ADS 综合用户手册

（Version 2.2）

深圳市紫光同创电子有限公司

文档版本修订记录

版本号	发布日期	修订记录
V1.0	2022.07.12	初始版本
V2.0	2022.09.21	添加 ADS 综合 Compile 配置选项“Multiple File Compilation Unit”说明
V2.1	2022.10.25	更新 NV 相关描述
V2.2	2022.12.01	添加 full_case,parallel_case,translate_off_on，syn_state_machine 相关描述

目录

图目录

1　ADS Flow 概述

ADS(Architecture-Driven Synthesis)是一款实现 design source 文件逻辑综合的工具软件。ADS 主要包含 compile 和 synthesize 两大流程，compile 阶段完成 source code RTL 网表的生成，synthesize 阶段基于 compile 的 DB，mapping 和 optimization 后生成 technology 网表。

ADS Flow 完成后，综合工具输出 Technology 网表文件(.vm)等，生成的 vm 网表文件可以用来使用第三方验证工具做验证工作。compile 和 synthesize 之间通过 DB 文件进行网表信息传递，逻辑综合的结果也是通过生成 DB 文件将网表信息传递给 DeviceMap。

下图为 ADS 综合工具实现流程，ADS 综合工具暂时只支持 verilog。

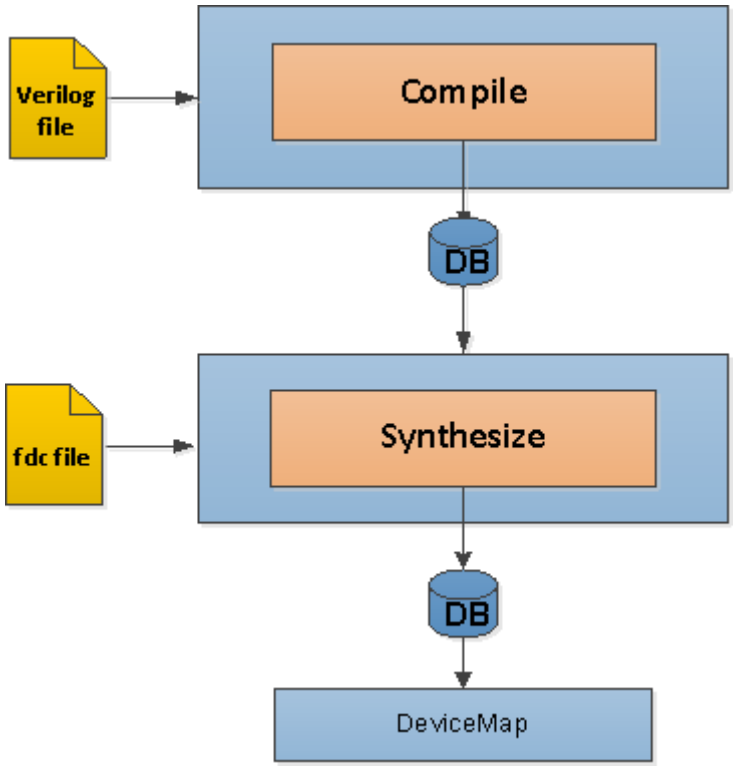

图 1-1 ADS Synthesis Design Flow

ADS 综合完成后，在工程目录生成的过程文件有 run.log，用于存放全部 log 信息。在工程文件所在目录中会生成两个子目录，分别为 compile 和 synthesize。

compile 目录中存放 compile 阶段的结果，主要文件为<top_module>_rtl.adf，该文件可用于 RTL 原理图的显示，也是 synthesize 阶段输入的 DB 文件。

synthesize 目录中存放 synthesize 阶段的结果，主要文件有 <top_module>_syn.adf ， <top_module>_syn.vm ， <top_module>.snr 等 。 <top_module>_syn.adf 可用于 Technology 原理图的显示。vm 文件为 map 之后的 GTP 网表，可用来做第三方工具验证。snr 文件为综合后的资源报告和时序报告，该文件的内容都包含在 run.log 中。

在 compile 和 synthesize 阶段都会生成 formal.pvf 文件，用于指导形式化验证。

2 ADS 综合的基本操作

2. 1 ADS 综合的实现流程

工程文件建好后，进入主界面，如果综合工具不是 ADS，可以切换至 ADS 综合工具。在工程管理区如图 2-1 中红色区域鼠标右击选择 Project Setting，会弹出 Project Setting 窗口，Synthesize tool 选择 ADS，点击 OK 完成设置，如图 2-2 所示。

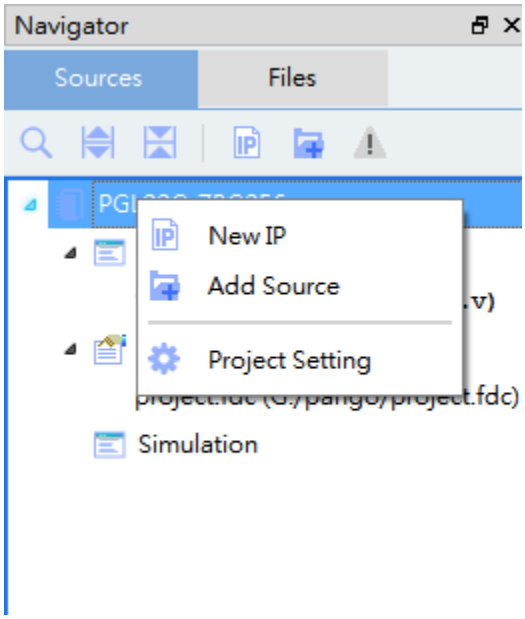

图 2-1 ADS 综合工具设置菜单

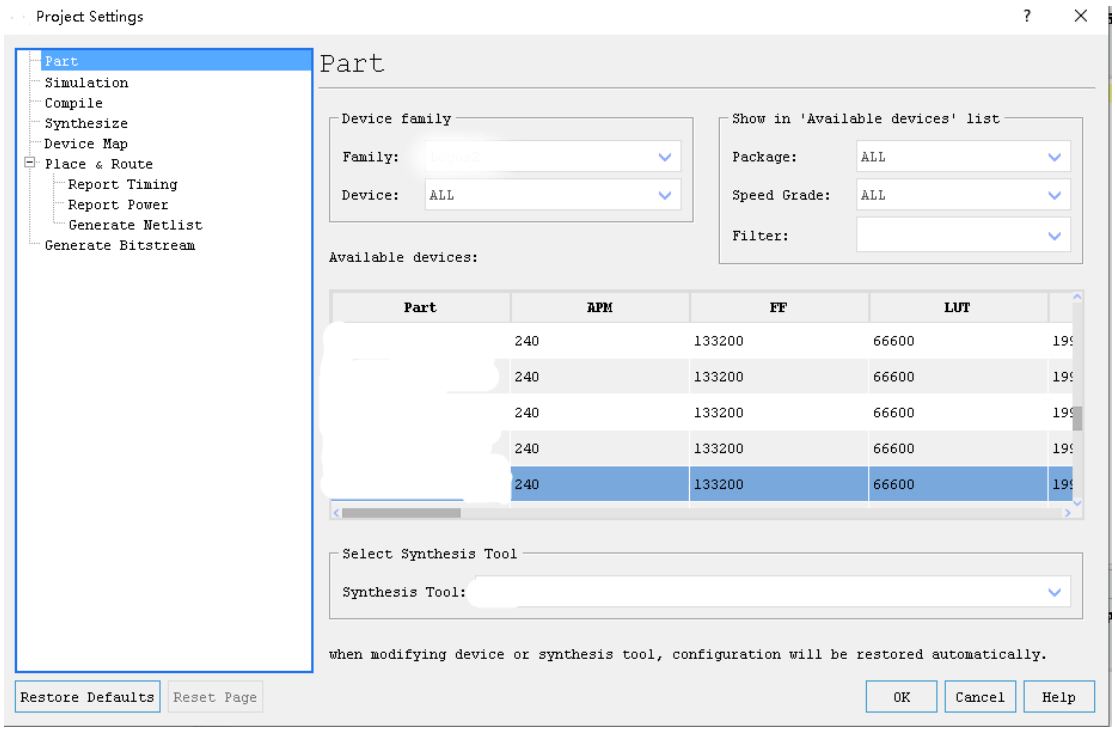

图 2-2 ADS 综合工具设置窗口

Project Setting 设置为 ADS 后，Flow 的流程的显示如图 2-3 所示。ADS Flow 包含 Compile 和 Synthesize 两部分，两个 action 可以单独运行。如只执行 Compile，在 Flow view 中的 Compile 处鼠标右击，点击 Run 即可执行，执行完成后生成 RTL 网表。

