ChiProgramBench：大语言模型智能体能否通过引脚级实验重建高效电路?

Tianrun Gou[1]，Qingyao Yang[1,2],Wendong Xu[1,2],Liang Chen[1],Kuan Li[1],Ngai Wong[2]
UniPat AI [2]香港大学

arXiv:2611.31082v1 [cs.AI] 23 Nov 2026

摘要——**理解一个能够运行却无法检查的系统，是一项核心工程技能：设计实验，针对隐藏状态和时序提出假设，舍弃被观测结果证伪的假设，最终构建出可用且高效的替代品。为评估大语言模型智能体在硬件领域的这一能力，我们推出 ChiProgramBench，其中包含 18 项任务，取材自开源处理器、加速器、编解码器、变换器、互连电路和控制电路。智能体只能驱动并读取一个隐藏的有状态仿真参考实现的引脚，然后提交由公共库单元构成的门级网表。智能体需在两种设置下工作：黑盒发现仅提供匿名化的功能引脚和固定存储器的接口，而函数引导重构则会进一步说明电路的用途及各引脚的作用。隐藏测试用例会逐周期将输出与参考实现进行比较；只有全部用例均通过，才会根据相对于参考实现的功耗、延迟和面积表现给予分数。在针对 6 种大语言模型智能体配置进行的 432 次试验中，成功的智能体能够恢复算法和数值约定，但黑盒试验中仅有 12.0%、函数引导试验中仅有 14.8% 通过全部用例。函数引导重构将平均功能得分（即通过用例所占的周期加权比例）从 24.6% 提高到 30.9%。在 58 个全部通过的设计中，只有三个在功耗、面积和延迟综合表现上优于参考实现，因此，行为恢复与高效实现仍是彼此不同的挑战。**

索引术语—**硬件重构、编码代理、主动实验、验证、基准测试**

I. 引言

面对一个可以运行但无法检视的有状态系统，工程师的工作方式如同实验科学家：设计实验，就隐藏状态和时序形成假设，舍弃被观测结果否定的假设 [1], ，并最终构建既正确又高效的替代实现。近期的智能体基准测试将此类实验视为一项核心能力 [2], [3]。硬件使这一循环变得具体且可严格检验，而且这种情况在实践中也会出现：第三方知识产权核仅以加密仿真模型形式交付 [4] ，或者遗留模块的寿命超过了其设计资料。难点在于，替代实现可能在迄今尝试的每一个输入上都与原件一致，却在下一个输入上出现差异。例如，重建的图像解码器或许能正确解码不间断的数据流，但当其消费者停顿或复位中断某项事务时便会失败。因此，智能体必须选择能够暴露此类差异的输入序列，进而还原计算、并发信号与内部状态如何在时钟边界处相互作用。

现有基准测试仅覆盖这一循环的部分环节。如今，人工智能代理能够生成寄存器传输级设计和模拟电路，并操作电子设计自动化工具，但设计基准测试会提供规范 [5]–[9], ，因此智能体无需自行发现行为即可进行构建。交互式基准测试在抽象游戏 [2] 和仿真实验室 [3], 中测试实验步骤，但无需根据智能体学到的内容进行构建。ProgramBench 通过根据可执行参考实现 [10]重建程序，闭合了软件领域的这一循环。主动学习和预言机引导合成也会根据查询 [11], [12]推断模型。硬件还增加了时钟状态、复位语义、反压，以及以标准单元计量的成本；目前没有任何基准测试能够评估智能体从实验到高效电路的完整循环。

我们推出 **ChiProgramBench**，其中包含 18 项任务，源自开源处理器、加速器、编解码器、变换器、互连电路和控制电路。每项任务的参考实现都是上游设计经过优化的门级实现。智能体通过探测工具在隔离的仿真环境中驱动参考实现的引脚，编写自己的测试，并提交核心的网表，映射到公开的标准单元库；大型存储器则作为固定静态随机存取存储器保留在核心之外。隐藏的周期精确测试用例用于衡量功能恢复情况；只有通过所有测试用例后，功耗、性能（延迟）和面积（功耗、性能和面积）才会根据相对于参考实现的表现计分。两种设置的区别在于告知智能体的信息：黑盒发现仅提供匿名化的功能引脚及其位宽和 SRAM 接口；而基于功能引导的重建 还会说明电路的用途以及各引脚的作用。两种设置都要求智能体自行发现延迟、状态转换和协议交互。

功能得分 F 是通过案例中按周期加权的占比，PPA 得分 S 则将功耗、延迟和面积的参考实现与提交结果之比相乘；完全通过后奖励为 $r = 0.5F + 0.5S$ ，否则为 $0.5F$ ，因此参考实现得分为 $r = 1$ ，而效率更高的设计得分会超过该值。在六种大语言模型智能体配置中，在 432 次试验中，当两种设定的权重相同时，GPT-6-Astra 和 Claude Opus 5.5 的平均奖励分别为 0.48 和 0.45，而其余四个模型均低于 0.07（图 1）。完全通过仍然罕见：黑盒试验中占 12.0%，函数引导试验中占 14.8%，且有八项任务从未被完全通过。函数引导重构将平均功能得分从 24.6% 提高到 30.9%，然而有 16 份提交的功能得分超过 99%，却未能通过所有测试用例。在 58 个完全通过的实现中，只有三个改善了参考实现的功耗–面积–延迟乘积；中位数实现的面积为 1.22× 倍，功耗为 1.52× 倍。我们的贡献如下：

- 包含 18 项硬件重建任务的套件，涵盖两种信息设定，并提供经过优化的参考实现、周期精确的隐藏测试，以及故障设计对照（测试必须拒绝的损坏网表）。
- 一种隔离、具备状态的探测机制，以及一项独立评估，将精确到周期的功能测试与所提交核心的功耗、性能和面积关联起来。
- 一项包含432次试验的研究，并通过轨迹分析探究智能体在哪些方面成功、又在哪些方面失败。

II. 相关工作

先前工作仅涉及从实验到电路的完整流程中的部分环节。

硬件设计智能体。 VerilogEval 和 RTLLM 测试根据自然语言规范生成寄存器传输级代码 [5], [6]; CVDP 则增加了调试、验证和智能体任务 [13]。RTLCoder 训练开放的寄存器传输级模型 [14]; AutoChip 利用编译器和仿真反馈修复生成的寄存器传输级代码 [15], ，而 RTLFixer 则修复生成代码中的语法错误 [16]。其他工作针对模拟电路设计 [8], [9] ，或功耗、性能和面积优化及设计收敛 [17], [18]。Build-A-Chip-Bench 从工作负载规范开始，一直到生成 GDS；与 ChiProgramBench 一样，只有通过隐藏检查后才计入功耗、性能和面积，其检查也包括布局 [7]。

