华为杯

第八届中国研究生创“芯”大赛

验证自动化流程设计说明

作品名称：混沌逻辑-大模型多智能体自动数字 IC 前端设计器

团队名称：混沌逻辑

参赛队员：孙林涵，诸人豪，张煜

2025 年 6 月 18 日

目录

摘要

本软件通过设置项目管理、设计工程师和验证工程师三个 Agent，模仿一般的 IC 设计流程，使通用大语言模型（LLM）生成可靠的 RTL 代码。本文档将详细介绍软件各个 Agent 的设计，以及软件的运行流程。

1 介绍

近年来，随着大语言模型（Large Language Model, LLM）技术逐渐进入应用场景，越来越多的领域开始尝试使用 LLM 来辅助工作。数字前端设计作为一个复杂的工程领域，也开始探索如何利用 LLM 来提高设计效率和准确性。然而，与其他领域相比，本领域的 RTL 代码开源少、且开源代码缺乏高质量设计；HDL 语言与 LLM 大量学习的软件语言看似相似，实则在基础思路上存在着巨大差距。此外，和 LLM 已取得广泛应用的软件工程相比，IC 设计的容错率极低，对代码正确率要求高。因此，直接调用已有的通用 LLM 到数字前端设计中是极不合理的。

为了解决这些问题，本项目设计了一套由通用 LLM 驱动的 RTL 代码的生成与验证自动化软件，旨在提高 IC 设计的效率和准确性。

2 自动化系统设计

本项目的设计目标是，在使用市场上现有的通用 LLM 的同时，最大化地减少 LLM 的幻觉，按照用户预期快速、自动地完成初版设计。为此，我们仿照现有的 IC 设计流程设计了多个智能体（Agent），逐步展开用户的需求；并在 LLM 运行的各个阶段尽可能地使用 Synopsys 工具链对大模型的结果提出反馈，避免设计错误在工作流中传导。

2.1 Agent 设计

考虑到前面提到的问题，期望 LLM 在单轮对话中直接给出正确的 RTL 设计是不现实的。要确保大模型的输出正确且符合用户需求，则需要为 LLM 提供反馈，在多轮迭代后取得期望的输出。但是，LLM 的输出较慢（如本项目使用的 Deepseek API 仅能达到 40 Token/s），若依赖人类产生反馈信息，则将占用用户大量时间，背离了自动化的设计初衷。因此，本项目模仿一般的 IC 设计流程，设置了项目管理、设计工程师和验证工程师三个 Agent，每个 Agent 均具有单独的短期记忆、长期记忆，并根据其职责设计了对应的工具供其使用。LLM 在使用工具时，间接调用了 Synopsys 的 VCS 等工具进行语法检查、仿真等，输出的结果将自动地反馈给大模型，实现了一定程度的自动化。下面详细介绍各个 Agent 的设计。

2.1.1 项目工程师 Agent

项目工程师解读用户输入的需求，生成项目的技术规范（下文简称 Spec），并将其存储在文件中。当设计完成时，项目工程师还会根据验证工程师的验证报告，检查设计是否满足规范要求。

系统提示词：

```
你是混沌逻辑公司 IC 设计部门的项目管理员，你的团队中包括设计工程师和验证工程师。

你需要将客户的需求转换为规范的 Spec 以协助设计。当收到验证工程师的验证报告时，你需要检查其是否符合客户需求
不准定义验证工作的内容
不准定义工艺节点和时钟频率
```

用户提示词，根据用户输入生成完整 spec 时：

```
# 项目当前状态

等待项目管理员的 Spec

# 客户需求

{user_requirements}

# 当前任务

根据客户需求，用中文完成 Spec，其中应当包括该模块的 Verilog IO 定义。使用 submit_spec 工具以
↪ 提交该 Spec。
```

用户提示词，验收项目时：

```
# 项目当前状态

等待项目管理员审阅验证报告

# Spec

{spec}

# 验证报告

{verification_report}

# 当前任务

根据 Spec，审阅验证报告，决定：

1. 批准当前验证报告，可用 accept_report 提交。

或者，

2. 否决当前验证报告，并通过 submit_spec 提交新的 spec ，使其他员工纠正现有版本的错误
```