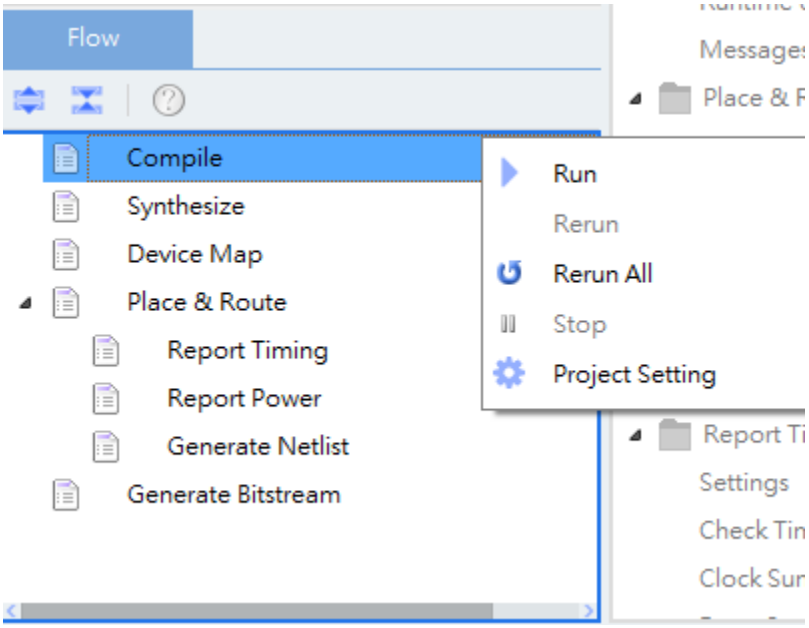

图 2-3 ADS Flow

Compile 和 Synthesize 执行成功后如图 2-4 所示，对应 action 的左侧会显示 ，失败显示 。

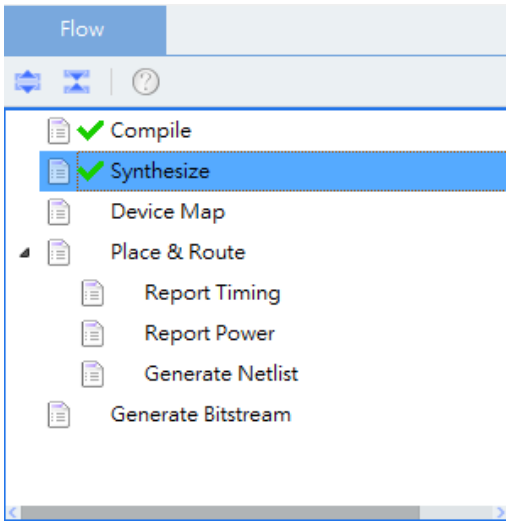

图 2-4 ADS 综合完成后 Flow 界面

图 2-5 可以看到 ADS 综合 Report 的链接已经可以点击打开，鼠标单击 Synthesize/Settings 看到如下界面。

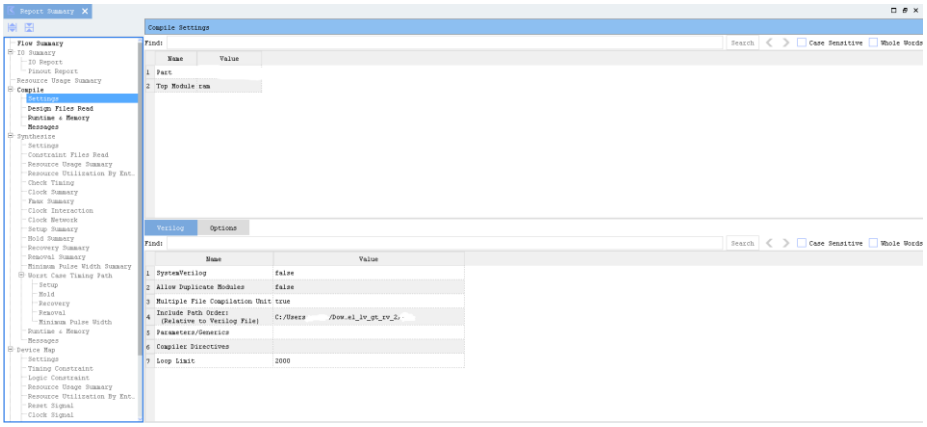

图 2-5 ADS Report Summary

综合报告的目录下，有一个后缀为*.ccr 的文件，里面是综合在 check fdc 约束时产生的所有约束相关的报告信息。

2. 2 ADS 添加源文件

1、ADS 综合流程支持的 design 文件包括：纯 verilog 文件、verilog 文件和 ADS 综合后的中间文件(*_syn. adf)混合使用等两种。

2、ADS 综合流程支持的约束包括 fdc、scf、lcf 文件，fdc 作用于 synthesize 阶段，scf 和 lcf 作用于 dev_map 阶段。

添加文件如下图 2-6 所示：

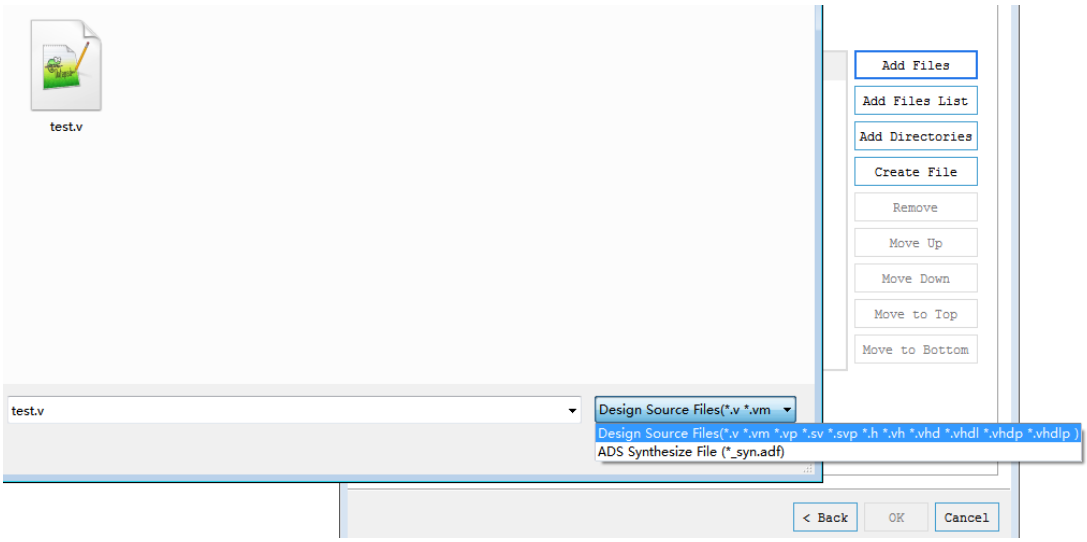

图 2-6 ADS 添加源文件

2.2.1 纯 verilog 文件作为源文件

ADS 目前支持 verilog 文件作为源文件，可以通过图 2-1 中所示的 add source 按钮来进行添加。

2.2.2 Verilog 文件和 adf 文件混用作为源文件

ADS 流程还支持 verilog 文件和 ADS 综合后的中间文件(*_syn. adf 文件)一起作为源文件。工程中使用此种方式作为 design 时，verilog 文件作为顶层使用；ADS 综合后的中间文件(*_syn. adf 文件)作为子层使用，其由原始工程中的子层 verilog 文件综合而来。sub module 的构造方式与. v 和 vm 混用一样，需要在顶层中使用 /*synthesis syn_black_box */ 属性标注 sub module。如下图所示：

```
module pgr_apb_mif_16bit(
    input               clk,
    input               rst_n,

    input       [23:0]  addr,
    input       [15:0]  data,
    input               we,
    input               cmd_en,
    output              cmd_done,

    output   reg   [7:0]  fifo_data,
    input               fifo_data_valid,
    output   reg           fifo_data_req,

    output  reg [23:0]  p_addr,
    output  reg [15:0]  p_wdata,
    output  reg         p_ce,      //p_sel
    output  reg         p_enable,
    output  reg         p_we,
    input               p_rdy,
    input       [15:0]  p_rdata
) /* synthesis syn_black_box */ ;
endmodule
```

图 2-7 black box module

*_syn. adf 文件是作为 sub module 来使用的，所以在综合生成 adf 时，需要在综合配置中勾选 disable I/O insertion 选项。如下图所示：

