在 MLIR 之上构建高层次综合工具是个好主意吗？构建 Dynamatic 高层次综合编译器的经验

Jiahui Xu 苏黎世联邦理工学院 瑞士苏黎世
jxu@ethz.ch

Emmet Murphy 苏黎世联邦理工学院 瑞士苏黎世
emurphy@ethz.ch

Lana Josipović 苏黎世联邦理工学院 瑞士苏黎世
ljosipovic@ethz.ch

arXiv:2603.19856v1 [cs.PL] 20 Mar 2026

摘要

MLIR 项目最初推出时，承诺解决高层次综合社区对 LLVM 项目存在的问题。但事实真是如此吗？MLIR 是高层次综合领域“合适”或“最佳”的编译器基础设施吗？本文将分享我们在开发 Dyna- matic（github.com/EPFL-LAP/dynamatic）过程中的经验。

1 引言

LLVM 项目 [1] 一直是许多开源和商业高层次综合（HLS）项目 [2——7],这些项目将高层次软件代码编译为寄存器传输级代码。然而，LLVM 并不适合高层次综合工具 [8]：中间表示无法定制以表示电路，这迫使高层次综合工具为电路创建自定义且难以复用的中间表示。

MLIR [8, 9]有望解决这一问题：它引入了一种标准方式，用于定义、创建、分析和变换自定义中间表示操作。借助 MLIR，高层次综合开发者可以克服 LLVM中间表示的僵化：他们既可以定义受益于高层次变换的高层次中间表示，也可以定义具有电路语义的低层次中间表示。对于自定义中间表示，MLIR 提供了一些默认的 C语言++ 例程，用于创建和操作中间表示对象、转储和解析其文本格式等。这大大降低了实现开销，并促进了基于 MLIR 的工具之间的互操作性。

作为 Dynamatic 的主要开发者之一 [4, 5]——这是一款基于 MLIR 的高层次综合工具——我们认可并利用了上述优势。不过，我们也发现它的一些特性会造成阻碍。本文向 LATTE 社区介绍我们在使用 MLIR（截至版本 d8eb4ac）时遇到的问题：中间表示的定义、分析和变换，以及不同的基于 MLIR 的高层次综合项目之间的集成。我们认为，这些观察具有普遍性，并不局限于高层次综合 MLIR 项目。我们希望这些发现能为 MLIR 社区带来新的见解和讨论，并帮助开发者为未来的高层次综合工具做出更好的决策。

2 背景

本节将介绍 MLIR 和 Dynamatic 的相关背景。

MLIR [9] 是一种编译器基础设施，可用于定义自定义中间表示（称为方言）和变换过程。中间表示包含操作，这些操作会消耗并生成值。

Dynamatic 生成数据流电路 [10, 11], ，这些电路由指令粒度的数据流单元组成，并通过握手通道连接；数据封装在令牌中，并通过握手通道交换。在数据流电路中，只要输入有效，操作就会执行。因此，当控制流或内存访问模式不可预测时，Dynamatic 生成的动态调度电路具有性能优势 [12–14]。Dynamatic 是一种基于 MLIR 的编译器：它在专用的握手方言中将单元和通道表示为操作和值。

Dynamatic 已不再只是一个研究原型：它为众多出版物提供了基础 [11–37]——其中许多成果已融入主高层次综合流程 [11–13,15, 16,21–25, 28, 32]。我们曾在多场技术会议上举办教程并发表演讲 [5, 38]。Dynamatic 每月会向主分支合并约 30 个拉取请求（PR），并通过自动化的持续集成/持续交付流水线监控每个 PR，以避免编译错误或性能回退。经验较少的开发人员提交的代码贡献也会得到详细审查和反馈。

3 在学术环境中构建基于 MLIR 的高层次综合编译器

我们充分认识到，在学术环境中以 MLIR 为基础开展高层次综合项目具有显著优势。Dynamatic 主要由学生开发人员构建。我们许多人具有电气工程背景，并非受过专业培训的软件开发人员；我们常常不了解最佳软件实践。MLIR 帮助我们克服了这一挑战：它让我们深入了解复杂的面向对象编程架构在实践中的运作方式，其组织方式也启发我们打造一款模块化且易于理解的工具。这些优势使我们能够将更多开发精力投入高层次综合专用功能。

