摘　要

在 FPGA 的 CPU 电路设计过程中，高位数的乘法器，尤其是基于 RSIC-V 系列的精简指令集 CPU 设计过程中，16 位乃至 32 位的高位数 2 进制乘法一直以来是 CPU 设计过程中的难题。如何在设计单周期或是流水线型乘法器的同时不调用高级的 FPGA 电路 IP 核的同时，让乘法器的关键路径延迟最短一直以来是工程师们设计乘法器时需要解决的痛点。

本次课程设计结合了本人在参加 NSCSCC2025“龙芯杯”全国大学生计算机系统能力设计大赛当中的参赛经验，力图构建一个只使用基础逻辑门就可以搭建出来一个任意相同输入位数的二进制可交互的乘法器阵列单元的可视化 UI 界面，最终做到：可移植、可交互、上手简单、一键生成。生成的任意位数的二进制乘法器可以通过 ModelSim 或 vivado 的仿真随机测试，并且所有的乘法器单元均采用基础的“与或非”逻辑门构建而成，不调用任何高级的 vivado 的 IP 核。

而要实现一个仅采用基础逻辑门构建而成的高位数二进制运算乘法，一个不得不涉及到的难题就是 wallace 树乘法器压缩。因此，本课程设计也将围绕相同输入位数的乘法 wallace 树的可视化压缩来进行问题展开。基于压缩前的 booth 算法进行带符号运算的处理，以及到最后对压缩结果的大数相加加法器的编写，整个工程的完整性得以保障，乘法器的运算 IP 核才能完整实现。

关键词：乘法器、基础逻辑门、FPGA 电路设计、wallace 树

第 1 章 绪论

1.1 研究背景

在 FPGA 设计 CPU 电路过程中，如何在不调用高端 FPGA 的 IP 资源的情况下实现乘法运算的同时，设计出的乘法器的关键路径延迟能尽可能短缩短，一直以来是乘法器设计的需要解决的痛点。如今，有一套已经非常成熟的乘法运算体系已经被工程师们证实有效：先采用 booth 算法对带符号二进制数进行符号处理，再采用 wallace 乘法树对部分积单元进行压缩，最后再采用并行前缀加法器对 wallace 树的压缩结果进行加法操作得到最终结果。但 wallace 树的压缩过程若是要采取基础逻辑门进行压缩的话，就不得不涉及到采用半加器（HalfAdder）与全加器（FullAdder）来对 wallace 树进行不用方式的圈划。而对于一些位数较多较高的乘法器，例如 16×16 位，32×32 位，压缩过程中的随机性太强，压缩的方式与过程太多太繁杂。如果没有一个可以视觉交互的压缩器来进行有条理的压缩交互操作，整个乘法器 IP 设计将会变得非常困难。

1.2 项目目标

本次课程设计的目标即为：设计一个可交互的 wallace 树圈划 UI 界面。用户在界面中利用半加器、全加器对 wallace 树进行手动布线圈划的同时，前端 UI 界面可以根据用户的操作过程记录用户的操作行为并传送到后端，后端则根据用户的圈划行为自动生成对应的 verilog 脚本语言代码，做到：一键生成、一键打包。

1.3 开发工具与技术栈（Python + QT + 通信库）

VScode、ModelSim、vivado

第 2 章 乘法器理论与实验设计

2.1 实验涉及的核心理论

二进制乘法

以简单的无符号 4×4 位二进制乘法为例：1010×1101，如下表所示

	高位						低位
0				1	1	0	1
1			1	1	0	1	
0		1	1	0	1		
1	1	1	0	1			

和 10 进制乘法及其类似，乘数按照不同权重的位展开，被乘数则作为每一行的系数权重，若为 1 则保留，若为零则清零，如下表所示：

	高位						低位
0				0	0	0	0
1			1	1	0	1	
0		0	0	0	0		
1	1	1	0	1			

然后再将每一列的数进行加法操作，即可得出结果：

	高位						低位
1				1	1	0	1	
1		1	1	0	1			
结果	1	0	0	0	0	0	1	0

而所谓的采用半加器、全加器进行圈划压缩，实际上就是不需要等待低位的进位信号，而是直接采用加法器事先对所有位进行压缩圈划，如下表所示，每一个方框都是一个加法器。一个加法器会产生一个结果（与被加数同权重）和一个进位（高一个权重）：