从观测结果推断电路。主动自动机学习通过在小型输入字母表上进行查询来推断有限状态模型 [11]–[19], ，包括真实处理器的缓存替换策略 [20]。

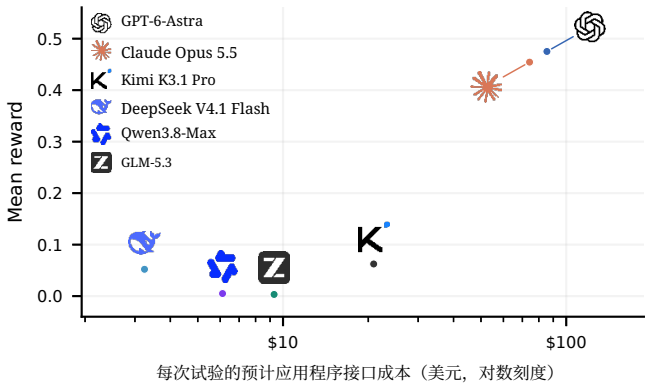

图 1. 每次试验的奖励与估算应用程序接口成本之间的关系，两个信息设定的权重相同。成本根据记录的令牌用量和公开标准美元费率计算，包括缓存上下文和推理输出。徽标标示各模型在 72 次试验中的平均值；两个领先模型的徽标经过错位处理，并用引导线指向其精确数值（圆点）。GPT-6-Astra 在 Codex 中运行，其他模型则在 Claude Code 中运行。

预言机引导合成针对区分性输入查询输入—输出预言机 [12]。SAT 攻击利用正常工作的芯片作为该预言机，恢复锁定网表的密钥 [21] 以及伪装时序电路的逻辑门，而无需扫描访问 [22]。其他方法则从输入—输出示例中学习紧凑的组合逻辑 [23] 或 RISC-V CPU [24] 。与此同时，相关工作在最小化组合电路语法规模的同时，恢复其精确 RTL [25] ，并利用大语言模型智能体推断处理器微架构 [26]。验证工具从 RTL 和轨迹中挖掘断言 [27], [28] ，或对插桩 RTL 进行模糊测试 [29], [30]。网表逆向工程则恢复寄存器和算术模块 [31], [32]。而我们的智能体针对仅能通过引脚观测到的各种时序设计进行实验，并提交综合后的门级网表，而非 RTL。

智能体实验基准。除了游戏和仿真实验室之外， [2], [3], CodeARC 还让智能体查询一个隐藏函数（可提供或不提供具有提示性的名称），并综合出匹配代码 [33]。ProgramBench 向智能体提供可执行文件及其文档，与我们的函数引导设置相同，并根据隐藏测试对重建的程序进行评分 [10]。我们的黑盒设置完全不提供功能描述。CodeARC 和 ProgramBench 都不评估重建程序的效率，而 ChiProgramBench 则通过对照优化参考实现测量功耗、性能和面积。

III. CHIPROGRAMBENCH

每项任务均固定一种电路配置，智能体通过图 2 所示的循环重建其时序行为；图 3 概述了任务和参考实现的构建方式。表 I 列出了涵盖 6 个领域的 18 项任务，参考核心规模从 3.7k 到 619.2k 个单元不等。基于 NVDLA 的任务仅涵盖卷积和后处理流水线，因为完整设计规模过大，无法使用开源电子设计自动化工具进行验证。

任务遵循 Harbor格式 [34]: 每个软件包包含指令、容器化环境、可执行验证测试，以及作为预言机解的参考实现，其还会对智能体的功耗、性能和面积进行归一化。

A. 参考实现构建

对于每项任务，我们精选一个开源设计，定义其接口和测试工作负载，然后构建经过验证的映射参考实现（图 3）。每个参考实现都以映射后的上游寄存器传输级设计为起点，随后进行优化。我们重构算术运算（例如三乘法器复数乘积以及 Booth 树或 Dadda树），将延迟线和查找表存储

表 I 18 项任务：参考核心单元数（不包括固定静态随机存取存储器）、隐藏测试用例及其总周期数。

Task	电路	单元数（千）	案例	周期（k）
中央处理器和图形处理器				
CV32E40P [35]	RISC-V 核心	61.0	396	163.8
VeeR EH1 [36]	RISC-V 核心	174.5	142	51.6
Vortex [37]	通用图形处理器	619.2	163	111.0
VeriGPU [38]	GPU	6.7	147	924.5
机器学习加速器				
基于 NVDLA [39]	卷积流水线	21.5	109	132.1
RedMulE [40]	矩阵引擎	341.4	155	31.8
OpenTPU [41]	脉动阵列 TPU	78.2	183	74.5
ITA [42]	注意力引擎	303.8	138	50.3
媒体编解码与压缩				
JPEG [43]	图像解码器	68.6	168	3,838.1
JPEG-LS [44]	无损编码器	19.9	150	621.3
Gzip [45]	压缩器	52.7	108	897.2
变换、编码与密码学				
FFT [46]	1024点快速傅里叶变换	25.6	101	580.0
里德–所罗门 [47]	里德–所罗门解码器	3.7	112	886.1
Camellia [48]	分组密码	44.8	125	8.8
互连与存储器				
WB DMA [49]	DMA 控制器	47.2	158	46.1
DDR3 AXI [50]	DRAM 控制器	6.1	178	10,986.1
输入/输出与控制				
以太网 [51]	UDP/IP 协议栈	10.6	169	84.3
FOC [52]	电机控制器	21.8	106	17,845.6

在锁存器中，采用独热与或读选择，并对空闲寄存器进行时钟门控。综合调优选择 ABC [53]脚本和延迟目标，保留锁存器和时钟门控单元，缓冲高扇出网络，并根据时序裕量调整单元尺寸。只有当某次修订的实测功耗–面积–延迟乘积降低，且在工作负载的门级重放和独立差分测试中，所有定义的输出位均仍与上一版本匹配时，才会保留该修订；这些有限检查并不构成等价性证明。