工具：

```
submit_spec = {
    "type": "function",
    "function": {
        "name": "submit_spec",
        "description": " 提交 SPEC 文档。",
        "strict": True,
```

```
        "parameters": {
            "type": "object",
            "properties": {
                "spec": {"type": "string", "description": "SPEC 文档内容"},
                "overwrite": {"type": "boolean", "description": "true 表示覆盖现有的
                ↪ SPEC, false 表示将其追加到现有 SPEC 中"}
            },
            "required": ["spec", "overwrite"],
            "additionalProperties": False
        }
    }
}

accept_report = {
    "type": "function",
    "function": {
        "name": "accept_report",
        "description": " 批准当前报告。",
        "strict": True
    }
}
```

2.1.2 设计工程师 Agent

设计工程师根据项目工程师提供的 Spec，生成 RTL 代码。设计工程师会将生成的代码存储在文件中，并在必要时进行修改。为避免 LLM 生成的代码存在语法错误，每次设计工程师提交代码时，都会使用 VCS 的 vlogan 工具进行语法检查。若检查出语法错误，设计工程师会根据错误信息进行修改，直到消除所有报错为止。

另外，设计工程师还会根据验证工程师的反馈，修改 RTL 代码以满足验证需求。

系统提示词：

```
你是混沌逻辑公司 IC 设计部门的设计工程师，你的团队中包括项目管理员和验证工程师。

你需要根据项目管理员提供的 Spec，设计对应的 Verilog RTL IP 块。

代码仅可通过外部工具提交，不能生成 markdown 代码块。
```

用户提示词，生成 RTL 代码时：

```
# 项目当前状态

项目管理员的 Spec 已完成

等待设计工程师的 RTL 代码
```

```
# 项目管理员的 Spec

{spec}

# 当前任务

根据 Spec 设计 Verilog RTL 代码
```

工具：

```
submit_design = {
    "type": "function",
    "function": {
        "name": "submit_design",
        "description": " 提交你的 Verilog 设计代码。设计代码将保存在一个 .v 文件中。你的设
        ↪  计代码在提交后会自动进行语法检查。",
        "strict": True,
        "parameters": {
            "type": "object",
            "properties": {
                "code": {"type": "string", "description": "Verilog 设计代码"}
            },
            "required": ["code"],
            "additionalProperties": False
        }
    }
}
```

2.1.3 验证工程师 Agent

验证工程师根据项目工程师提供的 Spec 与用户提供的验证方案，生成验证计划，并编写测试用例。验证工程师会使用 VCS 编译仿真程序，并运行测试用例，生成验证报告。同样，当编译仿真程序时，若出现编译错误，验证工程师也会根据错误信息进行修改，直至成功编译得到 simv 仿真程序为止。接下来，验证工程师会检查 simv 仿真程序的输出，判断 RTL 代码是否存在问题。当 RTL 代码存在问题时，验证工程师会将问题反馈给设计工程师，并要求其修改 RTL 代码。当验证工程师认为 RTL 代码满足验证需求时，会将验证报告提交给项目工程师。

系统提示词：

```
你是混沌逻辑公司 IC 设计部门的验证工程师，你的团队中包括项目管理员和设计工程师。

你需要根据项目管理员提供的 Spec ，严格按照验证计划设计 Testbench，以验证设计工程师提供的
↪  Verilog RTL 代码，找出任何可能存在的问题。
```

```
# 注意!

testbench timescale 固定为 1ns/100ps

代码仅可通过外部工具提交，不要生成 markdown 代码块

参考模型已提供，模块名为 ref_model，端口定义与 dut 一致，构建 testbench 时应当例化参考模型

不要对 dut 的任何输出进行直接检查，所有检查都应当将 dut 的输出与参考模型相比较，报错时打印
↪  dut 与参考模型的对比

使用 ref_inst 作为实例名实例化 ref_model

不要在时钟的上升沿改变输入信号或检查输出信号

SystemVerilog 中，必须在 Testbench* 开头处 * 声明新变量
```

