# AutoWire 使用说明

## 简介

AutoWire 是一个基于 pyverilog 的 Verilog SOC 自动连线工具，用于简化大型 SOC 设计中模块间的连线工作。它可以根据 YAML 配置文件自动完成模块实例化、端口连接、协议总线连线等工作，大幅减少手动连线的工作量和出错率。

## 特性

1. 支持宏定义处理

2. 支持参数化

3. 支持多次实例化同一模块

4. 支持端口类型识别（input/output/inout）

5. 支持二次表达式解析（如拼接、位宽选择、组合逻辑等）

6. 支持二维数组端口（如 input [3:0][7:0] data\_in）

7. 支持冲突检测与log输出

8. 支持多种总线协议的批量连线（如 AHB、AXI、APB等）

## 连线原理

1. 工具会为不同模块的端口创建一个集合，集合中同名的输入输出进行自动连线。

2. 可通过手动(批量)更改使想连接的端口同名以适配自动连线

3. 当集合中的端口只有输入，或只有输出，会自动扩展为top端口

4. 优先级: 手动连线(`connections`)  > 协议连线(`bundle\_con`) > 自动连线

## 基本用法

```bash

python autowire.py -i vcn.yaml -o output/

```

如需调试可增加 '-d' 参数，输出更详细的日志信息：

```bash

python autowire.py -i vcn.yaml -o output/ -d

```

## vcn.yaml 配置文件详解

# ============================================

# 1. 顶层模块名称（必选）

# ============================================

top\_module: soc\_top

# ============================================

# 2. 全局宏定义文件（可选）

# ============================================

# 如果 RTL 文件中使用了 `define，需要在这里指定定义文件

define\_files:

  - ./rtl/define.vh

  - ./rtl/global\_define.vh

# ============================================

# 3. RTL 文件路径（必选）

# ============================================

# 列出所有需要实例化的模块的 RTL 文件

rtl\_path:

  - ./rtl/cpu\_core.v

  - ./rtl/uart\_controller.v

  - ./rtl/irq\_controller.sv

# ============================================

# 4. 模块实例化配置（必选）

# ============================================

instances:

  # 基本实例化（无参数）

  - module: cpu\_core              # 模块名（必须与 RTL 中的 module 名匹配）

    name: u\_cpu                   # 实例名（自定义）

  # 带参数的实例化

  - module: uart\_controller

    name: u\_uart

    parameters:                   # 参数列表（可选）

      BAUD\_RATE: 9600            # 参数名: 参数值

      DATA\_BITS: 8

      PARITY\_EN: 1

  # 多个相同模块的实例

  - module: irq\_controller

    name: u\_irq\_0

  - module: irq\_controller

    name: u\_irq\_1

# ============================================

# 5. 手动连线配置（可选）

# ============================================

# 格式：实例名.端口名 : 目标信号/表达式

connections:

  # 基本连线：端口 → wire

  u\_cpu.clk: sys\_clk

  # 连接到常量

  u\_uart.hsel\_ahb: 1'b1

  # 信号拼接

  u\_cpu.irq: "{6'b0, irq[0], uart\_irq}"

  # 位宽选择

  u\_cpu.test\_in: irq[1:0]

# 信号组合逻辑

  u\_cpu.test\_in1: "app\_test\_in & app\_test\_en"

  # 输出悬空

  u\_uart.tx\_data:

# ============================================

# 6. 协议总线连线（可选）

# ============================================

# 用于批量处理标准协议接口

bundle\_con:

  # 格式 1：列表格式（推荐）

  - ahb:                          # 协议名（必须在 bundle.yaml 中定义）

      u\_cpu.a\_\*: cpu\_ahbm\_\*       # 实例.端口模式 : 目标信号模式

      u\_uart.\*\_ahb: cpu\_ahbm\_\*

  - axi:

      u\_dma.m\_axi\_\*: ddr\_axi\_\*

      u\_ddr\_ctrl.s\_axi\_\*: ddr\_axi\_\*

# ============================================

# 7. 外扩到TOP输出信号（可选）

# ============================================

# 把内部的wire拓展到top的输出端口

top\_add:

  - rst\_clk\_hsi

## 常见问题

# Q1: 端口连接冲突怎么办？

A: 手动连线优先级最高，会覆盖协议和自动连线。

# Q2: 如何确认协议连线是否生效？

A: 使用 `-d` 调试模式，查看日志中的协议匹配信息。

# Q3: 怎么增加总线协议类型？

A: 在boundle.yaml按原有格式增加