息，而 Xilinx IP 在 XCI 文件中包含接口细节。如果接口数据缺失，用户可以通过**源代码注释中的编译指示**或使用我们 Python API 中基于正则表达式的**接口规则**来提供。图 9 展示了一个 Verilog 源代码示例，其中第 5 行的单行编译指示为手工编写的内存“输入加载器” RTL 模块的全部 37 个 AXI 端口设置了握手接口。

平台分析器。设计优化需要来自下游供应商工具的信息，如每个模块的资源利用率，以均衡资源在器件区域间的分配。平台分析器与供应商工具交互以收集数据。

设计导出器。设计导出器从 IR 生成最终设计输出，使其与下游 EDA 工具兼容。对于未修改的叶子模块，导出器原样输出原始源代码。对于已修改的模块，生成对应的 Verilog 文件。如果 IR 包含额外元数据（如布局规划指导），导出器还会将其输出为约束文件。

3.3 可组合的变换

RapidStream IR 提供一组变换，逐步收集数据并细化 IR 以优化设计。每个变换遵循“做好一件事”的原则，专注于一个方面以确保健壮性和可维护性，并便于扩展以支持新的设计格式和 EDA 工具。

本节中，我们以 LLM 加速器示例 [8] 的子集来展示 RapidStream IR 的核心变换及其功能。为清晰起见，我们使用图 10 中 IR 的模块图。所有设计中的模块最初均作为叶子模块导入。

层次重建变换。重建变换将导入的叶子模块转换为分组模块以重构设计层次。它创建一个分组模块，包含从叶子模块中提取的子模块及其残余逻辑，后者被定义为**辅助模块**。分组模块的端口与原始叶子模块相同，所有端口连接到辅助模块。在

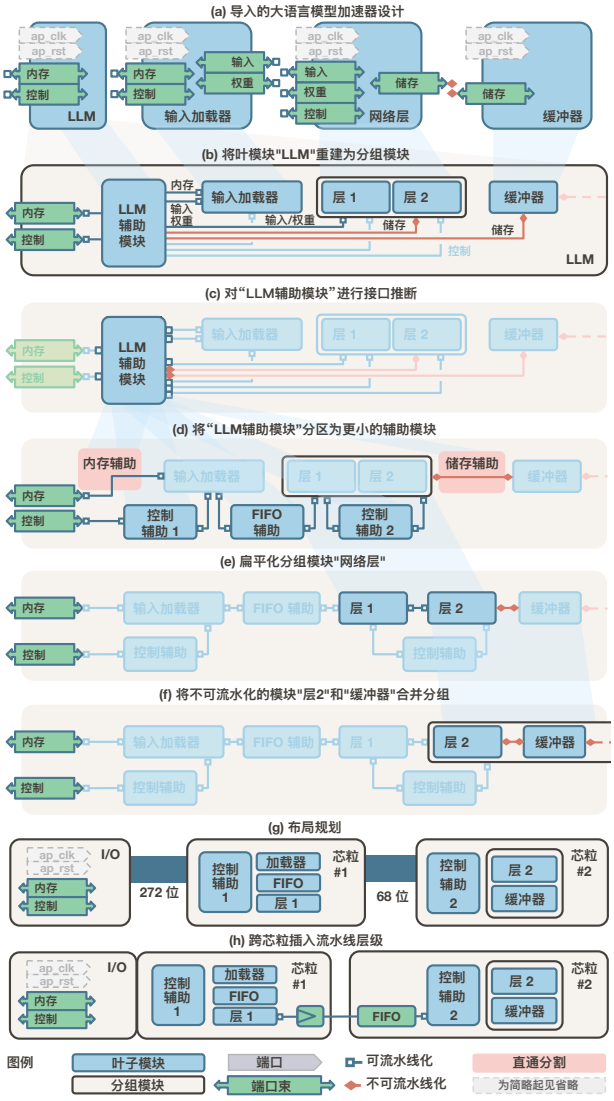

图 10: RapidStream IR 变换在 LLM 加速器示例上的应用。

此阶段，该变换**不分析**子模块间的互连，而是为原始子模块的每个端口在辅助模块上添加对应端口。

图 10b 展示了重建变换应用于 LLM 的过程，将其重构为包含子模块和辅助模块“ LLM 辅助模块”的分组模块。该辅助模块包含 LLM 的控制逻辑和互连。直接分析 LLM 的互连具有挑战性，因为其源格式的复杂性——包括 `always` 和 `generate` 等 Verilog 语法结构——需要完整的综合展开支持。为各种设计格式维护和更新此类展开工具将耗费大量人力。

该变换维护了 IR 的不变假设（§3.1）：

(1) 在新形成的分组模块中，每条线精确连接两个模块：辅助模块和一个子模块。
(2) (a) 提取的子模块端口连接到与辅助模块互连的线标识符。
(b) 辅助模块的每个端口通过线直接连接到一个提取的子模块或重构后分组模块上的端口；在任一情况下，它都是一个标识符。
(3) 子模块上的每个端口完整连接到辅助模块，确保接口不被拆分。