用户提示词，生成 testbench 时：

```
# 项目当前状态

项目管理员的 Spec 已完成

设计工程师的 RTL 代码已完成

等待验证工程师的报告

# 项目管理员的 Spec

{spec}

# 验证计划

{veri_plan}

# 当前任务

1. 按照验证计划，使用 submit_testbench 提交 testbench ，testbench 将自动执行，并返回测试结果

2. 当测试完成后，使用 write_feedback ，将测试中的问题反馈给设计工程师

3. 当测试无误后，使用 write_verification_report 提交验证报告
```

工具：

```
submit_testbench = {
    "type": "function",
    "function": {
        "name": "submit_testbench",
        "description": " 提交你的 Testbench 代码。Testbench 将保存在一个 tb.v 文件中",
        "strict": True,
        "parameters": {
            "type": "object",
            "properties": {
                "code": {"type": "string", "description": "Testbench 代码"}
            },
            "required": ["code"],
            "additionalProperties": False
        }
    }
}
write_feedback = {
    "type": "function",
    "function": {
        "name": "write_feedback",
        "description": " 撰写验证反馈。",
        "strict": True,
        "parameters": {
            "type": "object",
            "properties": {
                "text": {"type": "string", "description": " 验证反馈"}
            },
            "required": ["text"],
            "additionalProperties": False
        }
    }
}
write_verification_report = {
    "type": "function",
    "function": {
        "name": "write_verification_report",
        "description": " 撰写验证报告。",
        "strict": True,
        "parameters": {
            "type": "object",
            "properties": {
                "report": {"type": "string", "description": " 验证报告"}
            },
            "required": ["report"],
            "additionalProperties": False
        }
    }
```

```
}
```

3 运行流程

接下来，介绍本软件是如何调用各个 Agent 以完成 RTL 代码的生成与验证的。

图 1展示了本软件的运行流程。首先，项目工程师 Agent 会解读用户需求，生成 Spec 并存储在文件中。接着，设计工程师 Agent 会根据 Spec 生成 RTL 代码，并进行语法检查。若存在语法错误，设计工程师会进行修改。然后，验证工程师 Agent 会根据 Spec 与验证方案生成验证计划，并编写测试用例。验证工程师会编译仿真程序并运行测试用例，生成验证报告。当 RTL 代码存在问题时，验证工程师会将问题反馈给设计工程师，设计工程师会修改 RTL 代码并重新提交。最终，当验证工程师认为 RTL 代码满足验证需求时，会将验证报告提交给项目工程师。项目工程师确认 RTL 代码满足规范要求后，整个流程结束。

为了避免设计工程师的错误误导验证工程师，设计工程师和验证工程师将分别根据项目工程师提供的 Spec 与用户提供的验证方案生成各自的代码和测试用例。整个流程中多次出现的语法检查和编译步骤，确保了生成的代码和测试用例都是正确的。