B. 黑盒与功能引导设置

两种设置都提供 instruction.md 和 OBSERVE.md，其中规定了提交接口、探测操作、输出采样时间、评分以及功耗、性能和面积测量条件；两者还使用相同的逐任务验证器和参考实现。在两种设置中，时钟和复位均已命名，任何固定静态随机存取存储器都会通过其端口、容量和访问时序进行说明；其他功能端口仅作为输入i或输出j出现，并标明其位宽，而黑盒发现 不会提供更多信息。在函数引导重构中，简要描述会补充电路的用途以及每个引脚的作用，例如操作数、结果、有效或请求。在试验期间，智能体只能看到自己的编译器和仿真结果；隐藏测试和官方评分均不可用。

C. 探测访问与隔离

智能体只能通过探测工具观察参考实现。该工具以命令行工具和 Python 库的形式提供。请求会列出随时间变化的输入，可以采用定时引脚驱动、时钟推进和读取操作的形式，也可以按每个周期一行的形式给出，并采用固定的驱动与采样时间；探测工具会仿真参考实现并返回采样输出。文档说明了输出相对于时钟的采样时刻，以及未知（x）位的报告方式。探测工具会像驱动其他输入一样驱动复位。隐藏测试中使用的每种激励都可以通过探测工具发送。

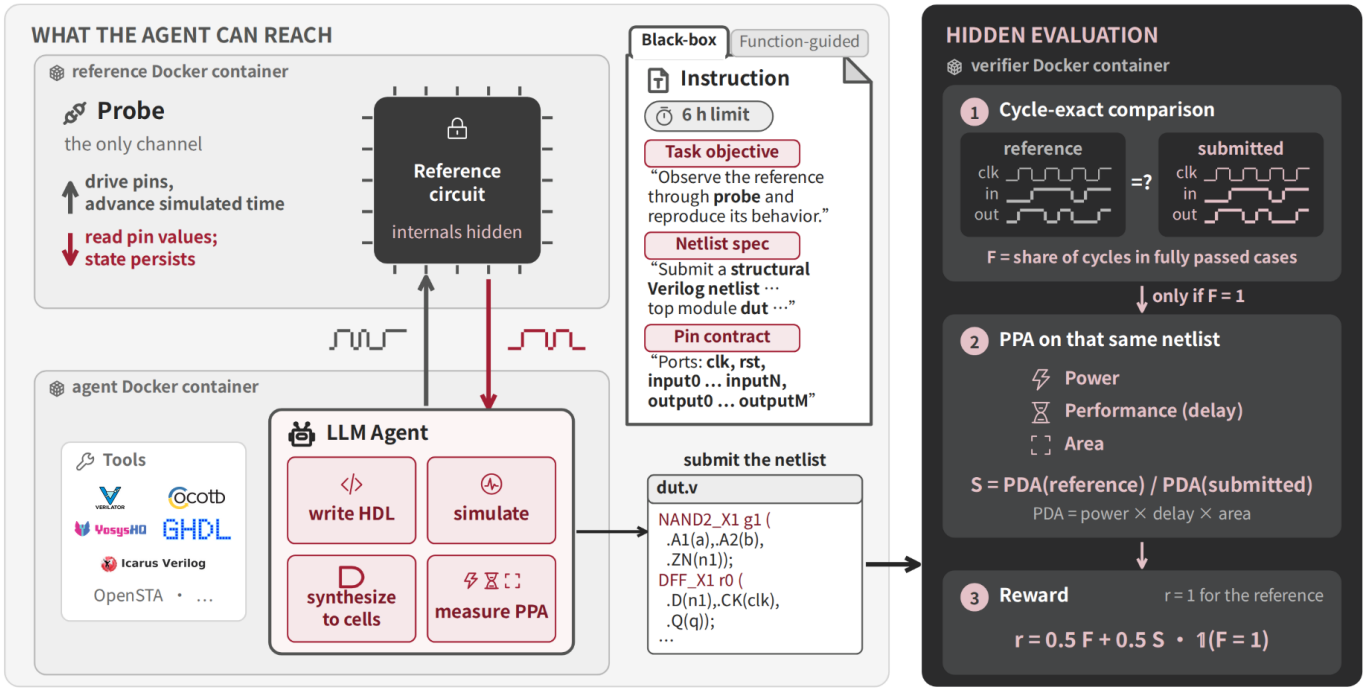

图 2. 重建与评分。智能体查询有状态且经过隔离的参考实现，并提交映射后的核心网表。隐藏测试用于衡量功能恢复，只有全部通过后才会启用功耗、性能和面积评分。指令选项卡展示了这两种设置。

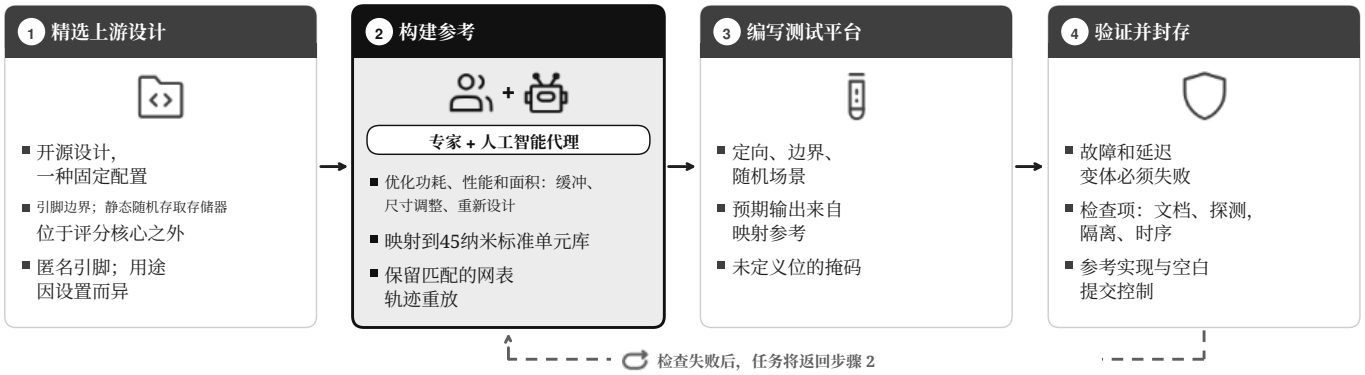

图 3. 基准构建。专家和 AI 辅助开发生成映射后的参考实现；在发布前，会对工作负载、隔离和故障设计对照进行检查。

两种请求格式都可以在会话中运行，会话会在不同请求之间保留电路状态和仿真时间，使后续激励可以根据上一次观测结果来确定；例如，在有效载荷固定的情况下改变 valid/ready 时序，就能揭示接收和缓冲行为。关闭并重新打开会创建一个全新的实例。