该实现对于**任意源格式**都简单直接，只要存在提供三种功能的重写器：（1）提取子模块名称和端口连接；（2）向模块添加新端口；（3）将表达式连接到这些新端口。

例如，Slang [35] 工具允许提取 Verilog 子模块信息；通过修改语法树添加新端口；通过追加新的 `assign` 语句重路由连接。此方法可适用于其他源格式。请注意，即使没有针对特定源格式的专用重写器，RapidStream IR 仍可将该格式的模块视为叶子模块进行管理；因此仍可在这些模块之间插入流水级或按需分区。

接口推断变换。当设计模块缺少流水线插入所需的显式接口信息时，该变换从其他模块推断接口。例如，用户可能拥有一个分组模块，其所有端口直接连接到子模块，但模块本身缺乏接口数据。通过利用这些子模块的接口细节，接口推断变换可以推导出父模块的接口。

接口信息不仅在父子模块之间传播，还在兄弟模块之间传播。具体而言，对于层次重建变换中创建的辅助模块，接口推断器通过从辅助模块的兄弟模块（即提取的子模块）传递信息来定义其接口，从而完成辅助模块的接口信息，如图 10c 所示。

分区变换。该变换将叶子模块分割为多个部分以进行独立布局规划。它与层次重建变换协同，有效充当通信分析变换的角色。在重建变换之后，它分割由重建变换创建的辅助模块以分散子模块通信。图 10d 展示了“ LLM 辅助模块”的分区过程——该模块最初连接所有子模块。分区后，它被分割为五个部分，包括内存连接（“内存辅助”和“储存辅助”）和控制逻辑（“控制辅助 1 ”和“控制辅助 2 ”）。在此示例中，LLM 模块在 Verilog 主体中实现了 FIFO 逻辑以连接输入加载器和层 1，该逻辑被分割为“ FIFO 辅助”。

它使用 EDA 流程将**任意格式**的模块转换为网表，并使用基于 RapidWright [27] 编写的并查集 [15] 分析端口连通性，排除时钟和复位信号（因为它们在子模块间共用）。不相连的组件被分离为新的部分。这些部分通过**封装**原始辅助模块来创建，仅暴露必要的端口并保留内部逻辑。未连接的逻辑保持悬空状态，将由后续 EDA 流程消除。新的部分在 IR 中替换原始辅助模块，时钟和复位信号通过专用广播辅助模块分配到所有子模块。

该变换通过将公共接口中的端口合并到同一并查集来维护 IR 假设，防止接口跨越多个部分。除了时钟和复位信号（由广播模块管理）外，它不引入新连接。这确保了变换后所有线仍精确连接两个模块，所有端口连接到标识符或常量。

直通变换。如果网表分析表明某接口仅直接连接到另一个接口，则可通过在两个接口间重路由连接来绕过该模块。在图 10d 中，“储存辅助”部分被绕过，使“层 2 ”和“缓冲器”模块直接连接。这简化了 IR 并减少了模块数量，使设计更具可读性且更易优化。直通变换通过先将线从一个模块断开再连接到另一个模块来维护 IR 假设。

扁平化变换。HLPS 优化建模方法，如 AutoBridge [17] 中使用的整数线性规划（ILP），通常需要扁平图而非多层次的超图。扁平化变换通过递归地将所有分组模块的子模块合并到单一层级，从而将层次化设计转换为扁平设计。在此过程中，线被整合，子模块及其连接在新模块中重新建立。图 10e 展示

了扁平化变换将“层 1 ”和“层 2 ”子模块纳入 LLM 模块。若无此变换，这两个模块将不得不被归入同一分区，由于它们都是资源密集型的，这将导致次优分区。

该变换通过不引入模块间新互连来遵守 IR 假设。它仅整合现有线和子模块，从而保持原始属性。

封装变换。该变换使用模板封装模块。在模板中，可以在被封装模块旁添加辅助子模块。封装器端口可连接到辅助模块或被封装模块。该变换可通过仅暴露特定端口来实现分区。它还可以将流水级作为辅助子模块添加。通常，随后执行扁平化变换以提升辅助模块层级，从而实质上在目标位置插入辅助模块。

分组变换。该变换将扁平设计重新组织为层次结构。在图 10f 中，子模块“层 2 ”和“缓冲器”被归入一个新的分组模块。该变换可用于合并不可流水化的模块和指定布局规划约束。

3.4 框架集成

我们在 RapidStream IR 中按照第 2.2 节描述的方法论开发了一个完整的 HLPS 系统，集成了我们的插件和变换以评估 RapidStream IR 的适用性。图 10 展示了该工具的工作流程，由 HLPS 方法论的四个阶段组成。

(1) **通信分析**：工具（a）将设计和接口数据导入 RapidStream IR，（b）使用层次重建变换将大模块重构为分组模块，（c）为辅助模块和缺乏接口信息的模块推断接口，（d）分区广播模块并应用直通变换，尤其针对辅助模块。此阶段捕获模块间的粗粒度通信模式。
(2) **设计分区**：工具（e）将设计转换为扁平表示，（f）将不可流水化的模块与相邻模块分组。此阶段根据识别的通信模式将设计分区为可流水化的部分。
(3) **粗粒度布局规划**：利用 AutoBridge 的整数线性规划（ILP）建模方法 [17]，工具（g）优化模块在虚拟器件预定义槽位上的放置并设计流水线插入方案。此阶段将分区分配到 FPGA 上的粗粒度区域，旨在最小化跨区域布线并满足 DSP 数量和跨边界线数等约束。
(4) **全局互连综合**：布局规划完成后，工具使用分组变换将同一区域的模块聚类。然后（h）使用封装变换插入流水级。此阶段根据估计的延迟生成分区间连接以打断关键路径并辅助时序收敛。最终，优化后的设计被导出用于实现。