不过，我们也认识到，MLIR 的某些特性会给高层次综合工具带来障碍，下面将对此进行讨论。

4 MLIR 对开源高层次综合项目的局限性及 Dynamatic 中的案例研究

本节讨论了我们在基于 MLIR 构建 Dynamatic 高层次综合工具时遇到的问题。

4.1 MLIR 值并不适合用于建模图边

将软件中间表示或电路建模为图是一种常见做法 [39]。有些信息自然属于节点，有些则属于边。MLIR 允许在操作上添加注释（称为 属性）。然而，无法为 MLIR 值添加属性，而 MLIR 值表示操作之间交换的数据。MLIR 高层次综合框架该如何注释边信息?

案例研究 *1*：内存依赖。图 1展示了一个 C 程序及其对应 MLIR 的一部分。在`store3` 与`load2` 之间存在潜在的读后写依赖，该依赖发生在后续迭代中。这类依赖通常由软件中间表示分析确定；store 和 load 之间的依赖边会标注依赖距离，并将其用作高层次综合调度约束，以确保操作顺序正确。由于 MLIR 不允许对边或操作对进行标注，因此必须以非标准且不直观的方式表示这些信息。

Dynamatic 使用别名分析和多面体分析来确定内存访问之间的依赖对 [13, 15]。为了在高层次综合流程中保留这些信息，Dynamatic 为每个操作分配一个唯一名称，在整个高层次综合流水线中保留该名称，并使用一种难以阅读的格式来表示依赖。为确保正确性，Dynamatic 还必须保证每次变换都能保留名称成对有效性，从而确保依赖始终有效。

案例研究 *2*：记录软件性能分析。在高层次综合中，软件性能分析通常用于确定操作执行的次数和顺序（例如，用于早期性能估算，或对频繁执行的结构进行定向优化）。虽然这些信息自然应归属于基本块之间的控制流边，但 MLIR 中没有为控制流边添加注释的标准方法。

Dynamatic 依靠软件性能分析来识别应优先优化的频繁执行循环 [16, 22]。这些信息本可以记录在分支单元的输出边上（参见条件_分支节点，见图 2b）。然而，由于 MLIR 不允许对值添加注释，Dynamatic 不得不依靠外部 CSV 文件来保存这些信息。

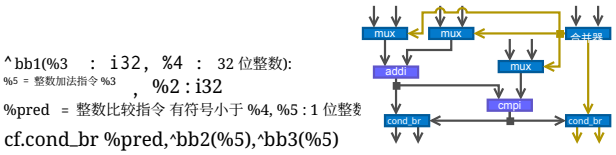
MLIR 若能提供完善的边注释支持，便可缓解这些问题。

4.2 BlockArguments 是否便于高层次综合转换?

在高层次综合流程中，我们需要转换一种软件表示——转换为电路。具体来说，我们必须将静态单赋值 ϕ节点（表示条件赋值）转换为多路复用器。在 MLIR 中，这种转换是否简单直接?

void histogram(in_int_t feature[1000], in_float_t weight [1000],inout_float_t hist[1000], in_int_t n) { for (int i = 0; i < n; ++i) { int m = feature[i]; float wt = weight[i]; float x = hist[m]; // -> "名称 =load2" hist[m] = x + wt; // -> "名称 = store3" } } (a) 一个存在读后写（RAW）依赖的程序。 %9 = memref.load %feature[%8] { handshake.name = "load2" } : memref<1000xf32>... // 依赖信息在概念上 // 是一条边 “st ore3 -> load2”，却以一种别扭的方式 // 标注在节点上。 memref.store %10, %feature[%11] { handshake.deps = # handshake<deps[["load2", 1]]>, handshake.name = " store3" } : memref<1000xf32>(b) 生成的 MLIR 代码片段。