高位						低位
			X	X	X	X
		X	X	X	X	
	X	X	X	X		
X	X	X	X			

将同一权重的二个或三个操作数位进行半加器、全加器运算进行压缩，会得到一个结果和一个进位。再对结果和进位进行新一轮的半加器、全加器的压缩，以此类推将整个 wallace 树最终压缩成二路输出。如下表所示：

高位						低位
			S2	S1	S0	X
		S3	C1	C0		
	S4	C2				
X	C3		X			

高位							低位
	C2’	S2’	S1’	S0’	S1	S0	X
	X	C1’	C0’		C0		

如上表所示，压缩为两路输出即为 wallace 树的最终目的，后续的并行前缀加法器将会对这二路输出进行加法操作得到最终的运算结果。

全加器的行为逻辑：

```
module   FullAdder(a, b, cin, sum, cout);
input    a, b, cin;
output   sum, cout;
assign   sum = a ^ b ^ cin;
assign   cout = (a & b) | (a & cin) | (b & cin);
```

```
endmodule
```

半加器的行为逻辑：

```
module HalfAdder(a, b, sum, cout);
input a, b;
output sum, cout;
assign      sum = a ^ b;
assign      cout = a & b;
endmodule
```

2.2 实验模块划分

Booth 算法、Wallace 树、Koggle-Stone 并行前缀加法器、最终整合 TopMutiplier

2.3 实验参数设计

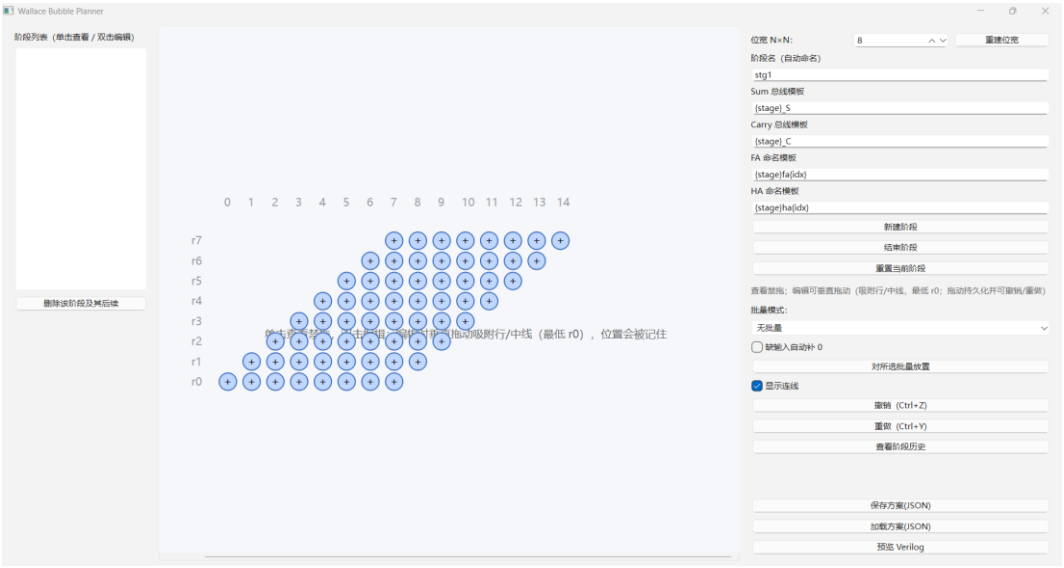
第 3 章 系统设计与实现

3.1 系统架构设计

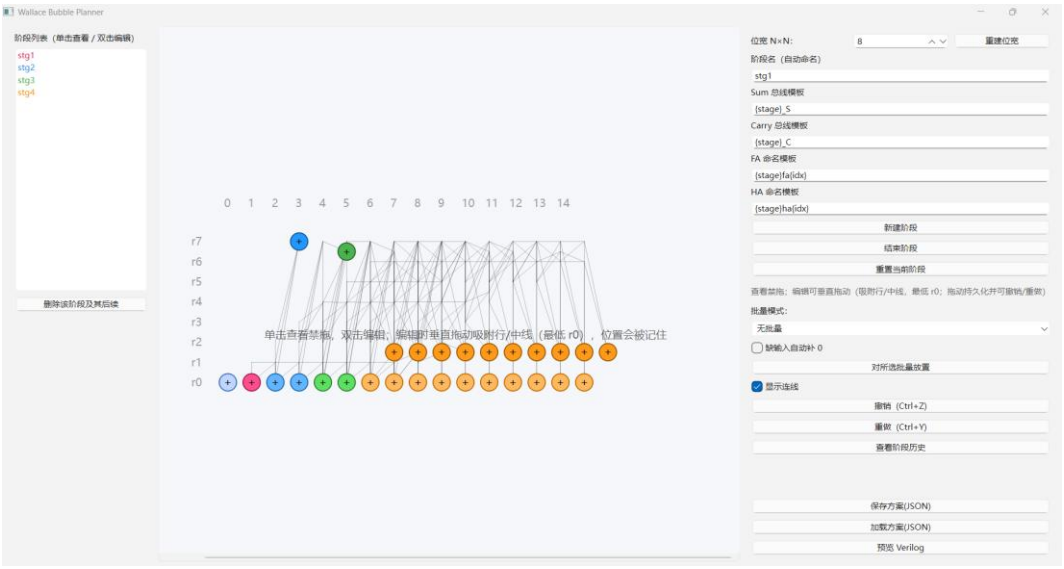
1、前端（QT 界面框架设计）

主界面如下：

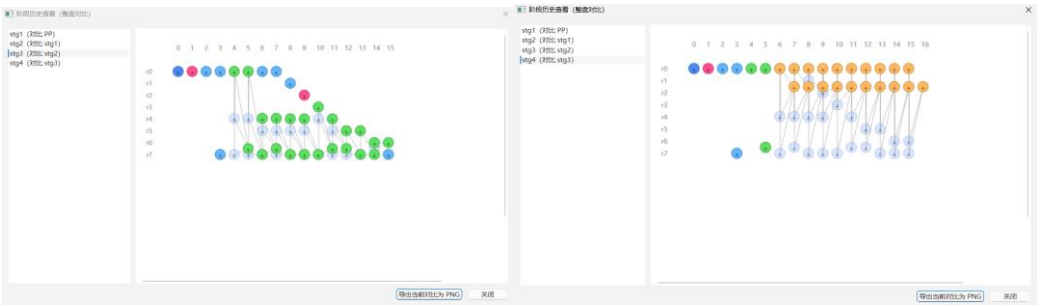
打开 Wallace 规划器后界面如下：



选择加载方案（预先已经圈划好）后查看、编辑等功能如下：