综上所述，我们将 AutoBridge 的建模方法 [17] 集成到 RapidStream IR 基础设施中，将其增强为面向新探索策略以及不同源格式和器件组合的灵活、模块化物理综合工具。第 4 节展示了框架的适应性，并评估了 FPGA 设计和器件上的频率提升，将结果与原始供应商工具和手动优化进行了比较。

4 评估

我们在 AMD FPGA 平台上使用 Vivado 2023.2 对 RapidStream IR 进行了评估。实验运行于配备 AMD EPYC 7282 CPU、128 GB 内存和 Ubuntu 22.04 的环境中。我们使用 COIN-OR 求解器 [36]，并将 ILP 优化任务的时间限制设为 400 秒。为测试多种设计格式，我们使用 Dynamatic 2.0、Catapult HLS 2021.1 和 Intel FPGA HLS 19.4.0 生成 RTL 输入。我们旨在回答以下研究问题：

研究问题 1 RapidStream IR 的变换能否用于转换各种输入格式的 FPGA 设计，包括手工编写的 Verilog 和不同供应商工具生成的 HLS 设计？

```
add_reset(module=".*", port="rst|reset", active="high")
add_handshake(module=top_level, pattern="{bundle}_{role}",
      role={ready:"ready", valid:"valid", data:"in|out"})
```

图 11: Dynamatic 接口规则的代码片段。

表 1: 支持各 HLS 工具所需的 Python 或 Verilog 代码行数。

软件	Dynamatic	Catapult HLS	Intel HLS
代码行数	146	158	204

研究问题 2 RapidStream IR 能否有效降低程序员实现新的研究探索功能所需的工作量？

研究问题 3 RapidStream IR 能否为复杂 FPGA 设计和新目标器件提供频率提升？

4.1 自定义 HLS 输入

Dynamatic 是一个开源编译器，可将 C++ 代码转换为采用握手协议进行动态调度的 VHDL 设计 [22, 23]。Catapult HLS [37] 和 Intel HLS [21] 等商业工具也通过自定义数据类型创建握手接口。先前的工作缺乏操作这类生成 RTL 设计所需的基础设施。要为 RapidStream IR 开发能够接收这些生成设计的新前端，需要三个组件：（1）元数据解析器，（2）接口分析器，以及（3）代码重写器。

元数据解析器和代码重写器对于使用标准硬件描述语言（如 Verilog 或 VHDL）的大多数输入都是通用的。然而，接口分析器则需要针对每个 HLS 框架分别编写。本小节聚焦于接口分析器。

Dynamatic 的弹性元件命名一致，与**接口规则**（§3.2）高度契合。我们使用 20 条 Python 规则来指定其**全部**握手接口。图 11 展示了其中两条：一条使用正则表达式 `".*"` 匹配并应用于所有模块以指定复位信号，另一条定义顶层模块的握手接口。**Catapult HLS** 使用可定制的设计库（如 `ccs_out_wait` 和 `ccs_in_wait`）综合握手接口；通过在这些模块的 Verilog 代码中添加简单的编译指示，接口信息可在接口推断变换期间自动传播到相邻模块。**Intel HLS** 创建的握手接口端口命名大多一致，因此也兼容基于 Python 的接口规则方法。

表 1 给出了为支持这些 HLS 工具输入所需的 Python 或 Verilog 代码总行数。我们使用来自三个来源的基准测试进行了实验：Dynamatic 仓库 [14] 中的全部 29 个示例、Catapult HLS 的一个稀疏线性代数加速器 [13]，以及 Intel HLS 的 CHStone 测试套件中的全部 12 个基准 [11]。我们的方法成功提取了接口信息。此外，我们的解析器和代码重写器还有效地将所有基准设计导入 RapidStream IR，完成层次变换、插入流水级，并导出功能等价的 RTL 设计。

小结 1（研究问题 1：输入格式）

通过扩展 RapidStream IR 以支持来自多种 HLS 工具的 RTL，我们展示了其处理多种输入格式的能力。

4.2 布局规划探索

布局规划需要在局部和全局优化目标之间进行权衡。如图 12 所示，LLM 示例设计 [8] 存在十种不同的布局方案。图中的折线图表明，减少布局中最拥塞区域的逻辑量可以降低局部拥