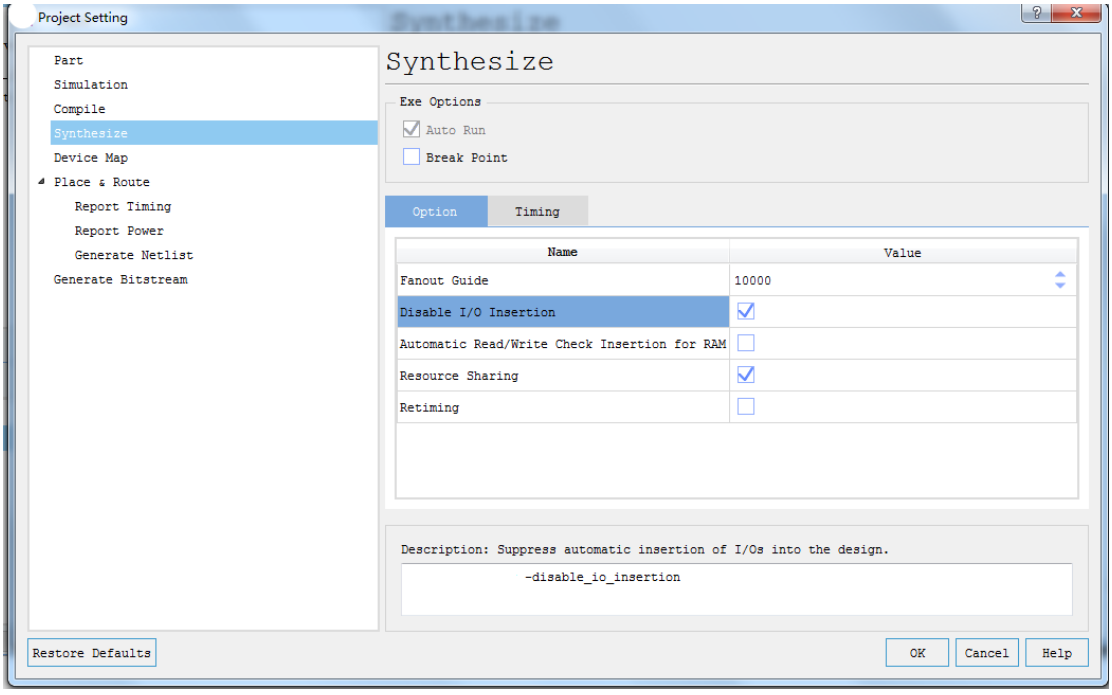

图 2-8 Disable I/O inserterion

例如：

现有顶层 top，子 module A, B。首先使用 ADS（需要在 synthezise 配置选项中勾选 disable I/O insertion 选项，见图 2-8）分别综合 A, B module，生成 A_syn. adf，B_syn. adf。

然后在顶层添加 A, B module 的声明(需增加/*synthesis syn_black_box */说明，见图 2-7)。

然后 design source 增加 A_syn. adf，B_syn. adf 文件。

最后可以开始综合顶层 top(注意综合配置中不要勾选 disable I/O insertion 选项)。此时子层 adf 文件作为黑盒是不会综合的，在 device map 时会将顶层综合结果与子层 adf 文件合并传递至布局布线阶段。

因为子层 adf 在综合时是作为黑盒使用，故此时无法得知子层 adf 中的 object 信息，故无法在 fdc 文件中对子层 adf 中的 object 做约束；dev_map 阶段时载入子层 adf 时，会将子层 adf 中的时序约束信息清空(top module 不受影响)；故引入 scf 文件来对子层 adf 做时序约束；同样的引入 lcf 文件来对子层 adf 做位置约束和属性约束；scf 和 lcf 文件中编写约束时，约束规则和 fdc 文件中是一致的，值得注意的是约束时 object 的层级结构与子层 adf 综合前的层级结构一致，故若要在 scf 或 lcf 文件中对子层 adf 做约束，需要知道约束对象综合前的层级结构。目前还不支持在 UCE 中直接对子层 adf 中的 object 做约束后保存到 scf 和 lcf 文件中，同样的打开 UCE 时也不能识别 scf 和 lcf 文件中的约束。若要在 scf 和 lcf 文件对子层 adf 做约束，目前仅支持在文件中手动编写约束。

2. 3 Project Setting 设置

Compile_settings 在 Flow 显示区域内点击任一 Flow 步骤的右键菜单中的 Project Setting 选项，进入如下图所示的 Project Setting 窗口。下图中的 Setting 窗口可以进行 Compile 和 Synthesize Options 的设置。

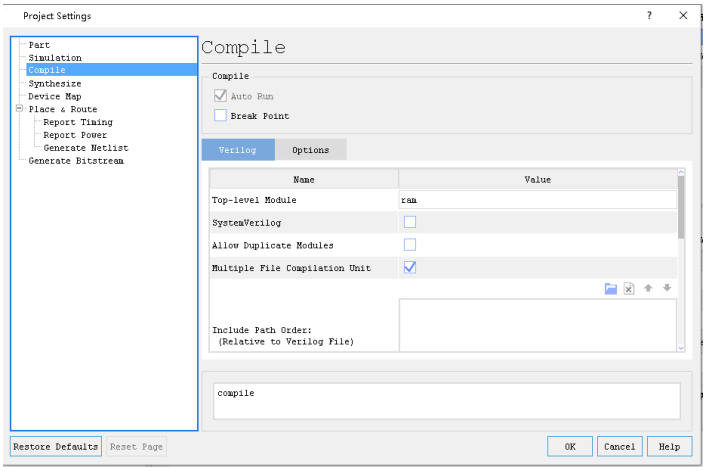

图 2-9 Compile Setting 窗口

选中左侧 Browser 中的 Compile，右侧设置 Compile 的 option。Compile 支持如下设置。

System Verilog: 默认不勾选。选中时默认 Verilog 被设置成 System Verilog。当设计中存在 System Verilog 的语法时，必须勾选该选项才能编译成功。与 shell 命令中的-system_verilog 功能等价。

Allow Duplicate Modules:默认不勾选。工程设计中一般不允许存在同名 module，不勾选本选项时，设计中存在同名 module 时会报错。当勾选本选项后，存在同名 module 可以编译成功，软件会按照 module 解析顺序将后一个 module 覆盖前一个 module。

说明：对于多次例化 IP 的场景，在不勾选的情况下软件也不会检查重名 module，会直接按照 module 解析顺序将后一个 module 覆盖前一个 module。

Multiple File Compilation Unit：控制当前 rtl 文件中定义的变量，宏等，能否在其他 rtl 文件中生效，默认勾选。不勾选时，每个文件单独编译到一个 compilation unit，可以使宏仅在当前文件生效。与 shell 命令的 -multi_file_compilation_unit 功能等价。

Top-level Module：设置 top module，value 值为 module 的名字。如果该项没有设置，PDS 会自动选择没有被实例化的 module 作为 top module，如果输入文件中有多个没有被实例化的 module，ADS 会根据字母排序的次序选择第一个没有被实例化的 module 作为 top module。

Include Path Order：设置工程的 include 路径，与 shell 命令的-include_path 功能等价。点击按钮  弹出 Select Directory 提示框，选择 include 路径完成 include path 的设置。本选项只会将文件目录 include 进来，不会 include 该目录下的文件，实际 include 文件的操作需要结合 rtl 文件中的`include 语句进行。

Include file 可以有多种形式：

- include 绝对路径，如直接在 rtl 文件中添加 `include "/user/dirA/fileA.v"语句。
- inlcude 相对路径，如`include "../fileA.v"。此时的相对路径是基于该 include 语句所在的 rtl 文件所在的目录，需要保证编写该语句的 rtl 文件和 fileA.v 的相对路径正确。
- 搭配 compile 配置选项的路径，如 rtl 文件中添加`include "fileA.v" 语句，并在配置选项 Include Path Order 中添加 fileA.v 所在的目录。

注意：include file 仅为宏定义文件，不是 Verilog 文件，故 include file 不需要添加到工程中。

Parameters/Generics ：默认为空。在选项中设置的参数值会覆盖设计中的顶层参数值。Generic Name 表示顶层参数名，Override Value 表示新的参数值，点击"+"新增，参数名和参数值间使用空格隔开，字符串类型的参数值需要用引号。如{ADDR_WIDTH 10 ASYNC "TRUE"};。与 shell 命令的-defined_parameters 功能等价。Extract Top Parameters 可以将顶层模块的 parameter 提取到 GUI 界面。配合 Generics 功能，可以修改新的参数值，若顶层模块没有 parameter，则会提示没有参数可提取。

Compiler Directives ：默认为空。用于设置宏定义参数，宏定义名和宏定义值间使用'='，多个宏定义指令间使用空格隔开，如"ADDR_WIDTH=8 DATA_WIDTH=4 ASYNC"。与 shell 命令的-compiler_directives 功能等价。