参考实现在一个独立的 Docker 容器中运行，且无法访问网络；智能体的容器仅挂载一个保存探测器 Unix 域套接字的只读卷，因此参考文件永远不会进入其中。沙箱检查还会移除已安装的软件实现，避免其充当密码学任务的替代参考实现。

D. 提交的核心

智能体提交门级网表：仅实例化公共 Nangate45 库中的单元的结构化 Verilog [54], [55], , 顶层模块为 dut，并采用规定的核心接口。允许使用多个源文件和结构层次。验证器会直接评估所提交的网表，不进行综合或重新映射，因此不接受寄存器传输级代码；行为过程、延迟、不支持的单元以及初始状态覆盖都会被拒绝。图 4 展示了包含固定静态随机存取存储器的任务核心边界。

指定的静态随机存取存储器保持固定，不属于提交内容。Nan-gate45 仅提供标准单元，而开源 SRAM 编译器 OpenRAM [56] 生成的时钟控制宏单元最多有两个端口；相比之下，任务中的存储器从单端口静态随机存取存储器到具有组合读操作的四端口阵列不等。固定这些存储器可避免将大型阵列当作人为的触发器实现来评估。它们的接口、容量、掩码、初始化和访问时序均有明确规定并公开，探测器会报告每个 SRAM 请求和读响应；SRAM 模型保留在参考环境中。核心中保留的寄存器和缓冲器计入功耗、性能和面积。在 FFT 任务中，提交的核心在三个 SRAM 的端口处终止。隐藏测试会重放参考存储器响应，并比较功能输出以及

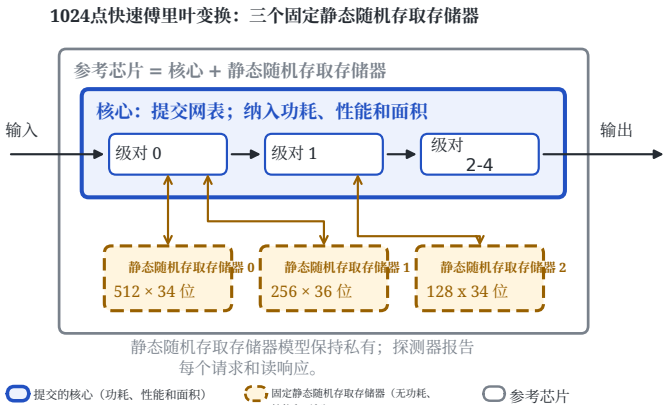

图 4. FFT 任务的核心边界。被探测的芯片包含核心和三个固定静态随机存取存储器；智能体仅提交核心（蓝色），其功耗、性能和面积将用于评分。若没有固定静态随机存取存储器，核心就是整个芯片。该组织方式为示意图。

存储器请求。候选方案可以更改内部逻辑、算术运算和状态编码，但必须保持引脚级行为、固定的存储器容量，以及可观测的核心与存储器事务时序。

E. 隐藏测试与评分

隐藏测试用例涵盖计算、复位与恢复、反压、算术边界和协议交互。每个用例都是固定激励，在固定周期数内包含规定的输出采样值；表 I 列出了每项任务的用例数和总周期数。验证器使用 Icarus Verilog [57] 和 Nangate45 单元模型，在提交的网表上重放每个用例。

功能恢复。只有在每个规定的采样点上，所有被比较的输出位都匹配，用例才算通过。仅当参考实现对有效载荷位进行限定时才会比较该位：关联的有效、请求或使能输出已置位，且参考位本身为 0 或 1。这些限定控制信号始终会被比较。凡参考值要求为 0 或 1，候选值为 x/z 都会导致失败。令 N_i 为用例 i的固定周期数，并令 $\pi_i = 1$ 表示整个用例是否通过：

$$F = \frac{\sum_i N_i \pi_i}{\sum_i N_i}. \tag{1}$$

核心实现质量。完全通过（$F = 1$）后，面积 A 汇总提交实现中的叶级单元面积。OpenSTA [58] 会根据候选实现自身在固定私有工作负载窗口中的仿真活动，在单元库的典型工艺角下计算功耗 P，候选实现和参考实现采用相同方法，即总窗口能量除以总窗口时长。参考实现和候选实现共用的 OpenSTA 构建版本增加了一项本地补丁：活动传播还会从每个经过 VCD 注释的引脚开始，因此测得的活动可以传播到主输入锥之外的逻辑，例如自由运行寄存器环路；时序和功耗公式保持不变。测量前会先进行输入驱动的初始化和预热。仅对于功耗，未知启动值按零计，并且之后每个测得的单元引脚都必须保持二进制状态，但有两个任务保留四态估算；不使用默认翻转率。

OpenSTA 还会在固定的静态时序约束下测量延迟。对于捕获域周期 T_j、最差建立裕量 s_j以及任务参考周期 T_{ref}，

$$D = T_{\mathrm{ref}} \max_j \left[\max\left(0, 1 - \frac{s_j}{T_j}\right) \right]. \tag{2}$$

对于具有 $T_{\mathrm{ref}} = T_j$ 的单个域，这可简化为 max(0，$T_j - s_j$)。每个任务的库角、时钟、输入/输出条件和约束都是固定的。时钟会通过单元时钟弧传播，因此会对时钟门控单元进行时序分析，而锁存器路径则使用 OpenSTA 的默认时间借用。SRAM宏位于时序图之外：读数据会在时钟边沿后 1 ns 到达，而请求会在下一个边沿被捕获。在三个任务中，验证器还会推导出由最多四级级联二分频触发器生成的时钟。任何其他未解析的时钟、时序环路、未约束端点、没有受约束路径的设计，或为零的指标，都会使功耗、性能和面积指标不可用；随后验证器会报告测量失败，而不是给出奖励——我们的任何完全通过都未触发这种情况。在没有布线寄生参数的情况下，D 会在布局前测量核心时序。

PPA 得分 S 和总体奖励 r 为

$$S = \frac{P_{\mathrm{ref}}}{P} \frac{A_{\mathrm{ref}}}{A} \frac{D_{\mathrm{ref}}}{D}, \tag{3}$$

$$r = 0.5F + 0.5S\,\mathbb{1}(F = 1). \tag{4}$$

参考实现本身得到 $S = r = 1$，改进后的结果可以超过 1。缺失或无效的提交得分为零。在全部 36 个任务–设置组合中，完整验证器将参考实现评为 1，将空提交评为 0；在 Harbor框架中，这分别是 oracle $= 1$ 和 nop $= 0$。