表 2: RapidStream IR 在不同 FPGA 上针对多种设计格式实现的自动化频率优化效果。

应用	目标器件	基准特性			LUT	FF	BRAM	DSP	URAM	频率 (MHz)		
		层次化	混合语言	新器件	(%)	(%)	(%)	(%)	(%)	原始	RapidStream IR	其他工作
CNN 13×4	U250				13	11	10	17	0	233	335 (+44%)	325 [17]
CNN 13×6	U250				15	16	13	26	0	234	327 (+40%)	324 [17]
CNN 13×8	U250				26	22	16	24	0	245	332 (+36%)	320 [17]
CNN 13×10	U250				30	27	28	43	0	-	320 (+∞%)	322 [17]
CNN 13×12	U250				27	33	30	51	0	-	305 (+∞%)	295 [17]
LLaMA2	VP1552	✓	✓	✓	32	16	13	22	18	198	258 (+30%)	N/A
LLaMA2	VHK158	✓	✓	✓	32	16	13	22	18	206	273 (+33%)	N/A
LLaMA2	U55C	✓	✓	✓	49	25	24	18	24	165	247 (+50%)	N/A
LLaMA2	VU9P	✓	✓		59	32	23	24	24	141	212 (+50%)	N/A
LLaMA2	U250	✓	✓		42	23	20	14	19	159	228 (+43%)	N/A
LLaMA2	U280	✓	✓		49	25	24	18	25	150	243 (+62%)	245 [8]
LLaMA2 (opt)	U280	✓	✓		35	19	15	18	25	201	306 (+52%)	245 [8]
Minimap2	VP1552	✓		✓	39	15	10	31	0	265	285 (+8%)	N/A
KNN	U280		✓		56	28	10	14	0	-	292 (+∞%)	N/A
将不可布线设计的频率记为零后计算的平均值					36	22	18	24	11	157	283 (+80%)	
排除原始不可布线设计后计算的平均值					36	20	16	21	14	200	277 (+39%)	

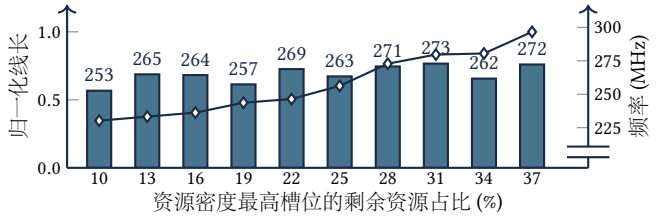

图 12: LLM 设计在 VHK158 上的资源分布、线长与频率之间的关系。

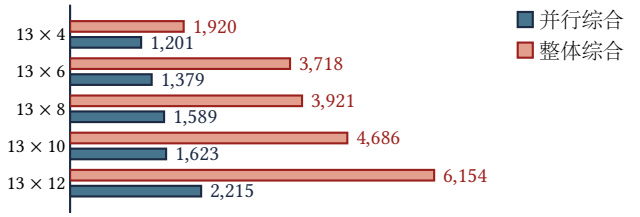

图 13: 综合实际耗时（秒）。

塞，但也可能导致线长增加，从而对全局布线结果产生不利影响，反之亦然。同一图中的柱状图进一步突显了这种权衡的复杂性，表明不同的局部/全局优化权衡点会带来最高达 20 MHz 的工作频率差异。

若没有 RapidStream IR，设计者需要手动探索设计空间——分区设计、重构层次结构（如前述图 1 所示）、修改布局约束，然后重新执行综合及布局布线过程。

为评估 RapidStream IR 的适用性，我们将该方法应用于这一布局规划探索任务。通过调整第 3.1 节所述虚拟器件模型中每个槽位的最大允许资源利用率，RapidStream IR 便可在给定约束下优化布局中的线长。借此，RapidStream IR 能够自动探索这一权衡设计空间，并逼近帕累托最优。该方法生成了图 12 中呈现的多种布局方案，使设计者能够评估线长与资源分布之间的平衡。整个自动化过程被实现为独立的 RapidStream IR 插件，仅用 207 行 Python 代码编写，且可在不同设计之间复用。相比之下，仅对此设计进行手动探索就需要大量 RTL 代码重写（数百行），不仅容易引入错误，还需要多轮迭代。

小结 2（研究问题 2：扩展性）

RapidStream IR 简化了高层次物理优化的扩展，例如探索不同的布局规划方案。

4.3 并行综合

在第 3.4 节中，我们将设计划分为若干粗粒度组，每组对应一个器件槽位。该方法天然具备并行综合的潜力，即各槽位可以并行综合。顶层模块可以将这些槽位标记为黑盒后与之一同综合。最终，我们组装这些综合后的网表以获得完整设计。我们将并行综合程序实现为独立的 RapidStream IR 后端插件，仅用 299 行 Python 代码。

我们使用 AutoSA [40] 生成的卷积神经网络脉动阵列架构 HLS 基准来评估 RapidStream IR 的并行综合插件。评估在 Alveo U250 FPGA 上进行，通过并行综合各器件槽位实现，如图 13 所示。对于规模从 13 × 4 到 13 × 12 的脉动处理单元阵列，该插件实现了平均 2.49× 的综合加速。

小结 3（研究问题 2：扩展性）

RapidStream IR 支持便捷的设计变换，为基于分治策略的 EDA 工具研究开辟了可能性。

4.4 基准测试

我们使用在 RapidStream IR 中开发的 HLPS 工具，对一组未达时序目标的真实 FPGA 设计进行了评估。频率结果与 AutoBridge [17] 以及 FPGA 供应商 AMD Vivado 提供的标准 EDA 工具结果进行了比较。