Loop Limit：设置 for 语句循环次数的上限值，默认值为 2000。与 shell 命令的-loop_limit 功能等价。

FSM Compiler：FSM 编码格式选择开关。默认为 auto，其他可选值见下表，与 shell 命令的-fsm_compiler 功能等价。

编码方式	说明
auto	根据软件优化策略选择最优编码方式，默认为 one_hot
off	不执行 FSM infer
one_hot	每个状态只有一位为 1，可减少组合逻辑，速度最快，可优化时序，但使用较多寄存器
gray	相邻状态转换时只有一个状态发生翻转，与 one_hot 相比，使用寄存器较少
sequential	自然数编码，相同条件下使用寄存器最少
original	保持源文件中的原始编码
safe	防止状态机被锁在非法状态，默认为 auto,safe 编码方式。一旦 FSM 进入任意非法状态，可以通过 safe logic 第一时间将 FSM 跳转回合法状态，可增强设计的稳定性
safe，one_hot	使用独热码编码方式，并防止状态机被锁在非法状态
safe，gray	使用格雷码编码方式，并防止状态机被锁在非法状态
safe，sequential	使用自然数码编码方式，并防止状态机被锁在非法状态
safe，original	保持源文件中的原始编码，并防止状态机被锁在非法状态

Synthesize_settings 在图 2-9 中选中左侧 Browser 中的 Synthesize，右侧设置 Synthesize 的 option。Synthesize 支持 Fanout Guide，Disable I/O Insertion，Enable Advanced LUT Combining，Automatic Read/Write Check Insertion for RAM，Resource Sharing，Retiming 和 Minimum Control-set Size 七个 option 设置。

Fanout Guide：设置全局 fanout 的最大值，默认值为 10000。但对寄存器的 CLK 信号，异步复位/置位信号，同步复位/置位信号，使能信号均是无效的。与 shell 命令中的-fanout_guide 功能等价。

Disable I/O Insertion：I/O insert 开关；默认该 switch 是关掉的，网表中插入 I/O buffer，选中该选项卡后，网表中不会插入 I/O buffer。与 shell 命令的-disable_io_insertion 功能等价。

Enable Advanced LUT Combining：将两个相容的 LUT 合并为双输出 GTP_LUT6D 的开关。仅对有 GTP_LUT6D 功能的系列有效，即当前仅支持

Logos2/Titan2/Titan3 系列。设置为非选中状态则不进行 LUT 的 pack。默认为打开状态，Logos2/Titan2/Titan3 系列外默认为置灰状态。与 shell 命令的–enable_prepacking 功能等价。

Automatic Read/Write Check Insertion for RAM：与 shell 命令中的-rw_check_on_ram 功能等价。

说明：当前只有简单双端口 ram 触发了 transparent_write 模式后并勾选该选项后，会插入 bypass 逻辑，保证在同一个时钟周期内，读出去的数据就是写入的数据，避免出现仿真不一致。

Resource Sharing：Resource Sharing 开关。默认是打开状态，对算数运算实现资源共享，主要是对加法（减法）运算和乘法运算进行资源共享。资源合并后可优化面积，但可能会降低时序。设置为非选中状态不进行资源共享。与 shell 命令的-resource_sharing 功能等价。

Pipelining：Pipelining 开关。默认是打开状态，通过挪动寄存器往前，将 dpram 异步输出端口的 mux－chain 就吸收掉，从而减少资源，优化面积。设置为非选中状态则不进行 Pipelining 操作。与 shell 命令的-pipelining 功能等价。

Retiming：Retiming 开关；默认是关闭状态，打开开关后，Retiming 通过移动电路中已有的 register 的方式，来实现对电路的 timing, power，或者 area 的优化。在此过程中，不会插入多余的逻辑，但是在移动 register 中可能会增减 register 的数量。与 shell 命令的 -retiming 功能等价。

Minimum Control-set Size：调整具有同一控制端口的 FF 的最小值，即 CLK、CE、同步 SET/RESET 等端口一致的 FF 的最小值。默认值为 2，可选值为正整数。当具有同一控制端口的 FF 大于等于设置值时，不进行优化；当小于最小值时，会将对应的 FF 的部分或全部控制引脚移至 D 输入端。与 shell 命令的–min_controlset_size 功能等价。

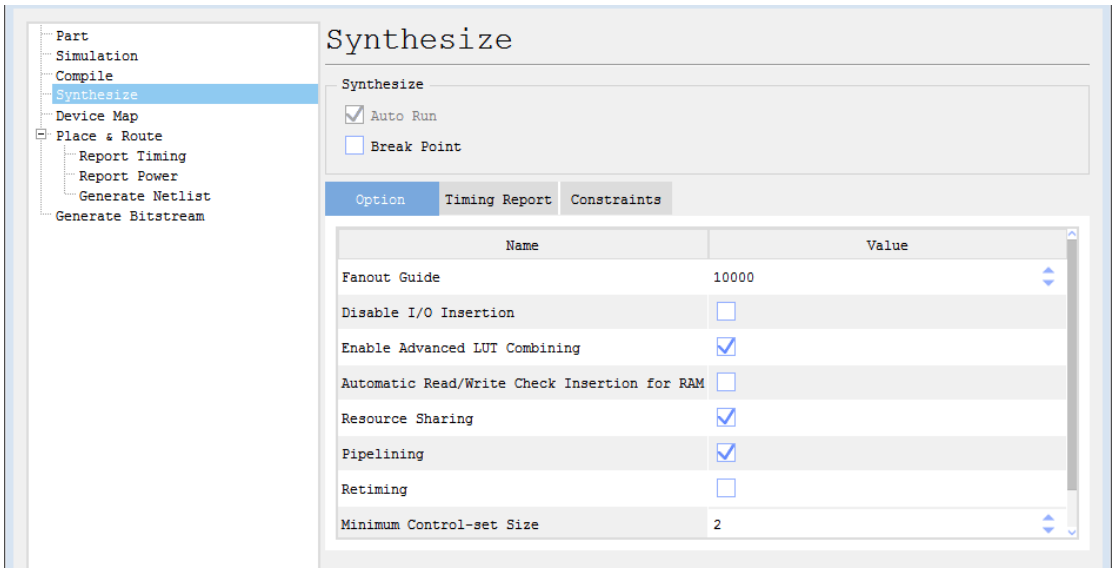

图 2-10 Synthesize Option 配置

在 Synthesize 页选中间的 Timing Report，可以设置 synthesize timing report 相关选项。Synthesize timing report 支持的如下：

Total Number of Paths (-max_path)：即 max_path，表示综合时序报告中报告的最差时序路径的数量。与 shell 命令的-max_path 功能等价。

Number of Paths per Endpoint (-nworst)：即 nworst，表示综合时序分析时，报出的每个 end point 最多能保存的 path 数量。与 shell 命令的-nworst 功能等价。

Report Paths with Logic Levels >：表示综合时序时，报出的每个 setup/recovery 时序路径的 Logic Levels 必须大于设定值，小于该值的时序路径将不予显示。该设置不影响 hold/removal 分析时序路径。当设置的 logic level > -1 时，时序分析会优先报出 logic level 大的时序路径，而不一定是 slack 差的路径，时序报告也会按照 logic level 的大小排序。（仅在 TA 中可用）

Limit Paths by Clock Group：设置时钟组的名字，时序报告中只会显示已设置时钟组的时序路径。（仅在 TA 中可用）

Number of Carry Instance as One Logic Level：设置 carry instance 的数目作为一级 logic level，默认是 4 个 carry instance 作为一级 logic level。（仅在 post-synthesize TA 中可用）