图 1 在 MLIR 中标注边信息颇为棘手。



(a) **(b)**

图 2。Dynamatic 旨在将软件中间表示（左）转换为数据流电路（右）。

案例研究 *3*：将基本块参数转换为多路复用器。图 2a 展示了控制流方言中的一个基本块（这是 MLIR 内置的、用于表示顺序程序的方言）；它以一个带有标识符（bb1）的基本块参数 (%3, %4) 开始，并以条件分支结束。MLIR 用这些基本块参数表示 ϕ 节点 [40]。这种表示给软件到电路转换带来了以下问题：（a）基本块参数是没有生产者的值。（b）分支收集了流出的值，但这些值与后继基本块断开连接。前一个问题使模式重写——方言之间转换的标准方法——不适合这种转换，因为没有可匹配的操作。后一个问题使得定位多路复用器的输入变得不必要地复杂：与 LLVM ϕ 节点不同，LLVM 节点可以直接访问来自其他基本块的输入值；要获取相同的信息，我们必须先定位父基本块、分支，然后再找出传入该分支的值。

Dynamatic 使用 MLIR 的模式重写来完成这一转换。我们承认，这种解决方案并不理想

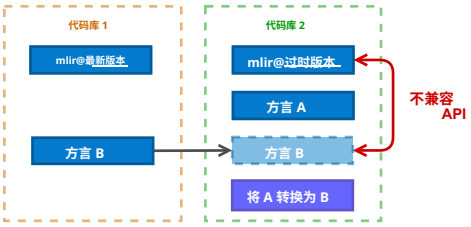


图 3. 集成两个 MLIR 项目颇具挑战。

因为重写必须作用于整个函数（这违背了局部重写规则的初衷）。

4.3 我们解决了软件分段问题吗?

MLIR 承诺通过让编译器项目重用彼此的转换遍，来解决软件碎片化问题。这在官方 MLIR代码库中或许确实可行，因为维护者对各个部分如何协同运作有统一的认识。然而，许多项目并不属于上游 MLIR 项目，例如 CIRCT [41], Polygeist [42],Dynamatic [4],MLIR-AIE [43], 等等。我们能顺利集成这些项目吗?

案例研究 *4*：为两个 *GitHub*代码库维护的方言实现一个变换通道。 图 3 展示了两个基于 MLIR 的项目的简化架构。这些项目实现了两种不同的方言，A 和 B。假设我们要实现一个新的 MLIR 通道，将方言 A 转换为方言 B。如果方言 A 和 B 的实现依赖于两个不同的 LLVM 项目，且应用程序接口不兼容（在任一 MLIR 版本中都会导致编译错误），那么从概念上讲，这几乎不可行。

Dynamatic 实现了一个自定义翻译通道，可将握手方言转换为 XLS MLIR方言 [44]。由于我们无法持续更新 Dynamatic 的 LLVM 版本并使其与 XLS 同步（XLS 每天都会更新），这一功能很快就有过时的风险。

4.4 我们是否拥有一个优秀的 HLS C前端，能够鼓励人们构建新的高层次综合工具?

高层次综合前端 将 C代码转换为中间表示，应用中间表示优化，并为电路生成标注重要信息。是否存在满足所有这些要求的 MLIR C语言前端?

案例研究 *5*：*C*到*MLIR*转换的现状。有些项目会将C语言转换为MLIR：*Polygeist* 将C抽象语法树转换为 MLIR SCF方言；而*CIRproject*（位于Clang中）会将C抽象语法树转换为CIR方言（一种用于建模C语义的方言）。这些方案本身无法提供细粒度的中间表示优化，因此在竞争力上不及基于LLVM的高层次综合编译器前端。

案例研究 *6*：对*LLVM*的依赖。由于LLVM提供了一套强大且经过充分验证的中间表示优化，大多数基于MLIR的项目最终都会将自定义方言降级为LLVM中间表示，以利用LLVM中间表示优化。利用LLVM中间表示的能力，消除了在MLIR中实现相同优化的动力。不幸的是，这也将LLVM的所有局限性引入了编译器流水线，而MLIR最初正是为了避免这些局限性。

由于现有的 MLIR C前端不如基于 LLVM 的前端具有竞争力，因此 Dynamatic 在这一过程中仍然依赖 LLVM中间表示。我们必须采用自定义变通方案来执行内存分析，并从原始 C代码中推断数组大小——这些变通方案在僵化的 LLVM框架中难以实现，但在 MLIR编译流程中却能自然适用。