(1) **CNN** 指使用 AutoSA [40] 通过 Vitis HLS 实现为脉动阵列架构的卷积神经网络加速器。它具有扁平层次结构，因此被 AutoBridge 所支持。我们使用这一基准来比较 RapidStream IR 与 AutoBridge 的频率结果。
(2) **LLaMA2** 指面向 LLaMA2 大语言模型推理的混合语言加速器，最初使用 HLS、Xilinx IP 和手写 RTL 在 Alveo U280 FPGA 上通过四级嵌套流水结构进行优化 [8, 9]。由于其层次化设计较为复杂，AutoBridge 不支持该设计。我们进一步使用 RapidStream IR 将其移植到 Versal 板卡，并将频率性能与 Vivado 进行比较，展示了 RapidStream IR 对多源多目标设计的适应性。
(3) **Minimap2** 指面向长读长基因组测序的加速器，具有多层次流水结构，最初使用 Vitis HLS 为 UltraScale+ VU9P 开发 [19]。在基准测试中，我们保留了 Minimap2 的原始层次结构，并将其移植到 AMD Versal VP1552 器件。
(4) **KNN** 指面向 Alveo U280 FPGA 的 k 近邻加速器，使用 HLS 内核和自定义 RTL 互连，在 Vitis 平台上实现 [29]。RapidStream IR 直接导入 Vitis 打包的 Xilinx Object（XO）文件进行优化，并以相同格式输出优化后的设计，充当 Vitis 框架的透明插件。

RapidStream IR 成功导入了所有设计，并按 HLPS 方法论应用了相应变换，如表 2 所示。表中的"基准特性"列给出了各设计的关键特征，包括多级可流水化层次结构（"层次化"）、混合语言格式（"混合语言"，包括 RTL、HLS 和 IP）以及面向新 FPGA 器件的实现目标（"新器件"）。这些都是现有 HLPS 框架难以支持的特性。在"频率"列中，"原始"列给出使用 Vivado 实现时的原始设计频率，" RapidStream IR "列展示经 RapidStream IR 优化后的设计频率，"其他工作"列则列出现有文献 [8, 17] 中的结果。对于原始设计因不可布线而失败的基准，其频率结果标记为"-"。在 FPGA 资源方面，我们报告目标器件上的原始利用率百分比。所有基准测试在优化后的资源变化均控制在 1% 以内。

小结 4（研究问题 1：输入格式）

RapidStream IR 有效重写了来自多种格式的 FPGA 设计，包括手工编写的 RTL、集成 IP 的项目以及具有多级可流水化层次结构的复杂设计。

表 2 还将我们的工作与 AutoBridge [17] 和手动优化 [8] 进行了比较（最后一列）。在" *CNN* "设计上，RapidStream IR 在 Alveo U250 上取得了与 AutoBridge 相当的频率提升；在将" *LLaMA2* "移植到不同器件时，则实现了 30% 至 62% 的提升。我们在 U280 上得到的" *LLaMA2* "频率与手动优化结果相当。进一步将" *LLaMA2* "重构为" *LLaMA2 (opt)* "后，通过将 HLS 函数拆分为更小、可流水化的部分，频率进一步提升至 306 MHz。

小结 5（研究问题 3：频率提升）

RapidStream IR 展示了与最先进方案相比领先的频率提升，同时支持更广泛的输入格式和目标 FPGA 器件。

5 相关工作

已有大量文献探讨 HLPS 方法论 [12, 16, 17, 25, 30, 32, 33, 42, 43]。AutoBridge [17] 通过在 HLS 阶段考虑布局信息来改善时序，从而构建跨多芯粒的高频 FPGA HLS 设计。然而，它仅支持对顶层函数中以数据流方式连接的流式端口进行流水化。相比之下，RapidStream IR 支持在混合语言设计的任意层次层级上进行流水化。第 4 节表明，AutoBridge 的方法论可以作为变换集成到我们的框架中，并借助我们对多级流水结构的解析能力进一步扩展其功能，而不会损失性能。其他现有的 HLPS 工作同样可以集成到我们的框架中，从而获得对更广泛输入格式和目标器件的支持。

尽管 HLPS 已取得显著进展，但仍缺乏可复用的 HLPS IR。现有的 IR 或语言，如 MLIR [26]、CIRCT [28]、Yosys IR [41]、ScaleHLS [44]、Chisel [4]、Calyx [34]、CIRRF [39]、HIR [31]、Allo [9] 和 Xilinx IPI [2]，分别处理编译、电路逻辑、HLS、数据通路、调度和 IP 集成等问题。然而，它们尚未提供支撑现有设计并对粗粒度分区进行流水化所需的基础设施，而这正是 HLPS 面临的独特挑战。这些框架与 RapidStream IR 彼此正交，可以作为叶子模块集成到 RapidStream IR 中，从而兼具双方优势。第 3.1 节对这一比较作了进一步讨论。

6 讨论与结论

第 4 节讨论的所有案例研究都以独立插件或 Python 脚本的形式实现，无需修改核心基础设施。我们预计 RIR 还可支持更广泛的应用，并提出以下潜在的未来研究方向：