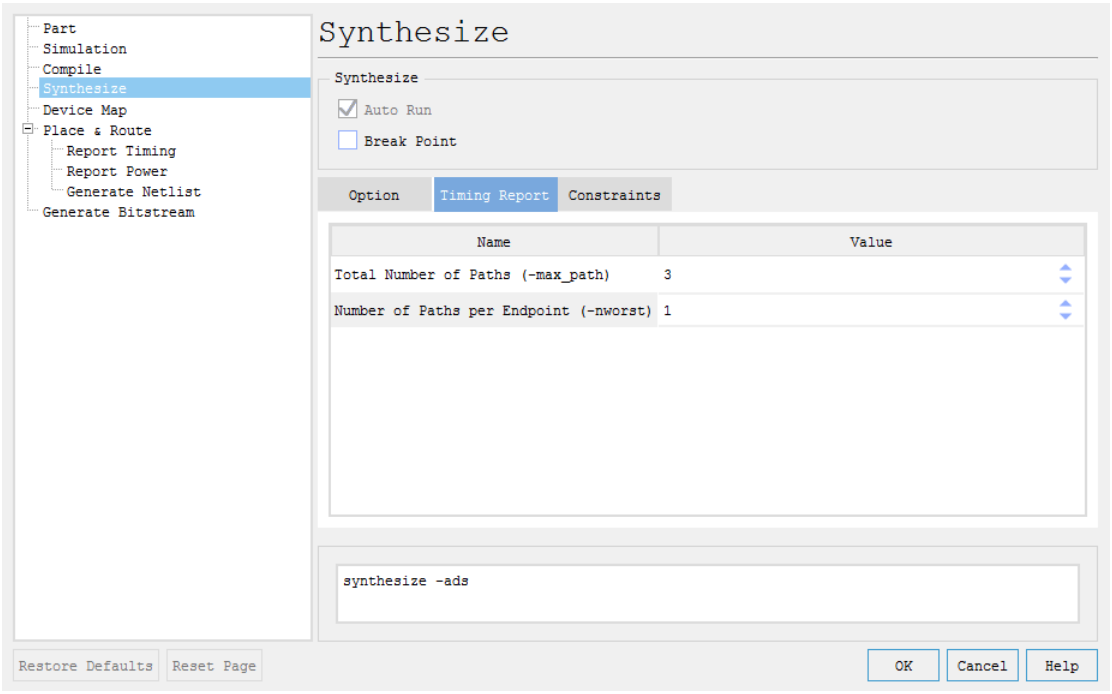

图 2-11 Synthesize Timing Report Option 配置

在 Synthesize 页选左侧的 Constraints，可以设置综合时序约束相关选项。Constraints 支持的 option 有两个：

Frequency（MHz）：用于 infer clock 中，如果无法根据 PLL 的参数以及参考时钟计算正确的时钟频率，就会用这个 option 的设置值为时钟频率来创建时钟，默认 1.0000 勾选。目前不支持 auto constraint，Auto Constrain 置灰，且 Auto Constrain 不可勾选。与 shell 命令的-frequency 功能等价。

Use Clock Period for Unconstrained IO：这个选项是为用户没有约束 IO delay 的 port 增加 IO delay 约束，提升时序约束的覆盖程度，默认不勾选。与 shell 命令的-use_clocked_period_for_unconstraint_io 功能等价。

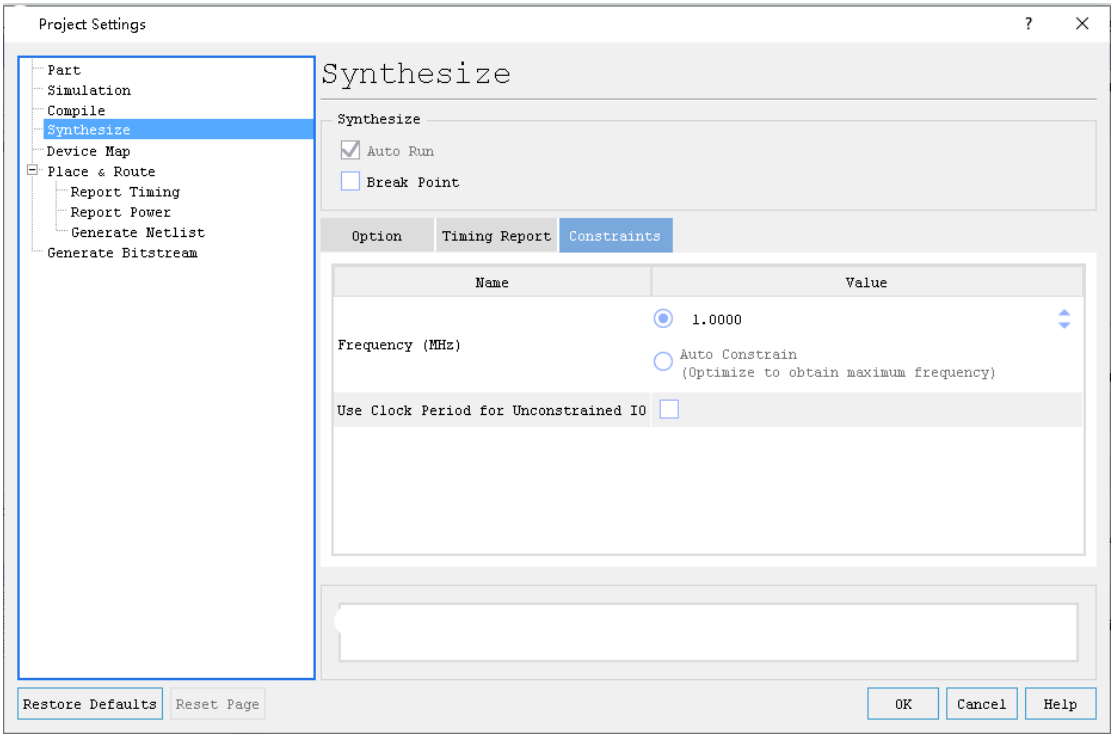

图 2-12 Synthesize Constraints Option 配置

3 ADS 综合网表分析

3. 1 ADS 综合网表的查看

ADS compile 阶段和 synthesize 阶段生成的 RTL 网表和 Technology 网表可以通过 View RTL Schematic 和 View Technology Schematic 进行查看和分析。

点击 toolbars 中的  (View RTL Schematic)按钮，可查看 RTL 网表，view 界面如图 3-1 所示。其中上方的工具栏按钮为 View RTL Schematic 支持的功能。

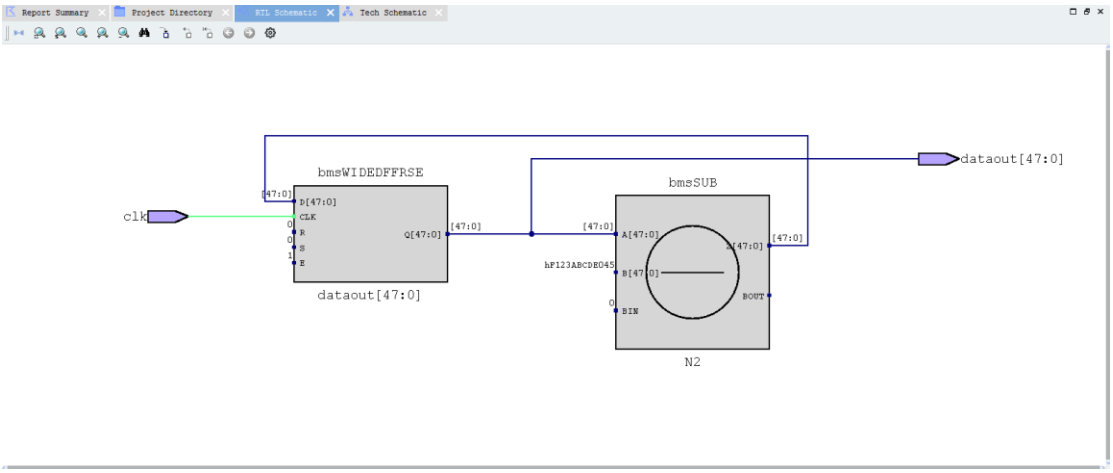

图 3-1 RTL 网表

Design Browser：点击工具栏按钮  会打开 Design Browser 窗口，显示网表的 Nets，Ports，Leaf cells 和实例化的 instance，选中 Design Browser 中的对象对应的结构在网表中会高亮显示。

Fit：适合图像；点击工具栏按钮  会根据当前视图中所有物体的外框进行适当缩放，使得所有物体都可以显示出来并充满整个视图。

Fit Selection：适合选中的物体显示；点击工具栏按钮  会根据当前视图中所有被选中物体的外框进行适当缩放，使得所有选中物体都可以显示出来并充满整个视图。

Zoom In：放大；点击工具栏按钮  会激活交互式操作，此时使用者可以按住鼠标左键并在视图中移动以画出一个矩形框，再松开鼠标按键则会根据该矩形框进行适当缩放，使得位于其中的物体都可以显示出来并充满整个视图。Zoom In 是一个连续的交互操作，需要点击 Esc 键来退出。

Zoom In By 2：放大两倍；点击工具栏按钮会将当前视图中的物体放大两倍显示，可显示的范围则会缩小。

Zoom Out By 2：缩小两倍；点击工具栏按钮会将当前视图中的物体缩小两倍显示，可显示的范围则会扩大。

Find：查找；点击工具栏按钮，进行对象查找。

Descend：下层显示；点击工具栏按钮激活 Descend 操作，此时移动鼠标到一个 instance 物体上时，如果光标图形变为手形则表示可以进入到下层的设计单元中，单击鼠标左键即可进入；或者先选中一个 instance 物体，再右键菜单，点击下拉菜单中的 Descend 项也可以进入到下层的设计单元中。