5 超越高层次综合

虽然本文着重讨论了我们的特定用例和高层次综合经验，但我们认为，上述问题具有普遍性，并且并不仅限于高层次综合 MLIR 项目：(1) 软件编译器和硬件编译器通常都会利用边特定信息（例如内存依赖、分支概率），并可能受到第 4.1 节中所讨论的 MLIR 表达能力有限这一问题的影响。(2) 第 4.2 节所述的静态单赋值挑战主要面向硬件；然而，缺少 ϕ可能会使软件编译器中基于使用-定义链和数据依赖的中间表示优化与分析变得复杂。(3) 第4.3 节所述的软件分段问题具有完全普遍性，适用于任何 MLIR 项目。(4) 任何以 C语言作为输入语言的编译流程，都会受到第4.4 节中所讨论的限制。

因此，我们相信，解决这些问题将有益于整个 MLIR 社区。

6 结论

我们讨论了 MLIR 的一些主要特性，这些特性导致 Dynamatic 中不得不采用脆弱的变通方案。我们期待与 LATTE 社区探讨从中汲取的经验，以及基于 MLIR 的高层次综合项目未来的发展。

参考文献

[1] http://www.llvm.org。LLVM 编译器基础设施。2018年。网址：http://www.llvm.org。[2] Vivado 高层次综合。赛灵思公司。2018年。网址：http : / / www. xilinx . com / products / % 20design - tools / vivado / integration / esl - design.html。[3] Andrew Canis 等人。“LegUp：用于基于 FPGA 的处理器/加速器系统的开源高层次综合工具”。载于：《ACM 嵌入式计算系统汇刊》13.2（2013年9月），24:1–24:27。[4] Dynamatic。EPFL-LAP。网址：https : / / github . com / EPFL - LAP / dynamatic/tree/999dc3ce2fb95eac1dd39cad441fbdf6b8389aee。[5] Lana Josipović、Andrea Guerrieri 和 Paolo Ienne。“Dynamatic：从 C/C++ 到动态调度电路”。载于：第 28 届 ACM/SIGDA 国际现场可编程门阵列研讨会论文集。加利福尼亚州西塞德，2020年2月，第 1–10 页。网址：https://doi.org/10.1145/3373087.3375391。[6] Catapult 高层次综合与验证。西门子 EDA。2026年。网址：https://eda.sw.siemens.com/en-US/ic/catapult-high-level-synthesis/。[7] Fabrizio Ferrandi 等人。“特邀论文：Bambu：用于复杂应用高层次综合的开源研究框架”。载于：第 58 届 ACM/IEEE 设计自动化会议论文集。加利福尼亚州旧金山，2021年，第 1327–1330 页。网址：https://doi.org/10.1109/DAC18074.2021.9586110。[8] Chris Lattner 等人。“MLIR：面向领域特定计算的可扩展编译器基础设施”。载于：第 19 届 IEEE/ACM 国际代码生成与优化研讨会论文集。韩国首尔，2021年，第 2–14 页。网址：https://doi.org/10.1109/CGO51591.2021. 9370308。[9] 多层级中间表示编译器框架（MLIR）。LLVM 项目。2020年。网址：https://mlir.llvm.org/。[10] Jordi Cortadella、Marc Galceran-Oms 和 Mike Kishinevsky。“弹性系统”。载于：第 10 届 ACM/IEEE 协同设计形式化方法与模型国际会议论文集。2010年7月，第 149–58 页。[11] Lana Josipović、Radhika Ghosal 和 Paolo Ienne。“动态调度高层次综合”。载于：第 26 届 ACM/SIGDA 国际现场可编程门阵列研讨会论文集。加利福尼亚州蒙特雷，2018年2月，第 127–36 页。网址：https://doi.org/10. 1145/3174243.3174264。[12] Lana Josipović、Andrea Guerrieri 和 Paolo Ienne。“推测性数据流电路”。载于：第 27 届 ACM/SIGDA 国际现场可编程门阵列研讨会论文集。加利福尼亚州西塞德，2019年2月，第 162–71 页。[13] Lana Josipović、Philip Brisk 和 Paolo Ienne。“用于空间计算的乱序加载存储队列”。载于：《ACM 嵌入式计算系统汇刊》16.5s（2017年9月），125:1–125:19。网址：https://doi.org/10.1145/3126525。[14] Ayatallah Elakhras 等人。“最快者生存：在数据流电路中实现更多乱序执行”。载于：第 32 届国际现场可编程门阵列研讨会论文集。加利福尼亚州蒙特雷，2024年3月，第 44–54 页。网址：https://doi.org/10.1145/3626202.3637556。