(1) **自动化片上网络综合**：RapidStream IR 可通过在握手模块之间自动集成路由器，来实现 FPGA HLS 设计的片上网络（NoC）综合。挑战包括分析节点内部模式以确定带宽需求、进行流量管理，以及优化网络以尽量减小对吞吐量的影响。
(2) **并行布局布线**：RapidStream [20] 为使用 Vitis HLS 的扁平数据流 FPGA HLS 设计实现了并行布局布线。RapidStream IR 提供了层次扁平化和重组能力，可适配多层次结构以及多种 HLS 工具，因此有望进一步增强 RapidStream。挑战包括与供应商工具的接口集成，以及基于 RapidWright [27] 开发自定义布局布线器。
(3) **设计插装**：RapidStream IR 可扩展为在模块之间自动插入性能计数器和监控 IP，并基于接口信息确定其放置位置。这将有助于进行板上性能分析、定位性能瓶颈，以及分析片上网络的带宽需求等行为。它还可通过对已识别的握手接口施加随机节流，来支持模糊验证。

RapidStream IR 是一套面向 FPGA 高层次物理综合（HLPS）工具开发的通用基础设施。它具备逐步细化的中间表示、支持供应商工具集成的插件以及可复用的变换。RapidStream IR 在多种基准和平台上将 FPGA 设计频率提升了 30% 至 62%。此外，部分最初无法布线的设计在经过 RapidStream IR 的 HLPS 优化后能够达到约 300 MHz 的频率。RapidStream IR 具有良好的研究扩展能力，有望推动 FPGA EDA 社区的后续创新研究。

致 谢

感谢 AMD 的 Chris Lavin、Eddie Hung、Pongstorn Maidee、DJ Wang 和 Luciano Lavagno 提供的技术见解，感谢 Hongzheng Chen 在 LLM 加速器方面的帮助，以及 Dynamatic HLS 团队在基准测试和代码生成方面的支持。Jason Cong 和 Zhiru Zhang 在 RapidStream 的技术顾问委员会任职。

参考文献

[1] Advanced Micro Devices, Inc. 2024. *Versal* 自适应 *SoC* 技术参考手册（*Versal Adaptive SoC Technical Reference Manual*）. https://docs.amd.com/r/en-US/am011-versal-acap-trm

[2] Advanced Micro Devices, Inc. 2024. *Vivado* 设计套件用户指南：使用 *IP* 进行设计（*Vivado Design Suite User Guide: Designing with IP*）. https://docs.amd.com/r/en-US/ug896-vivado-ip/IP-Integrator

[3] Advanced Micro Devices, Inc. 2024. *Xilinx* 知识产权（*Xilinx Intellectual Property*）. https://www.xilinx.com/products/intellectual-property.html

[4] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. 2012. Chisel：在 Scala 嵌入式语言中构建硬件（Chisel: constructing hardware in a Scala embedded language）. In *Proceedings of the 49th Annual Design Automation Conference (DAC '12)*.

[5] Oren Ben-Kiki, Clark Evans, and Brian Ingerson. 2009. Yaml ain ' t markup language (yaml™) version 1.1. *Working Draft 2008* 5, 11 (2009).

[6] Julien Boucaron, Anthony Coadou, and Robert de Simone. 2009. 延迟不敏感设计：重试中继站与融合外壳（Latency-Insensitive Design: Retry Relay-Station and Fusion Shell）. In *Proceedings of the 4th International Workshop on the Application of Formal Methods for Globally Asynchronous and Locally Synchronous Design (FMGALS '09)*, Vol. 245. 23–33.

[7] Pierre Bourhis, Juan L. Reutter, Fernando Suárez, and Domagoj Vrgoč. 2017. JSON：数据模型、查询语言与模式规范（JSON: Data model, Query languages and Schema specification）*(PODS '17)*.

[8] Hongzheng Chen, Jiahao Zhang, Yixiao Du, Shaojie Xiang, Zichao Yue, Niansong Zhang, Yaohui Cai, and Zhiru Zhang. 2024. 理解基于 FPGA 空间加速的大语言模型推理潜力（Understanding the Potential of FPGA-Based Spatial Acceleration for Large Language Model Inference）. *ACM Transactions on Reconfigurable Technology and Systems* (Apr 2024).

[9] Hongzheng Chen, Niansong Zhang, Shaojie Xiang, Zhichen Zeng, Mengjia Dai, and Zhiru Zhang. 2024. Allo：面向可组合加速器设计的编程模型（Allo: A Programming Model for Composable Accelerator Design）. In *Proceedings of the ACM on Programming Languages (PLDI '24)*.

[10] Jason Cong, Jason Lau, Gai Liu, Stephen Neuendorffer, Peichen Pan, Kees Vissers, and Zhiru Zhang. 2022. 当今 FPGA 高层次综合：成就、挑战与机遇（FPGA HLS Today: Successes, Challenges, and Opportunities）. *ACM Transactions on Reconfigurable Technology and Systems* 15, 4, Article 51 (Aug 2022), 42 pages.

[11] Steve Dai, Gai Liu, and Zhiru Zhang. 2018. 基于联合 SDC 与 SAT 建模的精确资源约束调度可扩展方法（A Scalable Approach to Exact Resource-Constrained Scheduling Based on a Joint SDC and SAT Formulation）. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '18)*.

[12] Linfeng Du, Tingyuan Liang, Sharad Sinha, Zhiyao Xie, and Wei Zhang. 2023. FADO：面向多芯粒 FPGA 高层次综合设计的布局感知指令优化（FADO: Floorplan-Aware Directive Optimization for High-Level Synthesis Designs on Multi-Die FPGAs）. In *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '23)*.