验证。我们还通过参考运行、探测回放和会话测试，检查探测与评估之间的一致性。故障设计对照是结构上有效的网表，但其输出延迟一个周期、地址或输出位取反、写使能卡住，或内部数据遭到破坏。所有对照至少有一个用例未通过（$F < 1$），因此均无法获得功耗、性能和面积方面的评分；这也包括那些旨在保留部分得分的轻微变体。

IV. 实验

现在，我们将衡量智能体完成这一闭环的频率及其成本。

A. 设置

我们使用 Codex CLI 0.164.0 评估 GPT-6-Astra，并使用 Claude Code 2.1.301 评估 Claude Opus 5.5、DeepSeek V4.1 Flash、Kimi K3.1 Pro、GLM-5.3 和 Qwen3.8-Max。每种模型与脚手架配置在每个设置下的每项任务上进行两次独立试验，总计 432 次。试验从公开任务文件和空白提交目录开始，采用所支持的最高推理强度，时间预算为六小时（不包括设置和评分）。每个智能体容器根据任务不同，配备 4–32 个 CPU 核心和 8–64 GB 内存，且无法访问外部网络；智能体只能通过探测访问参考实现。容器提供 YosysHQ 的 OSS CAD Suite [59], , 其中包括 Yosys 0.70 [60] 、ABC [53],、Icarus Verilog 14.0-devel [57],、Verilator、GHDL、cocotb、Amaranth 和 Migen，以及 Chisel [61], 、SpinalHDL [62], 、Bluespec [63], 和 Python。容器还包含验证器所用的 Nangate45 Liberty 和单元模型，以及其 OpenSTA 3.1 构建版本 [58], , 以便智能体能在本地综合、仿真候选网表并对其进行时序分析。

我们对每项任务中的两次试验取平均值，并对 18 项任务赋予相同权重。所有最终提交均计入平均值。

B. 总体性能

在每种设置下，GPT-6-Astra 和 Claude Opus 5.5 的平均奖励达到 0.44–0.48，而其他四种配置在任一设置下的最高值仅为 0.102（表 II 和图 5），在两种设置取平均后则低于 0.07（图 1）。完整的功能通过仍不常见：黑盒试验中仅有 12.0% 通过，函数引导试验中仅有 14.8% 通过。在两种设置下，18 个任务中有 8 个没有任何完全



图 5. 两种信息设定下各任务的平均奖励。每个单元格对两次试验取平均值；两个面板均采用与表 I 相同的色标和任务顺序。

表 II 结果（每行 36 次试验）。 r 和 F 是所有试验的平均值；功耗、性能和面积比率及 S 仅对完全通过的试验计算几何平均值，因此 S 是三个比率乘积的倒数。步骤数和探测数分别是所有智能体工具调用（包括子智能体）的单次试验中位数，以及驱动探测的调用的单次试验中位数；一次调用可以仿真任意长度的多组激励，因此这两个计数均不代表波形数或仿真周期数。

	全部 36 次试验					仅统计完全通过			
模型	$r\uparrow$	$F\uparrow$	完全通过	步骤	探测	$A/A_{\mathrm{ref}}\downarrow$	$P/P_{\mathrm{ref}}\downarrow$	$D/D_{\mathrm{ref}}\downarrow$	$S\uparrow$
黑盒发现									
GPT-6-Astra	**0.484**	**0.743**	12/36	456	61	1.334	1.424	0.990	0.532
Claude Opus 5.5	0.443	0.645	14/36	279	53	1.194	1.643	0.976	0.523
Kimi K3.1 Pro	0.023	0.047	0/36	754	58	–	–	–	–
DeepSeek V4.1 Flash	0.016	0.032	0/36	749	145	–	–	–	–
Qwen3.8-Max	0.003	0.007	0/36	162	41	–	–	–	–
GLM-5.3	0.003	0.006	0/36	217	58	–	–	–	–
函数引导重构									
GPT-6-Astra	**0.467**	**0.779**	16/36	337	56	1.598	2.761	1.859	0.122
Claude Opus 5.5	0.466	0.680	14/36	266	40	1.148	1.445	0.997	0.605
Kimi K3.1 Pro	0.102	0.198	1/36	853	42	1.886	2.324	1.226	0.186
DeepSeek V4.1 Flash	0.088	0.175	1/36	749	98	2.217	5.434	1.282	0.065
Qwen3.8-Max	0.007	0.013	0/36	156	33	–	–	–	–
GLM-5.3	0.003	0.007	0/36	181	37	–	–	–	–

通过的提交。在 155 个具有 $0 < F < 1$ 的提交中，有 23 2 16 99 . % 具有 $F \geq 0.9$；其功能得分超过 a%，却未通过每个测试用例。周期加权得分可能会被较长且正确重建的工作负载所主导，而较短的交互仍然是错误的。

C. 函数引导设置带来的变化

函数引导设置下，Aggregate F 从 24.6% 上升至 30.9%。图 6a显示，各任务的提升幅度不一：DDR3 AXI 提高了 39.0 个百分点，基于 NVDLA 的流水线提高了 31.0 个百分点，里德-所罗门解码提高了 21.3 个百分点。VeeR EH1 和 ITA 变化不大。

在 DDR3 黑盒运行中，Claude Opus 5.5 首先改变匿名输入，以识别突发长度和字节选通字段。函数引导描述提供了这些字段的作用，因此实验可以从有意义的事务开始。仲裁、缓冲和响应时序仍然需要通过实验来还原。在一段函数引导的基于 NVDLA 的轨迹中，智能体先探测静态配置，然后在活动流量期间更改控制项，以了解算术运算和旁路控制如何影响已经排队的数据。借助描述，智能体还减少了探测工具调用：每次试验的中位数从 216 次黑盒试验中的 57 次降至 216 次函数引导试验中的 44 次。这些调用次数与 F 并不相关（在 432 次试验中，斯皮尔曼 $\rho = -0.10$ ）：DeepSeek V4.1 Flash 的探测调用次数最多（其 72 次试验的中位数为 135 次），却很少通过。