Return：上层显示；如果位于一个下层的设计单元中，点击工具栏按钮，向上返回一层。

Return To Top：返回顶层显示；如果位于一个下层的设计单元中点击工具栏按钮，返回到顶层单元显示。

Previous view：上一显示状态；点击工具栏按钮，返回上一显示状态；如果该按钮为灰色，则表示已经为最初始的显示状态。

Next view:下一显示状态；点击工具栏按钮，显示下一状态；如果该按钮为灰色，则表示该网表为最后显示状态。

点击 toolbars 中的(View Technology Schematic)按钮，可查看 Technology 网表，view 界面如下图所示。

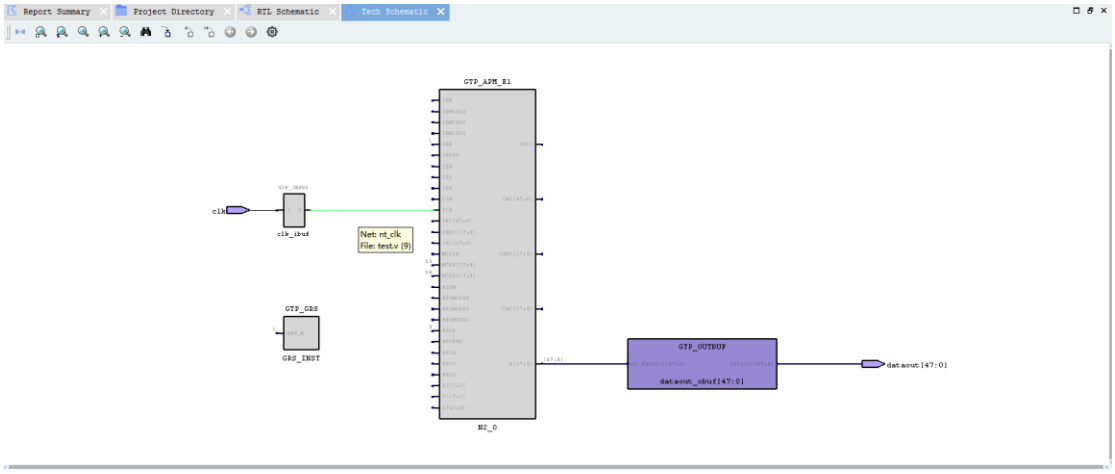

图 3-2 Technology 网表

View Technology Schematic 的上方工具栏的功能和操作与 View RTL Schematic 相同。

按住鼠标滚轮可拖动视图，按住 Alt+鼠标滚轮可上下滑动视图，按住 Ctrl+鼠标滚轮可放大缩小视图。

3. 2 Design Browser 显示

左侧的 design browser 可以显示当前视图的资源信息，如图：

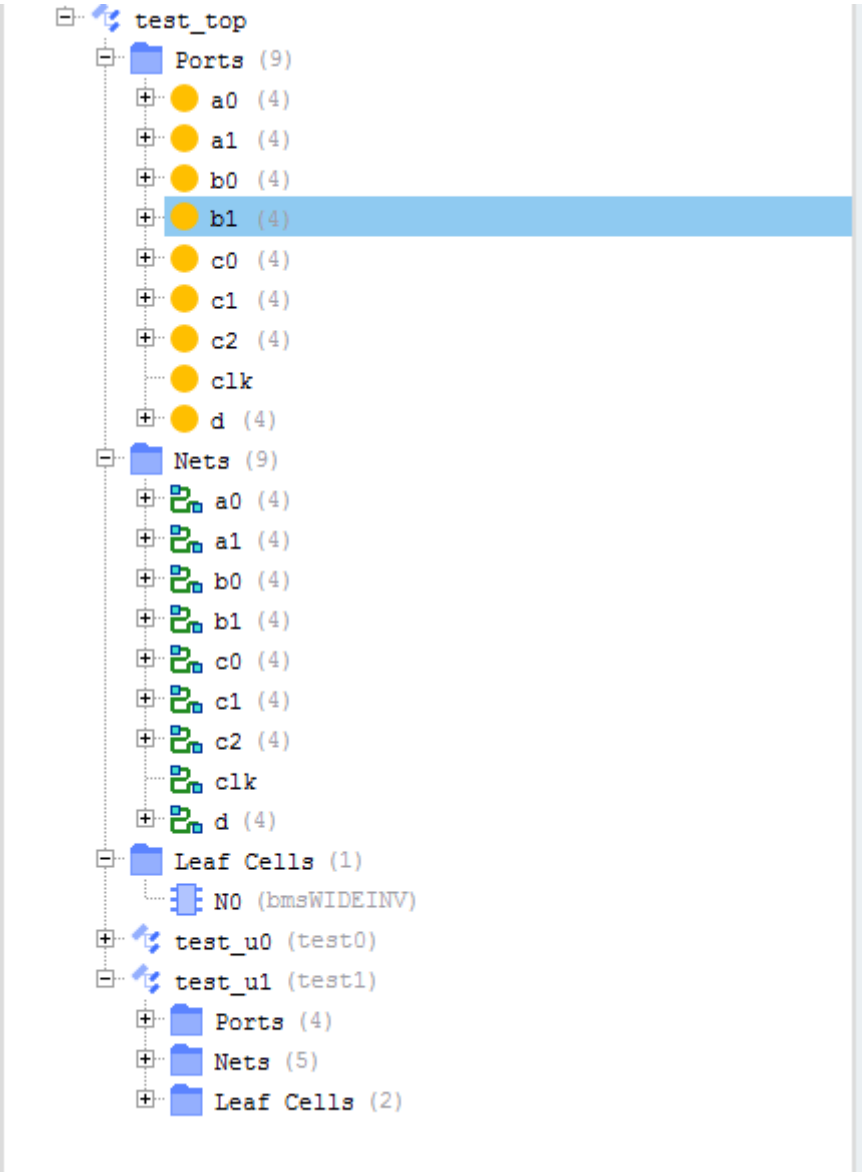

图 3-3 Design Browser

Design browser 中有几个右键菜单，分别如下表：

序号	一级菜单	二级菜单	说明
1	Zoom		将视图定位到选定对象
2	Descend		打开选定对象的原理图
3	Filter		选定对象打开原理图
4	Filter In New Window		选定对象在新界面打开原

			理图
5	Copy Name	Copy Name	拷贝名称
		Copy Hierarchical Name	拷贝层级名称（不含顶层）
6	View Obj In Source		跳转源文件查看对象
7	View Module In Source		跳转源文件查看对象模型定义
8	Properties		打开选定对象的属性窗口
9	Select Child Items	All	选中 module 下的所有对象
		All Ports	选中 module 下的所有 Port
		All Instances	选中 module 下的所有 Instance
		All Leaf Cells	选中 module 下的所有 Leaf Cell
10	Select Leaf Cells		新增仅选中当前选中列表下的 Leaf Cells，去除 Leaf Cell 类型外的选中
11	Collapse All		折叠所有

3. 3 Schematic 分页显示

对于较大的原理图，会分成若干页来进行显示。敲击键盘上的 Page Up 和 Page Down 按键可以进行翻页。在最后一页点击 Page Down 会回到第一页，在第一页点击 Page Up 则会翻到最后一页。

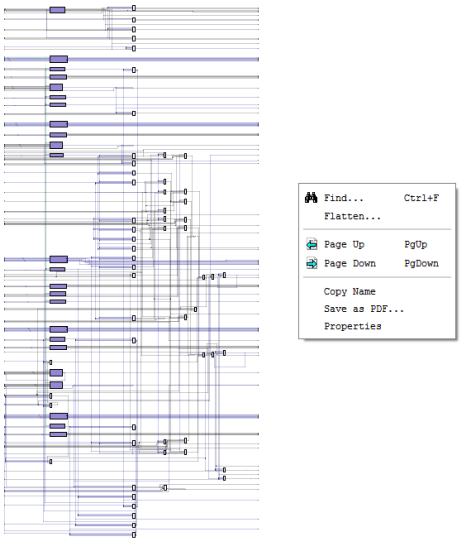

图 3-4 分页原理图显示

3. 4 Objects 查找功能

在原理图中经常存在大量的物体，通过 Find 功能可以对指定的物体进行查找。点击原理图工具栏中的按钮或通过快捷键 Ctrl+F，可以打开 Find 对话框。

在 Find what 输入框中输入要查找的对象名称，支持 UNIX 风格的通配符*和?等。例如输入 I*，就会匹配 I 开头的物体；如果不加通配符，则进行完整的匹配。

查找对象类型的类型有以下几种：Instances 表示直接查找单元实例、Cells 表示通过单元查找其实例、Groups 表示查找单元实例构成的组、Nets 表示查找线网，Port 表示查找端口。可以勾选它们的一个或几个，还可以通过按钮 A 和 N 快速重置它们的状态：