[13] Yixiao Du. 2024. 康奈尔大学：使用 HLS 构建稀疏线性代数加速器（Cornell University: Building Sparse Linear Algebra Accelerators with HLS）. Webinar. https://webinars.sw.siemens.com/en-US/cornell-university-building-sparse-linear-algebra-accelerators-with-hls/

[14] EPFL Processor Architecture Laboratory. 2024. 基于 MLIR 的 DHLS（动态高层次综合）编译器（DHLS (Dynamic High-Level Synthesis) Compiler Based on MLIR）. https://github.com/EPFL-LAP/dynamatic/

[15] Bernard A. Galler and Michael J. Fisher. 1964. 一种改进的等价算法（An improved equivalence algorithm）. *Commun. ACM* 7, 5 (May 1964), 301–303.

[16] Licheng Guo, Yuze Chi, Jason Lau, Linghao Song, Xingyu Tian, Moazin Khatti, Weikang Qiao, Jie Wang, Ecenur Ustun, Zhenman Fang, Zhiru Zhang, and Jason Cong. 2023. TAPA：面向现代 FPGA 的可扩展任务并行数据流及高层次综合与物理结构协同优化的编程框架（TAPA: A Scalable Task-parallel Dataflow Programming Framework for Modern FPGAs with Co-optimization of HLS and Physical Design）. *ACM Transactions on Reconfigurable Technology and Systems*, Article 63 (Dec 2023).

[17] Licheng Guo, Yuze Chi, Jie Wang, Jason Lau, Weikang Qiao, Ecenur Ustun, Zhiru Zhang, and Jason Cong. 2021. AutoBridge：面向多芯粒 FPGA 高频高层次综合设计的粗粒度布局规划与流水线协同优化的自动化框架（AutoBridge: Coupling Coarse-Grained Floorplanning and Pipelining for High-Frequency HLS Design on Multi-Die FPGAs）. In *Proceedings of the 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '21)*.

[18] Licheng Guo, Jason Lau, Yuze Chi, Jie Wang, Cody Hao Yu, Zhe Chen, Zhiru Zhang, and Jason Cong. 2020. 面向提升设计频率的 FPGA 高层次综合隐式广播分析与优化（Analysis and Optimization of the Implicit Broadcasts in FPGA HLS to Improve Maximum Frequency）. In *Proceedings of the 2020 57th ACM/IEEE Design Automation Conference (DAC '20)*.

[19] Licheng Guo, Jason Lau, Zhenyuan Ruan, Peng Wei, and Jason Cong. 2019. 针对基因组测序中长读成对重叠分析的硬件加速：FPGA 与 GPU 的竞赛（Hardware Acceleration of Long Read Pairwise Overlapping in Genome Sequencing: A Race Between FPGA and GPU）. In *Proceedings of the IEEE 27th International Symposium on Field-Programmable Custom Computing Machines (FCCM '19)*.

[20] Licheng Guo, Pongstorn Maidee, Yun Zhou, Chris Lavin, Jie Wang, Yuze Chi, Weikang Qiao, Alireza Kaviani, Zhiru Zhang, and Jason Cong. 2022. RapidStream：FPGA HLS 设计的并行物理实现（RapidStream: Parallel Physical Implementation of FPGA HLS Designs）. In *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '22)*.

[21] Intel. 2024. Intel 高层次综合编译器（Intel High Level Synthesis Compiler）. https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/hls-compiler.html.

[22] Lana Josipović, Radhika Ghosal, and Paolo Ienne. 2018. 动态调度的高层次综合（Dynamically Scheduled High-level Synthesis）. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '18)*.

[23] Lana Josipović, Andrea Guerrieri, and Paolo Ienne. 2020. Dynamatic：从 C/C++ 到动态调度电路（Dynamatic: From C/C++ to Dynamically Scheduled Circuits）. In *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '20)*.

[24] JSON Schema. 2020. JSON Schema：用于描述 JSON 文档的媒体类型（JSON Schema: A Media Type for Describing JSON Documents）. https://json-schema.org/specification.html. Draft 2020-12.

[25] Moazin Khatti, Xingyu Tian, Yuze Chi, Licheng Guo, Jason Cong, and Zhenman Fang. 2023. PASTA：面向现代多芯粒 FPGA 可扩展任务并行 HLS 程序的编程与自动化支持（PASTA: Programming and Automation Support for Scalable Task-Parallel HLS Programs on Modern Multi-Die FPGAs）. In *Proceedings of the 2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM '23)*.

[26] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2021. MLIR：面向领域特定计算的可扩展编译器基础设施（MLIR: Scaling Compiler Infrastructure for Domain Specific Computation）. In *Proceedings of the 2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO '21)*.

[27] Chris Lavin and Alireza Kaviani. 2018. RapidWright: Enabling custom crafted implementations for FPGAs. In *Proceedings of the IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM '18)*.

[28] LLVM. [n. d.]. CIRCT：电路 IR 编译器与工具（CIRCT: Circuit IR Compilers and Tools）. https://circt.llvm.org/.

[29] Alec Lu, Zhenman Fang, Nazanin Farahpour, and Lesley Shannon. 2020. CHIP-KNN：云 FPGA 上可配置的高性能 K 近邻加速器（CHIP-KNN: A Configurable and High-Performance K-Nearest Neighbors Accelerator on Cloud FPGAs）. In *Proceedings of the 2020 International Conference on Field-Programmable Technology (ICFPT '20)*.