功能提升未必会提高奖励：GPT-6-Astra 的平均值 F 从 74.3% 升至 77.9%，但其奖励下降了 0.017，因为函数引导的完全通过所对应的功耗、性能和面积较低。

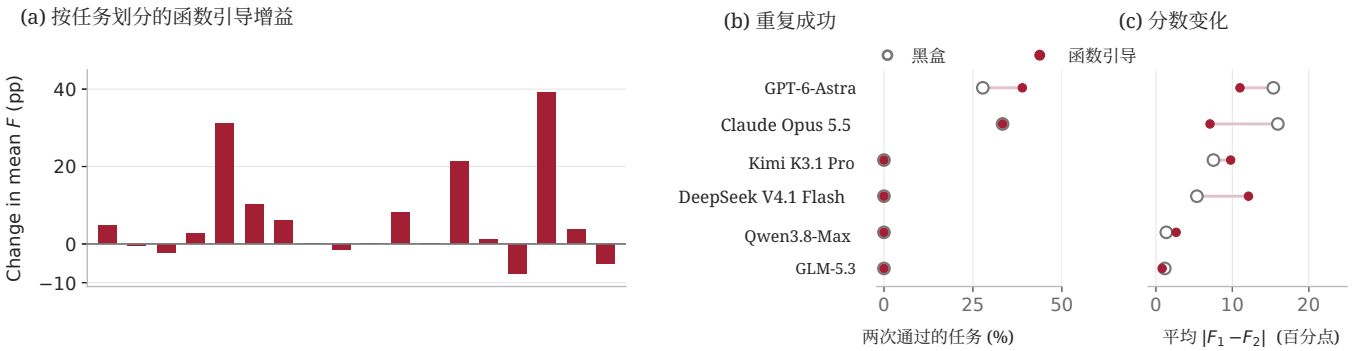

图 6. 函数引导设置与一致性。(a) 函数引导与黑盒的均值差 F，每项任务、每种设置进行 12 次试验。(b) 两次试验均通过的任务百分比。(c) 两次试验的 F 绝对差的均值。面板 (b 、 c) 包含每种模型和设置下的全部 18 项任务；pp 表示百分点。

D. 重复试验间的一致性

我们通过两次独立运行来衡量每项任务的一致性：两次运行中均通过的任务比例，以及两次运行的功能得分平均绝对差（图 6b、c）。对于 GPT-6- Astra，在函数引导设置下，两次均通过的比例从 27.8% 升至 38.9%；对于 Claude Opus 5.5，该比例在两种设置下均为 33.3%。两者的平均得分差距分别从 15.4 个百分点缩小至 11.0 个百分点，以及从 16.0 个百分点缩小至 7.1 个百分点。

函数引导设置扩大了 DeepSeek V4.1 Flash 和 Kimi K3.1 Pro 的分数差距。GLM-5.3 和 Qwen3.8-Max 的分数差距较小，只是因为它们两次试验的得分都接近零。

V. 重建分析

接下来，我们追踪一条成功轨迹以及反复出现的失败。

A. 从实验到可运行电路

一次成功的黑盒 FFT 轨迹展示了如何将观测结果转化为可实现的行为模型。GPT-6-Astra 首先检查了静态随机存取存储器的读写活动：读取地址比写入地址超前三个，存储的字对表明存在一条复数数据流水线。启动后，恒定输入每 1,024 个样本仅产生一个非零结果。模式切换揭示了不同变换模式下的缩放差异，而算术探测则识别出平局取奇舍入规则。

接下来，智能体研究了有效帧之间的行为。数据流间隙期间，静态随机存取存储器地址计数器继续运行，而蝶形相位计数器则复位。复位会清除控制状态，但保留蝶形数据；乘法器流水线则表现出独特的启动行为。这些观测结果为周期精确模型提供了依据，随后进行了 RTL 重放、技术映射和门级检查。时序报告随后促使设计采用复位缓冲和寄存的部分帧标志。最终核心通过了全部 101 个隐藏测试用例，其中包含 $S = 0.739$。其延迟接近参考实现，但面积和功耗仍分别高出 14.7% 和 18.3%。

B. 能力限制

完成计算与控制。 在 ITA 上，Claude Opus 5.5 最多通过 138 个用例中的 1 个。黑盒尝试将结果数据和结果有效信号固定为零；函数引导的尝试恢复了点积和量化，但在握手、SRAM控制和结果方面存在不匹配。在每种设置下，GPT-6-Astra 的一次尝试通过全部 138 个用例，另一次通过 133 个；通过 133 个用例的尝试在 SRAM写使能或数据方面未能通过 5 个用例。两次完全通过均优于参考实现（$S = 2.05$ 和 2.35），面积为参考实现的 0.83–0.93×，延迟为 0.56×。

还原条件协议行为。 智能体能够复现常见计算，但输入相关的状态转换仍不完整。以太网在多次试验中提供了一个具体示例。GPT-6-Astra 的四份提交功能得分均超过 99%，但每份都未通过两个 ARP 响应用例。其他失败涉及协议路径之间的仲裁。在一次函数引导的运行中，26 次本地实验均未报告不匹配，而隐藏测试发现头部有效信号从未置位，且在协议交互下目标地址有误。Claude Opus 5.5 在四次运行中均通过了这些用例。要解决这些错误，需要开展实验，让相互竞争的状态模型预测不同的输出。重复执行常规事务可以提高观察到的一致性，却仍可能无法测试这些差异。

在重建过程中保持效率。 在 58 个完全通过的提交中，面积、功耗和延迟比率的中位数分别为 1．221、1．520 和 1．000，相对于参考实现而言。只有三个实现改善了联合功耗–面积–延迟乘积：GPT-6-Astra 的两个 ITA 实现，以及一个黑盒 Claude Opus 5.5 JPEG-LS 实现（$S = 1.089$）。在完全通过的提交中，89.7% 的实现面积和功耗均超过参考实现；41.4% 的实现改善了延迟。一些设计还会产生较长的时序路径，在 JPEG-LS 上达到 58.99×参考延迟，在 OpenTPU 上达到41.98×。

VI. 讨论与结论

智能体能够从引脚级实验中恢复复杂计算和细微的时序规则，但完全通过仍然罕见，而且大多数通过的设计成本高于参考实现。由于每个隐藏激励也都可以通过探测器发送，因此失败并非源于访问受限；在六小时预算内，智能体必须选择真正重要的实验。它们能够确认普通事务中的行为，却很少构造出能区分不同状态模型的案例，例如被中断的握手或协议路径之间的仲裁。函数引导的描述可提高 F 并减少探测调用，但探测次数无法预测成功与否，描述也不会使重建电路更加高效。