[15] Lana Josipović 等人。“要么缩减，要么舍弃！最小化数据流设计中加载存储队列的使用”。载于：IEEE 国际现场可编程技术会议论文集。中国天津，2019年12月，第 197–205 页。网址：https://doi.org/10.1109/ICFPT47387.2019. 00031。[16] Lana Josipović 等人。“高性能数据流电路的缓冲区放置与大小调整”。载于：第 28 届 ACM/SIGDA 国际现场可编程门阵列研讨会论文集。加利福尼亚州西塞德，2020年2月，第 186–96 页。网址：https://doi.org/10. 1145/3373087.3375314。[17] Jianyi Cheng 等人。《在高层次综合中结合动态与静态调度》。载于：第28届 ACM/SIGDA 国际现场可编程门阵列研讨会论文集。加利福尼亚州西塞德，2020年2月，第288–298页。[18] Lana Josipović、Andrea Guerrieri 和 Paolo Ienne。《将通用代码综合为动态调度电路》。载于：《IEEE 电路与系统杂志》21.2（2021年第二季度），第97–118页。[19] Lana Josipović 等人。《数据流电路中的资源共享》。载于：第30届 IEEE 现场可编程定制计算机研讨会论文集。纽约，2022年5月，第1–9页。网址：https://doi.org/10.1109/FCCM53951.2022.9786084。[20] Jianyi Cheng 等人。《面向高层次综合的动态块间调度》。载于：第32届现场可编程逻辑与应用国际会议论文集。英国贝尔法斯特，2022年8月，第243–252页。[21] Ayatallah Elakhras 等人。《通过更快的令牌传递释放弹性电路的并行性》。载于：第32届现场可编程逻辑与应用国际会议论文集。英国贝尔法斯特，2022年8月，第253–261页。网址：https://doi.org/10.1109/FPL57034.2022.00046。[22] Carmine Rizzi 等人。《用于精确调节数据流电路频率的全面时序模型》。载于：第22届现场可编程逻辑与应用国际会议论文集。英国贝尔法斯特，2022年8月，第375–383页。[23] Jiantao Liu、Carmine Rizzi 和 Lana Josipović。《高效数据流电路的加载-存储队列定容》。载于：第21届现场可编程技术国际会议论文集。香港，2022年12月，第1–9页。网址：https://doi.org/10.1109/ICFPT56656.2022.9974425。[24] Ayatallah Elakhras 等人。《直达队列：数据流电路中快速分配加载-存储队列》。载于：第31届国际现场可编程门阵列研讨会论文集。加利福尼亚州蒙特雷，2023年2月，第39–45页。网址：https://doi.org/10.1145/3543622.3573050。[25] Jiahui Xu 等人。《利用模型检测消除数据流电路的过度动态性》。载于：第31届 ACM/SIGDA 国际现场可编程门阵列研讨会论文集。加利福尼亚州蒙特雷，2023年2月，第27–37页。网址：https://doi.org/10.1145/3543622.3573196。[26] Carmine Rizzi、Andrea Guerrieri 和 Lana Josipović。《一种面向映射感知频率调节的迭代方法》。载于：第60届设计自动化会议论文集。加利福尼亚州旧金山，2023年7月，第1–6页。网址：https://doi.org/10.1109/DAC56929.2023.10247686。[27] Jiahui Xu 和 Lana Josipović。《面向可扩展数据流电路验证的自动归纳不变量生成》。载于：第42届国际计算机辅助设计会议论文集。加利福尼亚州旧金山，2023年10月，第1–9页。网址：https://doi.org/10.1109/ICCAD57390.2023.10323796。[28] Hanyu Wang、Carmine Rizzi 和 Lana Josipović。《MapBuf：面向高层次综合性能优化的技术映射与缓冲器插入同步进行》。载于：第42届国际计算机辅助设计会议论文集。加利福尼亚州旧金山，2023年11月，第1–9页。网址：https://doi.org/10.1109/ICCAD57390.2023.10323639。[29] Jiahui Xu 和 Lana Josipović。《通过延迟与占用率均衡抑制数据流电路的虚假动态性》。载于：第32届 ACM/SIGDA 国际现场可编程门阵列研讨会论文集。加利福尼亚州蒙特雷，2024年3月，第188–198页。网址：https://doi.org/10.1145/3626202.3637570。[30] Andrea Guerrieri 等人。《DynaRapid：加速从 C语言到布线后电路的流程》。载于：第34届现场可编程逻辑与应用国际会议论文集。意大利都灵，2024年9月，第24–32页。网址：https://doi.org/10.1109/FPL64840.2024.00014。