A：All checked 选择全部物体种类

N：None checked 去选全部物体种类

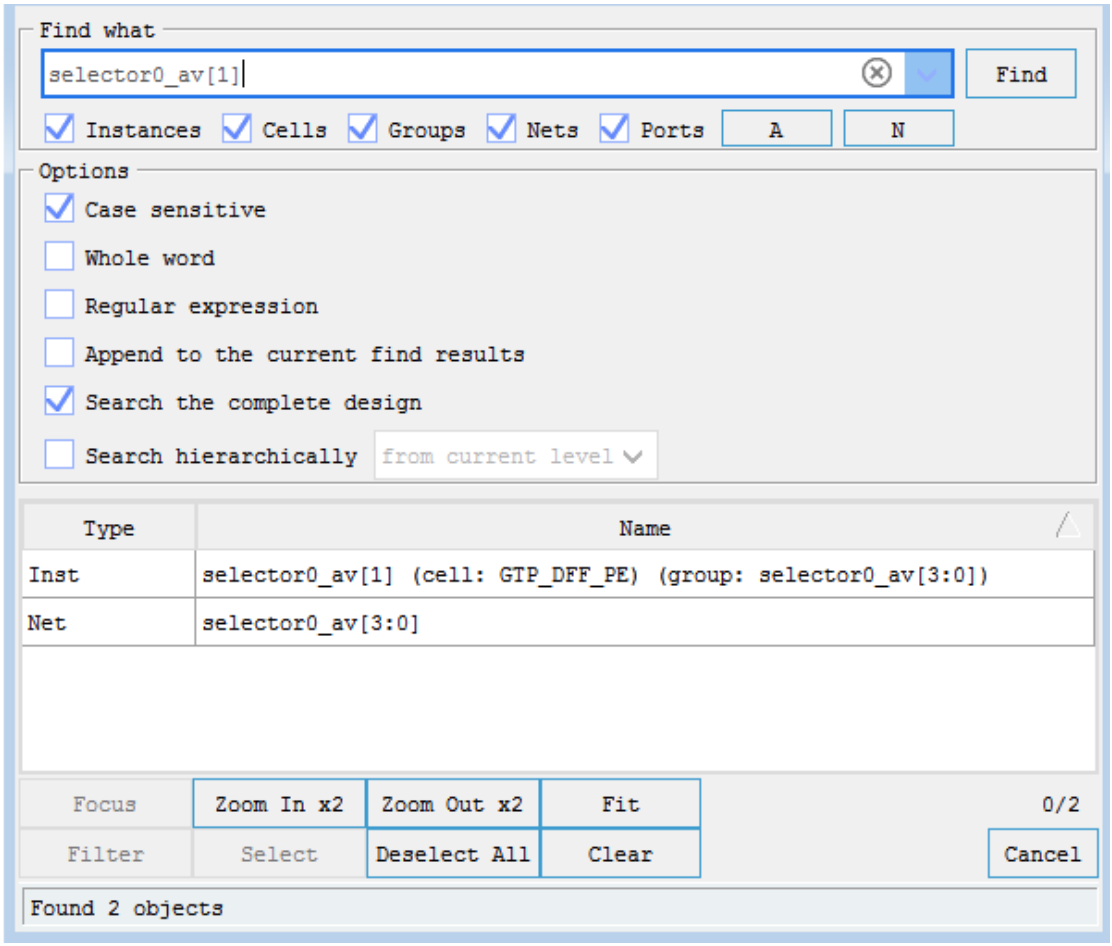

图 3-5 原理图 Find 对话框

其它的选项还有：

Case sensitive：对大小写敏感，在查找过程中区分大小写。

Whole word：全词匹配。

Regular expression：模糊匹配搜索，可以搭配"* ?" 进行模糊搜索，并且可以将"[]"特殊字符进行自动转义。

Append to the current find results：在查找开始后，并不清空以前的结果，新找到的物体会合并到结果列表中。

Search the complete design：如果当前原理图是经过过滤后的，那么勾选此项则会在完整的网表中查找；否则，只在当前可见的原理图中查找。

Search hierarchically：进行层次查找，可以选择 from top level 从顶层逻辑单元开始，或 from current level 从当前逻辑单元开始。在层次查找时是不能对单元实例构成的组 Groups 进行搜索的。在 Find what 输入框中回车或点击 Find 按钮，

则会开始查找过程，查找的结果会显示在列表中。查找结束后，可以对结果列表中的物体进行跳转、缩放、定位等操作。最简单的，双击列表中的一项，可以直接跳转到该物体所在位置，并闪烁高亮之。还有一些按钮可以方便地进行一些操作。