由于只有在完全通过后才计算功耗、性能和面积，因此遗漏一个案例就会失去全部效率得分。我们的测试是有限的且在布局前进行，因此完全通过并不能证明等价性，延迟也不包括布线寄生参数。智能体还可能记住公开的上游设计，这是一个有待未来研究解决的混淆因素。取得进展需要三种能力，而 ChiPro- gramBench 在相同的电路上对其进行衡量：能够揭示

条件状态行为、准确的映射实现，以及在保留恢复的行为的同时进行优化。

参考实现

[1] E. F. Moore，《时序机上的思想实验》，载于《自动机研究》系列《Annals of Mathematics Studies》，C. E. Shannon 和 J. McCarthy 编，Princeton Univ. Press，1956年，第34卷，第129–153页。[2] ARC Prize Foundation，《ARC-AGI-3：前沿智能体智能的新挑战》，2026年，arXiv:2603.24621。[3] P. Jansen 等，《DiscoveryWorld：用于开发和评估自动化科学发现智能体的虚拟环境》，载于 Adv. Neural Inf. Process. Syst. (NeurIPS)，第37卷，2024年，第10 088–10 116页。[4]《电子设计知识产权（IP）加密与管理推荐实践》，IEEE Std 1735-2023，2023年11月。[5] M. Liu、N. Pinckney、B. Khailany 和 H. Ren，《特邀论文：VerilogEval：评估用于 Verilog 代码生成的大语言模型》，载于 Proc. IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD)，2023年，第1–8页。[6] Y. Lu、S. Liu、Q. Zhang 和 Z. Xie，《RTLLM：面向大语言模型的开源寄存器传输级生成基准》，载于 Proc. Asia and South Pacific Design Autom. Conf. (ASP-DAC)，2024年，第722–727页。[7] S. Gunukula、J. Y. Wang 和 K. Zhang，《介绍 Build-A-Chip-Bench：从工作负载规范到 GDS》，ChipAgents Research Blog，chipagents.ai/research/build-a-chip-bench，2026年10月。[8] Y. Lai 等，《AnalogCoder：通过免训练代码生成进行模拟电路设计》，载于 Proc. AAAI Conf. Artif. Intell. (AAAI)，第39卷，第1期，2025年，第379–387页。[9] Analog Design Bench Team，《长时程模拟设计基准：对智能体执行数小时模拟与混合信号电路设计任务的基准测试》，2026年，arXiv:2609.33356。[10] J. Yang 等，《ProgramBench：语言模型能否从零重建程序？》，载于 Adv. Neural Inf. Process. Syst. (NeurIPS)，2026年。[11] D. Angluin，《从查询和反例中学习正则集合》，Inf. Comput.，第75卷，第2期，第87–106页，1987年11月。[12] S. Jha、S. Gulwani、S. A. Seshia 和 A. Tiwari，《预言机引导的基于组件的程序合成》，载于 Proc. ACM/IEEE Int. Conf. Softw. Eng. (ICSE)，第1卷，2010年，第215–224页。[13] N. Pinckney 等，《Comprehensive Verilog Design Problems：用于评估大语言模型和智能体执行寄存器传输级设计与验证的新一代基准数据集》，2025年，arXiv:2506.14074。[14] S. Liu 等，《RTLCoder：完全开源且高效的大语言模型辅助RTL代码生成技术》，IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.，第44卷，第4期，第1448–1461页，2025年4月。[15] J. Blocklove、S. Thakur、B. Tan、H. Pearce、S. Garg 和 R. Karri，《利用 EDA工具反馈自动改进基于大语言模型的 Verilog 生成》，ACM Trans. Des. Autom. Electron. Syst.，第30卷，第6期，第100:1–100:26页，2025年11月。[16] Y.-D. Tsai、M. Liu 和 H. Ren，《RTLFixer：利用大语言模型自动修复RTL语法错误》，载于 Proc. ACM/IEEE Design Autom. Conf. (DAC)，2024年，第1–6页。[17] P. Liu、N. Xu、J. Tang、Y. Cao 和 C. Ding，《跨越电路设计的最后一英里：PostEDA-Bench，一种面向PPA收敛和DRC修复的分层基准》，2026年，arXiv:2605.06936。[18] P. Zhou 等，《CLOSER-Bench：评估面向硬件智能体的预算约束的跨阶段设计收敛》，2026年，arXiv:2607.16632。[19] F. Vaandrager，《模型学习》，Commun. ACM，第60卷，第2期，第86–95页，2017年2月。[20] P. Vila、P. Ganty、M. Guarnieri 和 B. Köpf，《CacheQuery：从硬件缓存中学习替换策略》，载于 Proc. ACM SIGPLAN Conf. Program. Lang. Des. Implement. (PLDI)，2020年，第519–532页。[21] P. Subra manyan、S. Ray 和 S. Malik，《评估逻辑加密算法的安全性》，载于 Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)，2015年，第137–143页。[22] M. El Massad、S. Garg 和 M. Tripunitara，《无扫描访问条件下对伪装时序电路进行逆向工程》，载于 Proc. IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD)，2017年，第33–40页。[23] S. Rai 等，《逻辑综合遇上机器学习：以精确性换取泛化》，载于 Proc. Des., Autom. Test Eur. Conf. Exhib. (DATE)，2021年，第1026–1031页。[24] S. Cheng 等，《通过从输入输出示例中学习实现自动化CPU设计》，载于 Proc. Int. Joint Conf. Artif. Intell. (IJCAI)，2024年，第3843–3853页。[25] W. Tang、F. Liu、R. Wang、T. Hwang、B. Yu 和 C.-Y. Wang，“RTLREX：大语言模型引导的高级 RTL 精确恢复框架”，2026，arXiv:2611.10798。[26]G. Lyu 等，“ProcReverser：用于处理器漏洞分析的多智能体黑盒微架构逆向工程”，2026，arXiv:2611.20827。[27] S. Vasudevan、D. Sheridan、S. Patel、D. Tcheng、B. Tuohy 和 D. Johnson，“GoldMine：结合数据挖掘与静态分析的自动断言生成”，载于《欧洲设计、自动化与测试会议论文集》(DATE)，2010年，第626–629页。[28] W. Li、A. Forin 和 S. A. Seshia，“面向验证与诊断的可扩展规范挖掘”，载于《ACM/IEEE设计自动化会议论文集》(DAC)，2010年，第755–760页。[29] K. Laeufer、J. Koenig、D. Kim、J. Bachrach 和 K. Sen，“RFUZZ：在现场可编程门阵列上对寄存器传输级进行覆盖率引导模糊测试”，载于《IEEE/ACM计算机辅助设计国际会议论文集》(ICCAD)，2018年，第28:1–28:8页。[30] J. Hur、S. Song、D. Kwon、E. Baek、J. Kim 和 B. Lee，“DifuzzRTL：用于发现 CPU 漏洞的差分模糊测试”，载于《IEEE安全与隐私研讨会论文集》(SP)，2021年，第1286–1303页。[31] N. Albartus、M. Hoffmann、S. Temme、L. Azriel 和 C. Paar，“DANA：用于门级网表逆向工程的通用数据流分析”，《IACR密码硬件与嵌入式系统汇刊》，第2020卷，第4期，第309–336页，2020年。[32] N. Wu、Y. Li、C. Hao、S. Dai、C. Yu 和 Y. Xie，“Gamora：面向大规模布尔网络的图学习符号推理”，载于《ACM/IEEE设计自动化会议论文集》(DAC)，2023年，第1–6页。[33] A. Wei 等，“CodeARC：大语言模型智能体归纳式程序合成推理能力基准测试”，载于《语言模型会议论文集》(COLM)，2025年。[34] Harbor 框架团队，“Harbor：用于在容器环境中评估和优化智能体与模型的框架”，软件，Zenodo，doi: 10.5281/zenodo.20953922，2026年。[35] M. Gautschi 等，“配备数字信号处理扩展的近阈值 RISC-V 核心，适用于可扩展的物联网终端设备”，《IEEE超大规模集成电路系统汇刊》，第25卷，第10期，第2700–2713页，2017年10月。[36] CHIPS 联盟，“Cores-VeeR-EH1”，github.com/chipsalliance/Cores-VeeR-EH1，2026年。[37] B. Tine、K. P. Yalamarthy、F. Elsabbagh 和 H. Kim，“Vortex：扩展 RISC-V 指令集架构以支持通用图形处理器和 3D 图形”，载于《IEEE/ACM微架构国际研讨会论文集》(MICRO)，2021年，第754–766页。[38] H. Perkins，“VeriGPU”，github.com/hughperkins/VeriGPU，2026年。[39] 英伟达，“NVDLA 开源硬件”，github.com/nvdla/hw，2026年。[40] Y. Tortorella、L. Bertaccini、L. Benini、D. Rossi 和 F. Conti，“RedMulE：用于灵活、高能效片上线性代数和 TinyML 训练加速的混合精度矩阵乘法运算引擎”，《未来一代计算机系统》，第149卷，第122–135页，2023年12月。[41] 加州大学圣巴巴拉分校 ArchLab，“OpenTPU”，github.com/UCSBarchlab/OpenTPU，2026年。[42] G. ̇Isla mo˘glu 等，“ITA：面向量化变换器的节能注意力与 Softmax 加速器”，载于《IEEE/ACM低功耗电子设计国际研讨会论文集》(ISLPED)，2023年，第1–6页。[43] Ultra-Embedded，“core_jpeg”，github.com/ultraembedded/core_jpeg，2026年。[44] WangXuan95，“FPGA-JPEG-LS-encoder”，github.com/WangXuan95/FPGA-JPEG-LS-encoder，2026年。[45] WangXuan95，“FPGA-Gzip-compressor”，github.com/WangXuan95/FPGA-Gzip-compressor，2026年。[46] digicroo，“pure-vhdl-fft”，github.com/digicroo/pure-vhdl-fft，2026年。[47] FreeCores，“reed_solomon_decoder”，github.com/freecores/reed_solomon_decoder，2026年。[48] FreeCores，“camellia-vhdl”，github.com/freecores/camellia-vhdl，2026年。[49] FreeCores，“wb_dma”，github.com/freecores/wb_dma，2026年。[50] Ultra-Embedded，“core_ddr3_controller”，github.com/ultraembedded/core_ddr3_controller，2026年。[51] A. Forencich，“verilog-ethernet”，github.com/alexforencich/verilog-ethernet，2026年。[52] WangXuan95，“FPGA-FOC”，github.com/WangXuan95/FPGA-FOC，2026年。[53] R. Brayton 和 A. Mishchenko，“ABC：一种学术界与工业界通用的验证工具”，载于《计算机辅助验证国际会议论文集》(CAV)，2010年，第24–40页。[54] Nangate 公司，“Nangate 45 纳米开放单元库”，FreePDK45 标准单元库；Liberty 文件来自 github.com/The-OpenROAD-Project/OpenROAD-flow-scripts，2011年。

