用户指南

vtags 是一款在 gvim 下实现类似 verdi 的信号追踪、显示拓扑等功能的插件。vtags 插件完全使用 python 实现，实现功能包括信号追踪、宏定义追踪、显示模块拓扑、快速打开文件、保存和打开快照、添加断点等功能。

支持功能和快捷键:

快捷键	功能
	创建 vtags.db
vtags	在当前目录对所有当前目录及子目录的设计文件建立 vtags data base
-v <design_file>	指定单独文件建立 vtags data base
-f <design_file_list>	指定文件列表建立 vtags data base（可以直接使用 vcs 或 verdi 的 filelist）
+vtags_incdir+ <dir_path>	将 dir_path 及其子目录所有 rtl 文件添加到 vtags data base
+incdir+ <dir_path>	同 vcs/verdi 作用相同，添加 include file 搜寻目录
	Gvim/VIM 窗口内功能
gi	进入子模块
gu	回到上层模块
mt	从顶层到当前模块的调用结构【3.00 +】
ct	清空 vtags 记录的模块调用历史【3.00 +】
<Space><Left>	追信号源，或宏定义
<Space><Right>	追信号目的
<Space><Down>	回退
<Space><Up>	向前
<Space> + v	显示侧栏导航，和展开收回侧栏条目 动态加载 vtags,在当前文件打开 dir 动态创建 vtags.db【3.00+】
<Space> + c	添加记录点
<Space> + b	添加基本模块
<Space> + d	删除记录点或基本模块
<Space> +h	固定当前窗口
<Space>	快速访问: 1. 侧栏模块 2. Report 结果 3. mt 结果跳转到对应模块【3.00+】
<Space> + s	储存快照
<Space> +r	在当前窗口回复储存快照【3.00+】

<Space> + q	快速关闭所有窗口【3.00+】
	Gvim/VIM 打开时功能
gvim/vim <vtags.db path>	加载快照储存在 vtags.db 下的快照【3.00+】
	基于 vtags.db 的离线功能
vtags -db <vtags.db path> “mtrace(<module_name>)”	
在<vtags.db path>所在设计中，列出<module_name> 从顶层到该模块的所有调用结构	
vtags -db <vtags.db path> “mfilelist(<module_name>)”	
在<vtags.db path>所在设计中，列出<module_name> 调用模块的 filelist	
vtags -db <vtags.db path> “mtopo(<module_name>, depth, mask)”	
在<vtags.db path>所在设计中，列出<module_name> 调用模块 topo	
depth 可选参数，列出 topo 深度	
mask 可选参数，模块调用次数多余 maks 数值不一一列出	
vtags -db <vtags.db path> “mopen(<module_name>)”	
在<vtags.db path>所在设计中，打开模块<module_name>文件	

详细信息看《二：开始使用及支持功能和快捷键》

vtags 安装：

（1）下载插件代码，解压并复制到安装路径下。（vtags 无需安装，解压即可）

tar -zxvf vtags-3.00.tar.gz // 解压 vtags 压缩包，假设目前在用户根目录下，则安装路径为： ~/vtags-3.00/

（2）在 linux 配置文件中添加别名。

1）使用 cshrc 的用户：

~/.cshrc 中添加：alias vtags 'python ~/vtags-3.00/vtags.py'

使用 bashrc 的用户：

~/.bashrc 中添加：alias=vtags 'python ~/vtags-3.00/vtags.py'

2）source ~/.cshrc 或 source ~/.bashre 使设置生效。

注： 使用别名的目的是为了在任何位置都可以，直接输入 vtags 来调用安装目录下的 vtags.py 脚本。 所以能达到这个目的的方式都可以，也可以直接通

过全局路径调用脚本。

（3）在 vim 配置文件中加入插件。

~/.vimrc 中添加： source ~/vtags-3.00/vtags_vim_api.vim

注：在 vtags_vim_api.vim 中除了加载脚本外，还可以修改所有支持操作的快捷键。

(4) 本步骤可选， 可以在 vtags-3.00/vim_glb_config.py 中设置插件的一些全局设置，你也可以使用 vtags 时生成的局部配置文件 vim_local_config.py 中进行设置，详细配置见下文。

vtags 使用：

一： 生成 vtags.db 数据结构

完成安装后，vtags 的使用和 ctags 类似，首先需要生成对应的数据结构。

（1）cd 到代码所在目录。

（2）执行命令：vtags (得益于前面设置的别名，其实是在代码目录下调用 ~/vtags-3.00/vtags.py 脚本)

注：（1）执行 vtags 生成数据文件，主要分 3 步。

第一步：找到当前目录及子目录下所有支持的 verilog 文件（各自代码使用的 verilog 后缀可以在配置文件中添加，默认只支持.v 文件，可以在 config 中任意添加支持的 verilog 后缀）

第二步：分析所有 verilog 文件找到所有 verilog module。

第三步：分析所有 verilog 文件找到每个模块中子模块调用信息。

（2）vtags 执行完成后会在当前目录下生产 vtags.db 文件夹，里面存放的是前面分析的数据，以及当前目录 vtags 的局部配置文件，配置详见下文。

二： vtags vim config

配置文件：

vtags 的配置可以在两个位置配置，全局配置和局部配置。

一、全局配置：

在 vtags 的安装目录下，在 vtags_glb_config.py 中添加配置。该配置是所有使用 vtags 的默认配置。

二、局部配置：

在 code 文件夹使用 vtags 生成 vtags.db 后在 vtags.db 中会生成一个局部配置文件 vtags_local_config.py。

当使用 gvim 打开文件时，会首先选择局部配置中的配置，如果没有局部配置才会使用全局的默认配置。

可配参数：

frame_window_width = 30

设置 gvim 窗口最左侧 sidebar 的宽度，该窗口主要显示模块拓扑，checkpoint 和 basemodule

report_window_height = 10

设置 gvim 窗口最底部 report window 的高度，该窗口主要显示提示信息和信号追踪结果。

max_open_work_window_number = 1

当使用 vtags 机制（如追踪到新的文件，topo 打开新的文件...）打开新的文件时，打开窗口的最大数量，当已经打开的窗口数量到达该值时，会自动关闭一个最早打开的文件窗口在打开新的文件。

max_roll_trace_depth = 1000

使用“空格 + 前/后方向键”能够倒退/前进的最大深度。

max_his_check_point_num = 1000

使用“空格 + c” 能够保存的最近最多 checkpoint 个数。

base_module_threshold = 200

在生成 code 的数据文件 vtags.db 时，对所有模块如果该模块在整个文件夹下的 code 中被实例化的次数到达该参数的数量时，将该模块设置为 basemodule 当使用“空格 + v”显示模块拓扑时不显示该 basemodule 模块的调用细节，只显示模块名+实例次数。

support_verilog_postfix = ['v']

vtags 将当前目录下所有后缀在 support_verilog_postfix 中的文件认为可识别的 verilog 文件。默认只有‘v’，可以自己添加，如： support_verilog_postfix = ['v', ‘V’， ‘sv’]等

debug_mode = False

该模式是开发过程用于调试的选项，打开后会在 gvim 运行过程中，在 vtags.db 中生成运行过程的 log，会减慢执行速度，不建议打开。

dynamic_update_vtags_db = True

当文件修改时动态更新 vtags db