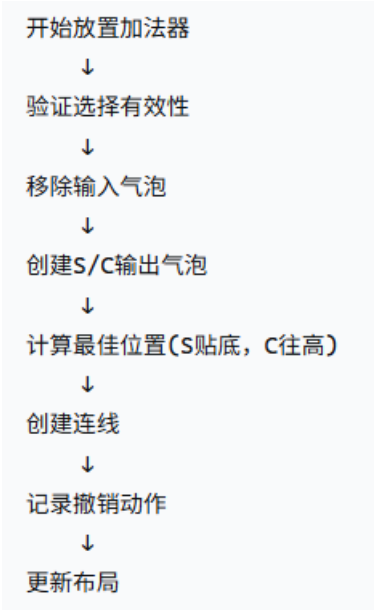


查看历史记录界面：

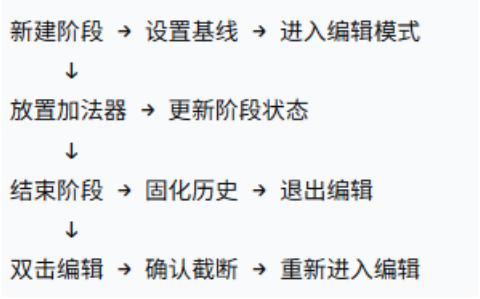


2、后端（Python 通信算法框架/流程图/结构设计）

加法器放置流程图



阶段管理流程图



历史重算流程图

从PP开始重建
↓
逐阶段回放加法器
↓
采集节点和连线状态
↓
生成结构化历史记录
↓
保存完整状态序列

3.2 核心算法实现

1、booth 算法（以 8 位 booth 为例）：

```
def gen_booth_module(N: int, module_name: str) -> str:
    # 扩展位宽: N+1位
    M = N + 1

    # 关键算法: 2-bit窗口编码
    for i in range(N):
        # 根据相邻2位决定部分积值
        # tmp[i+1:i] = 01 → +x
        # tmp[i+1:i] = 10 → -x
        # 其他 → 0
        lines.append(f"assign pp{i}_ext = (tmp[{i+1}:{i}] == 2'b01) ? plusx_in :")
        lines.append(f"                   (tmp[{i+1}:{i}] == 2'b10) ? negx_in  : {M}'sd0;")
```

2、并行前缀加法器（以 8 位为例）

```
def gen_adder_module(N: int, module_name: str) -> str:
    W = 2 * N  # 输出位宽

    # 关键算法: 进位生成传播
    s.append(f"  wire [{W-1}:0] g = a & b;")  # 生成位
    s.append(f"  wire [{W-1}:0] p = a ^ b;")  # 传播位
    s.append(f"  wire [{W}:0]   c;")          # 进位链

    # 递推公式: c[i+1] = g[i] | (p[i] & c[i])