[31] Jiantao Liu 等人。《面向高层次综合生成的数据流电路的快速开关活动估算》。载于：第*34*届现场可编程逻辑与应用国际会议论文集。意大利都灵，2024年9月，第118–125页。网址：https://doi.org/10.1109/FPL64840.2024. 00025。[32] Jiahui Xu 和 Lana Josipović。《CRUSH：一种用于动态调度高层次综合中功能单元共享的基于信用的方案》。载于：第*30*届编程语言与操作系统架构支持国际会议论文集。鹿特丹，荷兰，2025年4月，第249–263页。网址：https://doi.org/10.1145/ 3669940.3707273。[33] Ayatallah Elakhras 等人。《ElasticMiter：形式化验证的数据流电路重写》。载于：第*30*届编程语言与操作系统架构支持国际会议论文集。鹿特丹，荷兰，2025年4月，第293–308页。网址：https://doi.org/10.1145/3676641.3715993。[34] Mathias Bouilloud、Lana Josipović 和 Wayne Luk。《面向动态调度高层次综合的资源与阶段感知》。载于：第*15*届高能效加速器与可重构技术国际研讨会论文集。熊本，日本，2025年5月，第14–24页。网址：https://doi.org/10.1145/3728179. 3728194。[35] Shun Katsumi、Emmet Murphy 和 Lana Josipović。《EagerlyElastic：动态调度高层次综合中的正确性构造式急切执行》。载于：第*34*届国际现场可编程门阵列研讨会论文集。加利福尼亚州西塞德，2026年2月，第103–113页。网址：https://doi.org/10.1145/3748173.3779196。

[36] Rouzbeh Pirayadi 等：《告别加载存储队列：动态高层次综合中的内存访问重排序定制电路》。载于：第34届现场可编程门阵列国际研讨会论文集。加利福尼亚州西塞德，2026年2月，第92–102页。网址：https://doi.org/10.1145/3748173. 3779204.[37] Yann Herklotz 等：《Graphiti：数据流电路中经形式化验证的乱序执行》。载于：第31届编程语言与操作系统体系结构支持国际会议论文集。即将发表。匹兹堡，2026年3月。[38] Lana Josipović 等：《Dynamatic Reloaded：基于 MLIR 的动态调度高层次综合编译器》。FPGA ’24 教程。2024年3月。网址：https://www.isfpga.org/past/fpga2024/workshops-tutorials/。[39] Giovanni De Micheli。《数字电路的综合与优化》。纽约：麦格劳-希尔，1994年。[40]《MLIR 原理：块参数与 PHI 节点》。LLVM 项目。2026年。网址：https : / /mlir. llvm . org / docs / Rationale / Rationale / #block - arguments-vs-phi-nodes。[41] https://github.com/llvm/circt。CIRCT IR 编译器与工具。2020年。网址：https://github.com/llvm/circt。[42] William S. Moses 等：《Polygeist：将 C 提升为多面体 MLIR》。载于：第30届并行架构与编译技术国际会议论文集。佐治亚州亚特兰大，2021年，第45–59页。[43] IRON API 与基于 MLIR 的 AI 引擎工具链。AMD。2026年。网址：https://github.com/Xilinx/mlir-aie。[44] XLS：加速硬件综合。谷歌公司。网址：https://github.com/ google/xls。