[55] J. E. Stine等，“FreePDK：一种开源、支持变化感知的设计套件”，载于《*IEEE*微电子系统教育国际会议论文集》*(MSE)*，2007年，第173–174页。[56] M. R. Guthaus、J. E. Stine、S. Ataei、B. Chen、B. Wu 和 M. Sarwar，“OpenRAM：一种开源存储器编译器”，载于《*IEEE/ACM*计算机辅助设计国际会议论文集》*(ICCAD)*，2016年，第1–6页。[57] S. Williams，“Icarus Verilog”，开发版本14.0（提交 dfc800afd），github.com/steveicarus/iverilog，2026年。[58] J. Cherry，“OpenSTA：Parallax 静态时序分析器”，版本3.1.0（提交 f2f431a，经过本地补丁修改），github.com/parallaxsw/OpenSTA，2026年。[59] YosysHQ GmbH，“OSS CAD Suite”，构建版本 2026-11-01，github.com/YosysHQ/oss-cad-suite-build，2026年。[60] C. Wolf，“Yosys开源综合套件”，版本0。70+286 （提交 63eecf3dd），github.com/YosysHQ/yosys，2026年。[61] J. Bachrach等，“Chisel：使用嵌入于 Scala 的语言构建硬件”，载于《*ACM/IEEE*设计自动化会议论文集》*(DAC)*，2012年，第1216–1225页。[62] C. Papon 和 Y. Xiao，“SpinalHDL”，版本1.15.1，github.com/SpinalHDL/SpinalHDL，2026年。[63] Bluespec Inc.，“Bluespec编译器 (BSC)”，版本2026.07.1，github.com/B-Lang-org/bsc，2026年。