    s.append("    for (i = 0; i < {W}; i = i + 1) begin : GP")
    s.append("      assign c[i+1] = g[i] | (p[i] & c[i]);")
    s.append("      assign sum[i] = p[i] ^ c[i];")
```

3、wallace 核心算法

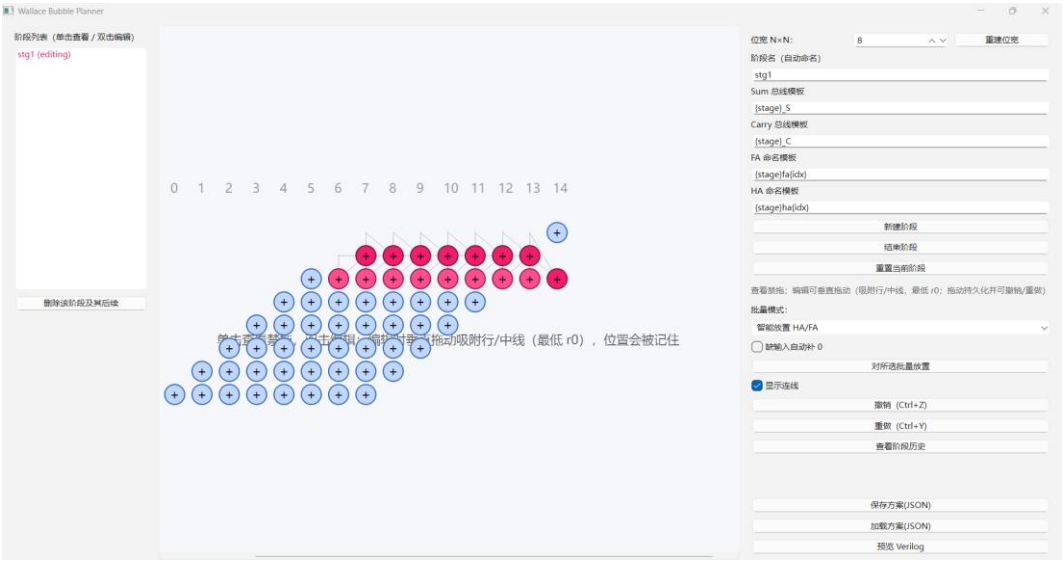
```python
def apply_stage(columns: List[List[str]], st: Stage):
    """Wallace树阶段压缩算法"""

    for ad in st.adders:
        # 1. 输入选择：从指定列取2/3个操作数
        if ad.inputs:
            inps = list(ad.inputs)
            # 移除已使用的操作数
            for token in inps:
                try: columns[col].remove(token)
                except ValueError: pass
        else:
            # 自动从列顶取操作数
            take = [columns[col].pop() if columns[col] else "1'b0" for _ in range(need)]

        # 2. 加法器实例化（HA/FA）
        if typ == "FA":
            inst_lines.append(f"FullAdder {inst} ( {a}, {b}, {cin}, {s_bit}, {c_bit} );")
        elif typ == "HA":
            inst_lines.append(f"HalfAdder {inst} ( {a}, {b}, {s_bit}, {c_bit} );")
```