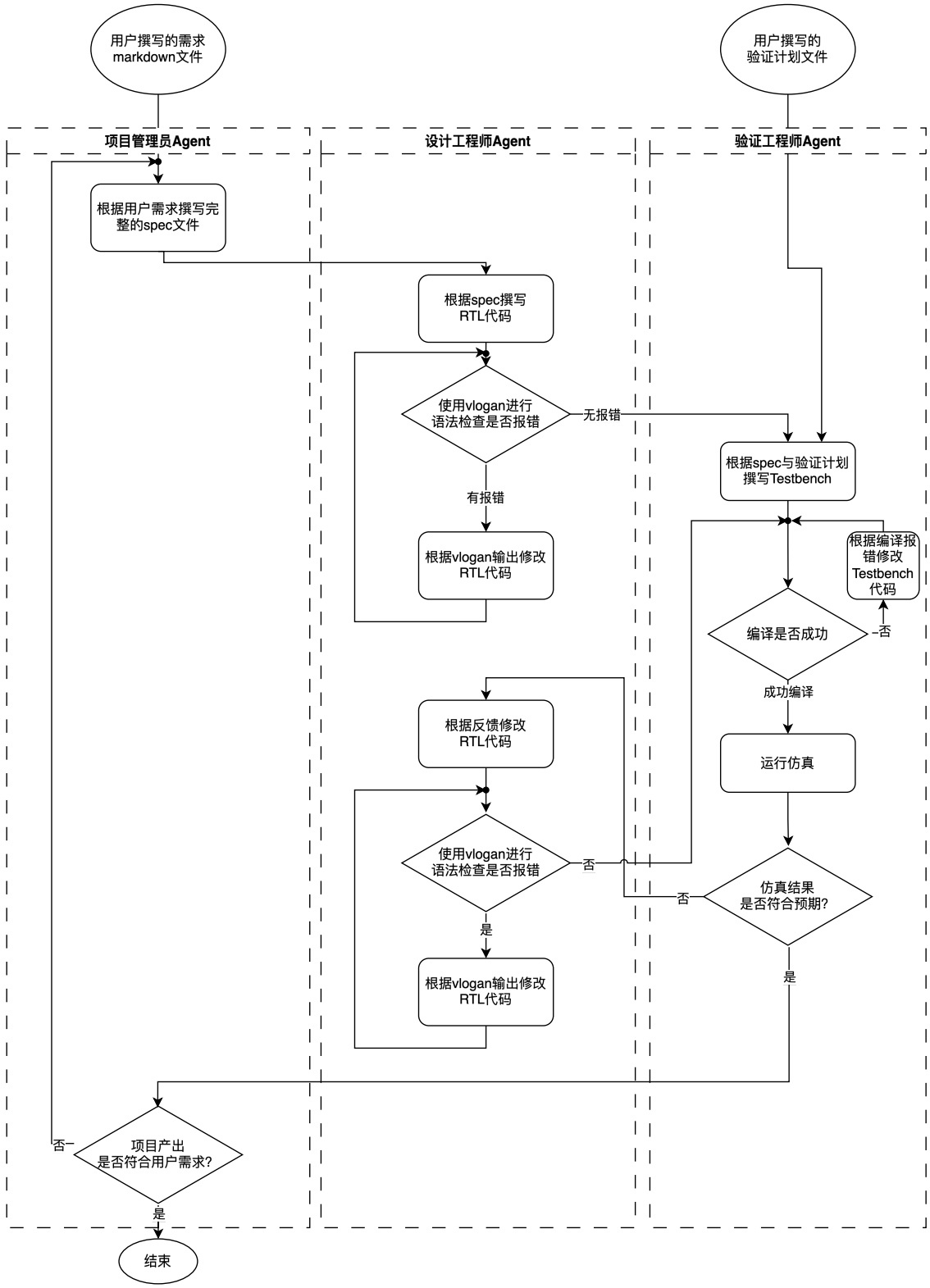
用户撰写的需求
markdown文件
用户撰写的
验证计划文件
项目管理员Agent
设计工程师Agent
验证工程师Agent
根据用户需求撰写完
整的spec文件
根据spec撰写
RTL代码
使用vlogan进行
语法检查是否报错
无报错
有报错
根据vlogan输出修改
RTL代码
根据spec与验证计划
撰写Testbench
根据编译报
错修改
Testbench
代码
否
编译是否成功
成功编译
根据反馈修改
RTL代码
运行仿真
使用vlogan进行
语法检查是否报错
否
否
仿真结果
是否符合预期?
是
是
根据vlogan输出修改
RTL代码
否
项目产出
是否符合用户需求?
是
结束

图 1: 程序流程图

4 总结

本项目设计了一套由 LLM 驱动的 RTL 代码的生成与验证自动化软件，实现了 IC 设计流程的自动化，能够快速产出原型设计。然而，在设计中，我们发现，一般的通用大模型在数字电路相关任务上、尤其是验证任务上，存在相当大的局限性。要进一步减少本项目的迭代次数，提升设计效率，必须对大模型进行针对性的训练，以使其能够更好地理解数字电路的设计与验证流程。本项目产出的原型设计，也应由专业的设计工程师和验证工程师进行进一步的修改与验证，以确保其满足实际的设计需求。