[30] Jianwen Luo, Xinzhe Liu, Fupeng Chen, and Yajun Ha. 2023. HRFF：面向基于 NoC 的可扩展多芯粒 FPGA 的层次递归布局规划框架（HRFF: Hierarchical and Recursive Floorplanning Framework for NoC-Based Scalable Multidie FPGAs）. *IEEE Transactions on Circuits and Systems I: Regular Papers* 70, 11 (2023).

[31] Kingshuk Majumder and Uday Bondhugula. 2024. HIR：基于 MLIR 的硬件加速器描述中间表示（HIR: An MLIR-based Intermediate Representation for Hardware Accelerator Description）. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '23)*.

[32] Mohammadmahdi Mazraeli, Yu Gao, and Paul Chow. 2023. 面向数据中心的大规模多 FPGA 应用分区（Partitioning Large-Scale, Multi-FPGA Applications for the Data Center）. In *Proceedings of the 33rd International Conference on Field-Programmable Logic and Applications (FPL '23)*.

[33] Tan Nguyen, Zachary Blair, Stephen Neuendorffer, and John Wawrzynek. 2023. SPADES：面向 Versal 可编程逻辑的高效设计流程（SPADES: A Productive Design Flow for Versal Programmable Logic）. In *Proceedings of the 33rd Conference on Field-Programmable Logic and Applications (FPL '23)*.

[34] Rachit Nigam, Samuel Thomas, Zhijing Li, and Adrian Sampson. 2021. 面向加速器生成器的编译器基础设施（A compiler infrastructure for accelerator generators）. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21)*.

[35] Michael Popoloski. 2024. Slang：SystemVerilog 编译器与语言服务（Slang: SystemVerilog compiler and language services）. https://github.com/MikePopoloski/slang.

[36] Matthew J. Saltzman. 2002. *Coin-OR: An Open-Source Library for Optimization*. Springer US, Boston, MA, 3–32.

[37] Siemens. 2024. *Catapult High-Level Synthesis and Verification*. https://static.sw.cdn.siemens.com/siemens-disw-assets/public/2viQ3qHCWJQxzqSkwBauMQ/en-US/Siemens-SW-Catapult-HLS-HLV-Platform-FS-82981-D1.pdf

[38] Jérôme Siméon and Philip Wadler. 2003. XML 的本质（The essence of XML）. *SIGPLAN Not.* 38, 1 (January 2003), 1–13.

[39] Jason Villarreal, Adrian Park, Walid Najjar, and Robert Halstead. 2010. 使用 ROCCC 2.0 在 C 语言中设计模块化硬件加速器（Designing Modular Hardware Accelerators in C with ROCCC 2.0）. In *Proceedings of the 2010 18th IEEE Annual*

International Symposium on Field-Programmable Custom Computing Machines (FCCM '10).

[40] Jie Wang, Licheng Guo, and Jason Cong. 2021. AutoSA：面向 FPGA 高性能脉动阵列的多面体编译器（AutoSA: A Polyhedral Compiler for High-Performance Systolic Arrays on FPGA）. In *Proceedings of the 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '21)*.

[41] Clifford Wolf, Johann Glaser, and Johannes Kepler. 2013. Yosys：一款免费的 Verilog 综合套件（Yosys: A free Verilog synthesis suite）. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*.

[42] Yuanlong Xiao, Aditya Hota, Dongjoon Park, and André DeHon. 2022. HiPR：面向快速增量 FPGA 编译的高层部分重配置（HiPR: High-level Partial Reconfiguration for Fast Incremental FPGA Compilation）. In *Proceedings of the 2022 32nd International Conference on Field-Programmable Logic and Applications (FPL '22)*.

[43] Yuanlong Xiao, Dongjoon Park, Zeyu Jason Niu, Aditya Hota, and André Dehon. 2024. ExHiPR：面向快速增量 FPGA 编译的扩展高层部分重配置（ExHiPR: Extended High-Level Partial Reconfiguration for Fast Incremental FPGA Compilation）. *ACM Transactions on Reconfigurable Technology and Systems* 17, 2, Article 21 (Mar 2024), 28 pages.

[44] Hanchen Ye, Cong Hao, Jianyi Cheng, Hyunmin Jeong, Jack Huang, Stephen Neuendorffer, and Deming Chen. 2022. ScaleHLS：基于多级中间表示的新型可扩展高层次综合框架（ScaleHLS: A New Scalable High-Level Synthesis Framework on Multi-Level Intermediate Representation）. In *Proceedings of the 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA '22)*.

[45] Shulin Zeng, Jun Liu, Guohao Dai, Xinhao Yang, Tianyu Fu, Hongyi Wang, Wenheng Ma, Hanbo Sun, Shiyao Li, Zixiao Huang, Yadong Dai, Jintao Li, Zehao Wang, Ruoyu Zhang, Kairui Wen, Xuefei Ning, and Yu Wang. 2024. FlightLLM：基于完整映射流程的 FPGA 高效大语言模型推理（FlightLLM: Efficient Large Language Model Inference with a Complete Mapping Flow on FPGAs）. In *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '24)*.