3.3 QT 界面设计细节

1、交互控件设计

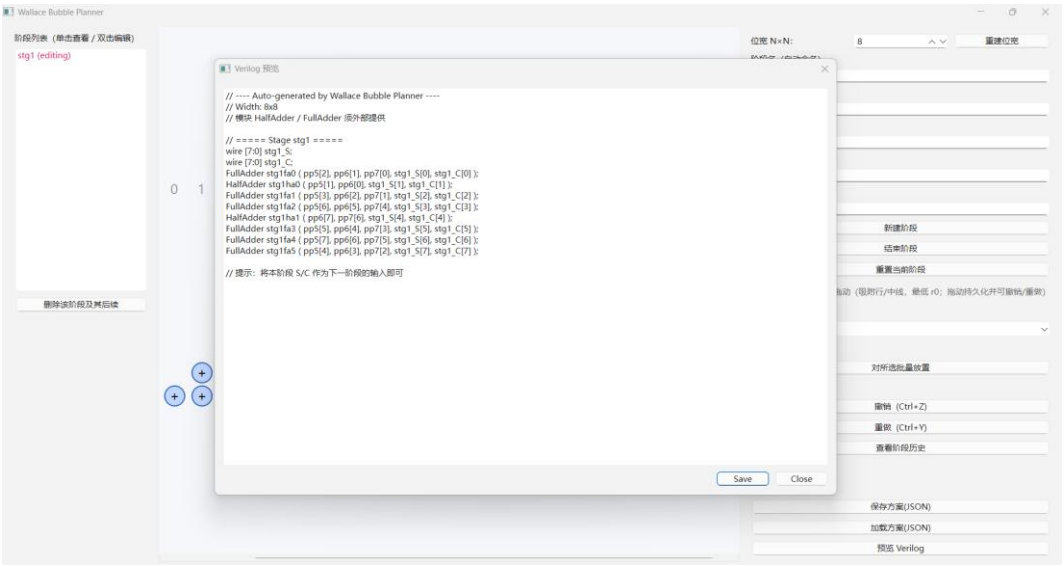
用鼠标拖动批量选择数据后如下图所示：

对选择的数据点击智能放置 FA、HA 后可以得到如下效果：



2、数据可视化方案

预览 verilog 代码功能如下图所示



第 4 章 系统测试与应用

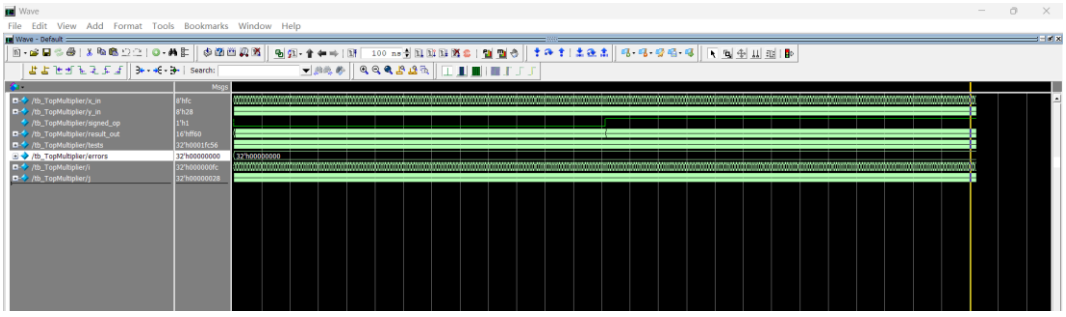
4.1 测试用例设计

基于 8 位的乘法器的 verilog 乘法 IP 测试文件如下所示：

```
Ln#
  1
  2   `timescale 1ns/1ps
  3   module tb_TopMultiplier;
  4
  5     // DUT ??
  6     reg  [7:0]  x_in;
  7     reg  [7:0]  y_in;
  8     reg         signed_op;     // 1=???, 0=???
  9     wire [15:0] result_out;
 10
 11     // ??? DUT
 12     TopMultiplier dut (
 19     // ??
 20     integer tests  = 0;
 21     integer errors = 0;
 22
 23     // ? for ??????????????????????
 24     integer i, j;
 25
 26     // ????????? automatic?? 2001 ???
 27     task check_case(input [7:0] x, input [7:0] y, input sop);
 57     // ??????
 58     task directed_tests(input sop);
 59     begin
 95     endtask
 96
 97     // ?????? $random??? 8 ??
 98     task random_tests(input sop, input integer count);
110     initial begin
149   endmodule
150
```

4.2 典型实验结果截图与分析（附输出波形/数据对比）

```
ModelSim> vsim -gui -novopt work.tb_TopMultiplier
# vsim -gui
# Start time: 15:09:59 on Oct 09,2025
# ** Warning: (vsim-8891) All optimizations are turned off because the -novopt switch is in effect. This
ebug or PLI features please see the User's Manual section on Preserving Object Visibility with vopt.
#
# Refreshing D:/Users/15432/Desktop/mulgen/work.tb_TopMultiplier
# Loading work.tb_TopMultiplier
# Loading work.TopMultiplier
# Refreshing D:/Users/15432/Desktop/mulgen/work.Booth8
# Loading work.Booth8
# Loading work.WallaceTop8
# Loading work.FullAdder
# Loading work.HalfAdder
# Refreshing D:/Users/15432/Desktop/mulgen/work.CS_Adder16
# Loading work.CS_Adder16
VSIM 2> run
# === tb_TopMultiplier (N=8) start ===
VSIM 3> run -all
# === PASS: 131118 tests, 0 errors ===
# ** Note: $finish    : D:/Users/15432/Desktop/mulgen/mul_tb.v(146)
#    Time: 131118 ns  Iteration: 0  Instance: /tb_TopMultiplier
```



生成的 ip 核通过了 13 万次测试，测试结果均符合预期。

第 5 章 总结与展望

5.1 项目成果总结

本项目将高位数乘法器设计过程中较为繁杂的 wallace 树圈划过程采用可视化可交互的方式让用户自主设计，设计的方法方式非常科学的同时也非常的便捷。

5.2 创新点

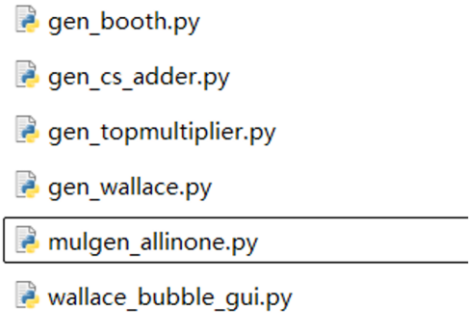
本项目将所有的乘法器生成交互功能打包成了一键生成的 exe 应用程序，在 PC 端双击该程序可一键执行生成所有有关乘法器的 ip 核部件。

5.3 不足与改进方向

不足之处在于：若输入的乘法操作数位数不相同时，乘法器的设计该如何修改，本次项目没有进行涉及。

附录（使用方法）

1. 将程序源码中的内容分别拷贝至下面 6 个文件中(请依照注释内容完整拷贝)，并保存在同一文件夹目录下。



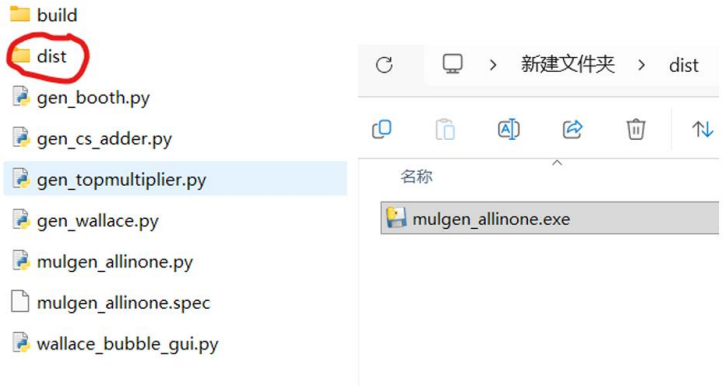
2. 用 WindowsPowerShell 打开当前文件夹，运行以下命令：(需自行安装 python)

   py -m PyInstaller -F -w --clean mulgen_allinone.py

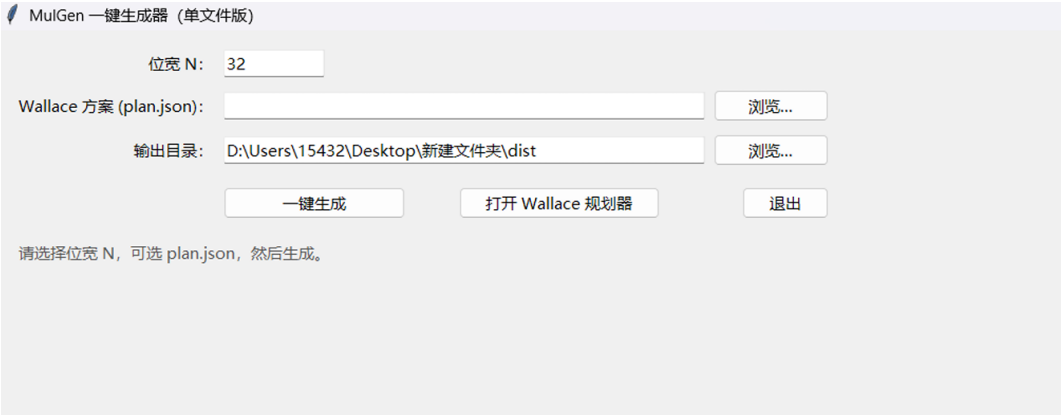
3. 运行成功后，会得到以下类似输出：

```
26350 INFO: Extra DLL search directories (AddDllDirectory): ['C:\\Users\\15432\\AppData\\Roaming\\Python\\Python312\\site-packages\\shiboken6']
26351 INFO: Extra DLL search directories (PATH): ['C:\\Users\\15432\\AppData\\Roaming\\Python\\Python312\\site-packages\\PySide6']
27400 INFO: Warnings written to D:\Users\15432\Desktop\新建文件夹\build\mulgen_allinone\warn-mulgen_allinone.txt
27411 INFO: Graph cross-reference written to D:\Users\15432\Desktop\新建文件夹\build\mulgen_allinone\xref-mulgen_allinone.html
27465 INFO: checking PYZ
27465 INFO: Building PYZ because PYZ-00.toc is non existent
27465 INFO: Building PYZ (ZlibArchive) D:\Users\15432\Desktop\新建文件夹\build\mulgen_allinone\PYZ-00.pyz
27582 INFO: Building PYZ (ZlibArchive) D:\Users\15432\Desktop\新建文件夹\build\mulgen_allinone\PYZ-00.pyz completed successfully.
27611 INFO: checking PKG
27612 INFO: Building PKG because PKG-00.toc is non existent
27612 INFO: Building PKG (CArchive) mulgen_allinone.pkg
37976 INFO: Building PKG (CArchive) mulgen_allinone.pkg completed successfully.
37987 INFO: Bootloader C:\Users\15432\AppData\Roaming\Python\Python312\site-packages\PyInstaller\bootloader\Windows-64bit-intel\runw.exe
37987 INFO: checking EXE
37987 INFO: Building EXE because EXE-00.toc is non existent
37987 INFO: Building EXE from EXE-00.toc
37989 INFO: Copying bootloader EXE to D:\Users\15432\Desktop\新建文件夹\dist\mulgen_allinone.exe
37995 INFO: Copying icon to EXE
38121 INFO: Copying 0 resources to EXE
38121 INFO: Embedding manifest in EXE
38837 INFO: Appending PKG archive to EXE
38946 INFO: Fixing EXE headers
44380 INFO: Building EXE from EXE-00.toc completed successfully.
44391 INFO: Build complete! The results are available in: D:\Users\15432\Desktop\新建文件夹\dist
PS D:\Users\15432\Desktop\新建文件夹>
```

4. 在生成的 dist 文件夹中双击打开生成的 exe 程序：



5. 进入如下界面，开始操作。



6. 打开 wallace 规划器，新建方案，当 wallace 树圈划至两行后，将该方案保存并加载，并点击一键生成，生成的.v 文件即可应用于 vivado 或 ModelSim 中进行仿真与调试。