Focus：将视图定位到选定对象并闪烁（同双击效果）。

Zoom In x2：放大两倍显示。不需选中物体，直接点击即可将当前窗口中的视图放大两倍显示。

Zoom Out x2：缩小两倍显示。不需选中物体，直接点击即可将当前窗口中的视图缩小两倍显示。

Fit：将当前窗口中的视图缩放至充满整个窗口。

Filter：对结果列表中选中的物体做过滤操作，将它们显示在另一个原理图中。注意这些物体需要位于同一个设计单元中。

Select：如果选中的若干物体位于同一个单元中，则可以跳转到该单元中并选中高亮这些物体。

Deselect All：取消当前窗口中选中的所有物体。

Clear：清除当前的查找结果。

点击 Cancel 按钮或对话框的关闭按钮，可以关闭对话框。如果此时查找过程还未结束，则会询问用户是否需要中断查找。

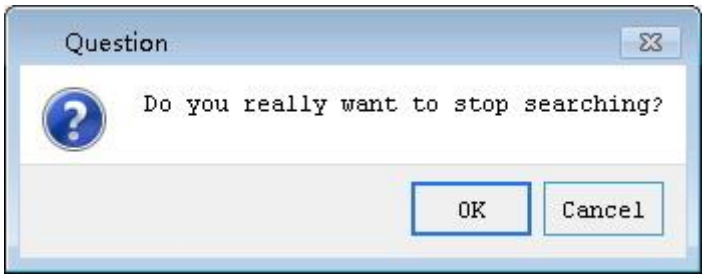

图 3-6 中断查找

3. 5 Schematic Preferences 设置

用户可以通过 Preferences 来设置原理图的显示模式等。点击工具栏中的 (Preferences)按钮弹出可以窗口进行以下设置。

Display 部分主要控制一些与原理图显示内容等相关的属性，如图。

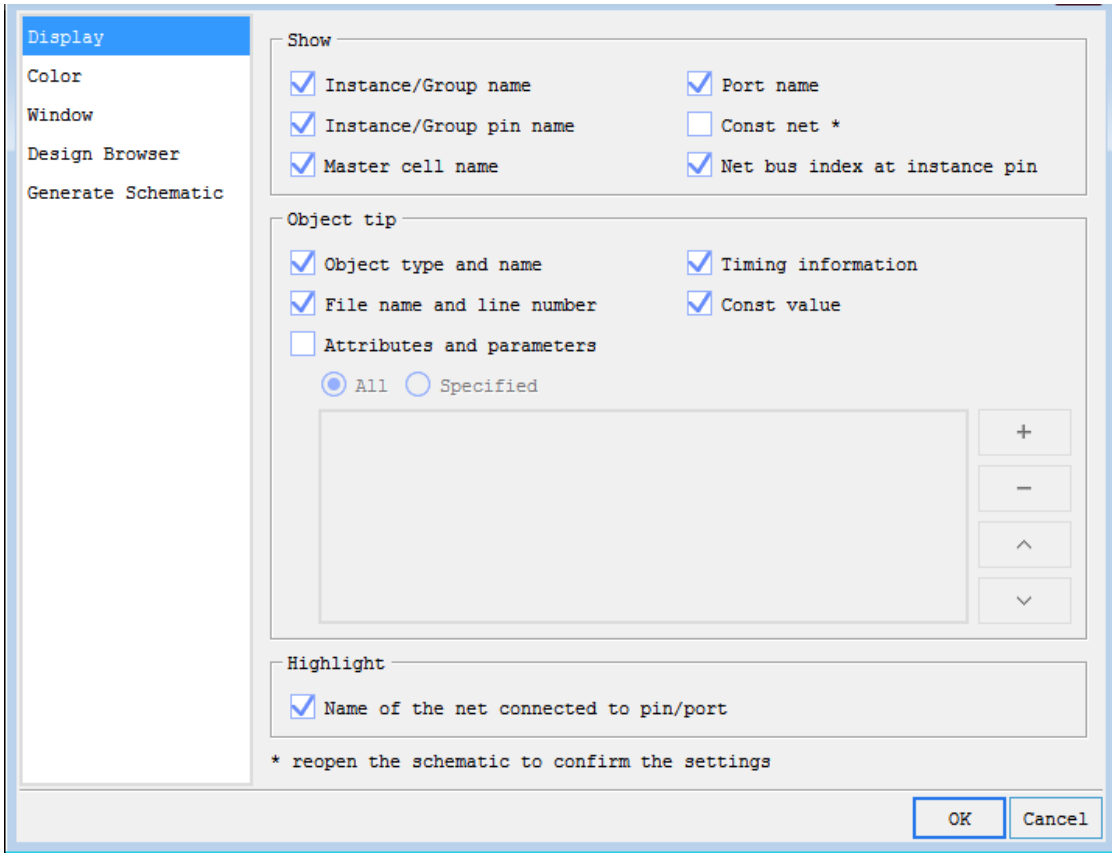

图 3-7 原理图 Preferences 设置对话框 Display 部分

- Instance/Group name：是否显示实体或分组的名称。
- Instance/Group pin name：是否显示实体或分组的管脚名称。
- Master cell name：是否显示实体/分组的单元名称。
- Port name：是否显示端口的名称。
- Const net：是否显示常量线网，如 VCC/GND 等。
- Net bus index at instance pin：是否在管脚处显示所连 net-bus 的下标。
- Object type and name：是否显示对象的类型和名称。
- File name and line number：是否显示对应的 rtl 文件名和对象所在的行

- Attributes and parameters：是否显示对象所含属性及参数，可选择显示所有或显示指定的属性/参数。
- Timing information：是否显示对象的时序信息。
- Const value：是否显示常量值。
- Name of the net connected to pin/port：是否高亮显示与端口/管脚连接的线网名称。

Color 部分控制绘制原理图时的颜色，如图。用鼠标双击列表中的一项，可以选择新的配色。点击 Reset 按钮，可以恢复上次保存的配色，或者恢复为缺省的白色或黑色背景配色。

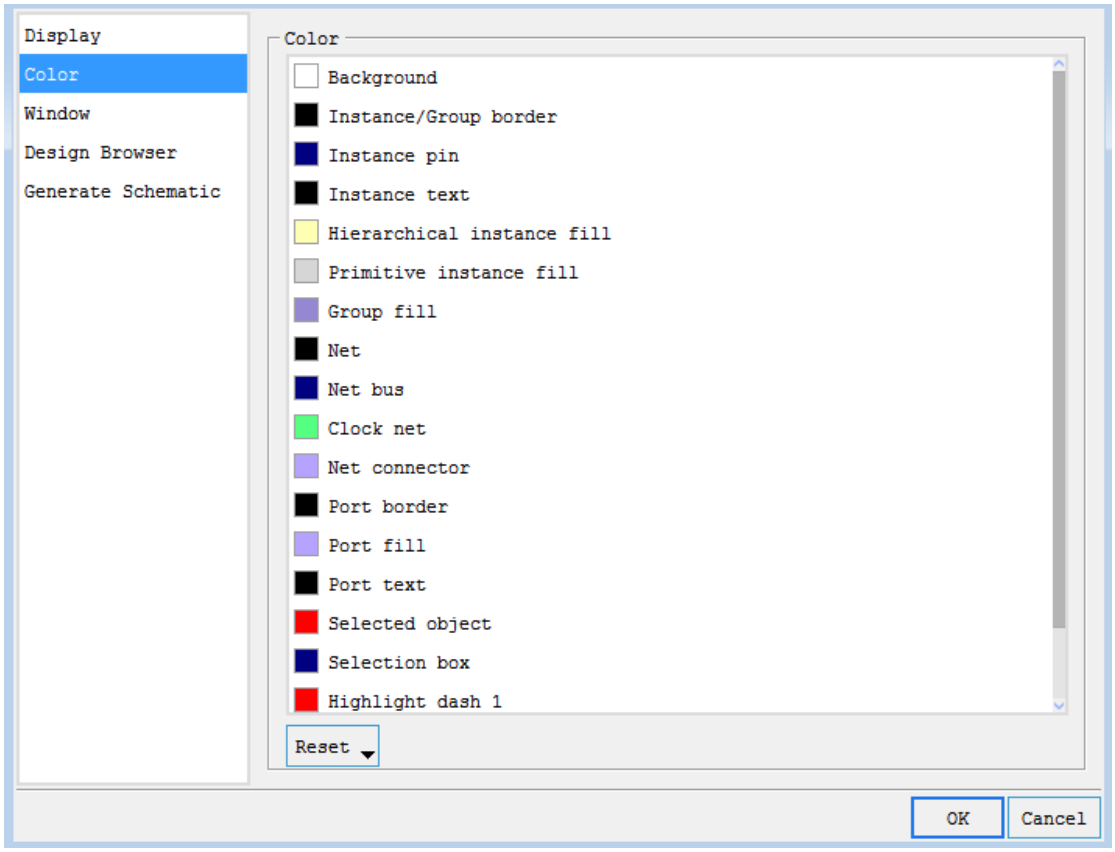

图 3-8 原理图 Preferences 设置对话框 Color 部分

Window 部分控制原理图窗口的一些控件，如图。

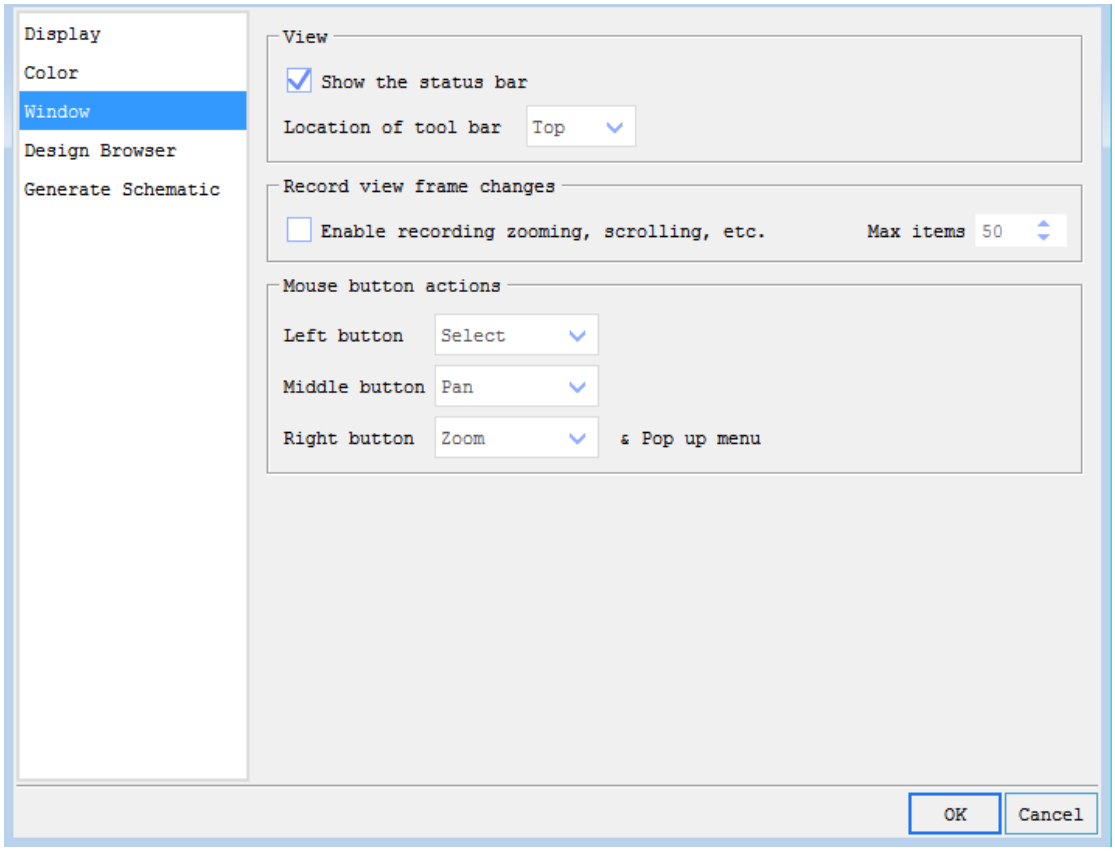

图 3-9 原理图 Preferences 设置对话框 Window 部分

- Show the status bar：是否显示窗口底部的状态栏。
- Enable recording zooming, scrolling, etc：当绘图区域有缩放或滚动等操作时，是否记录这些操作，使得可以通过 Previous View / Next View 来重复刚才的操作。如果该选项打开，可以设置记录的步数。
- Mouse Button actions：选择鼠标左键、中键（滚轮）、右键的功能。

Design Browser 部分控制 Design Browser 窗口的显示与操作，如图。

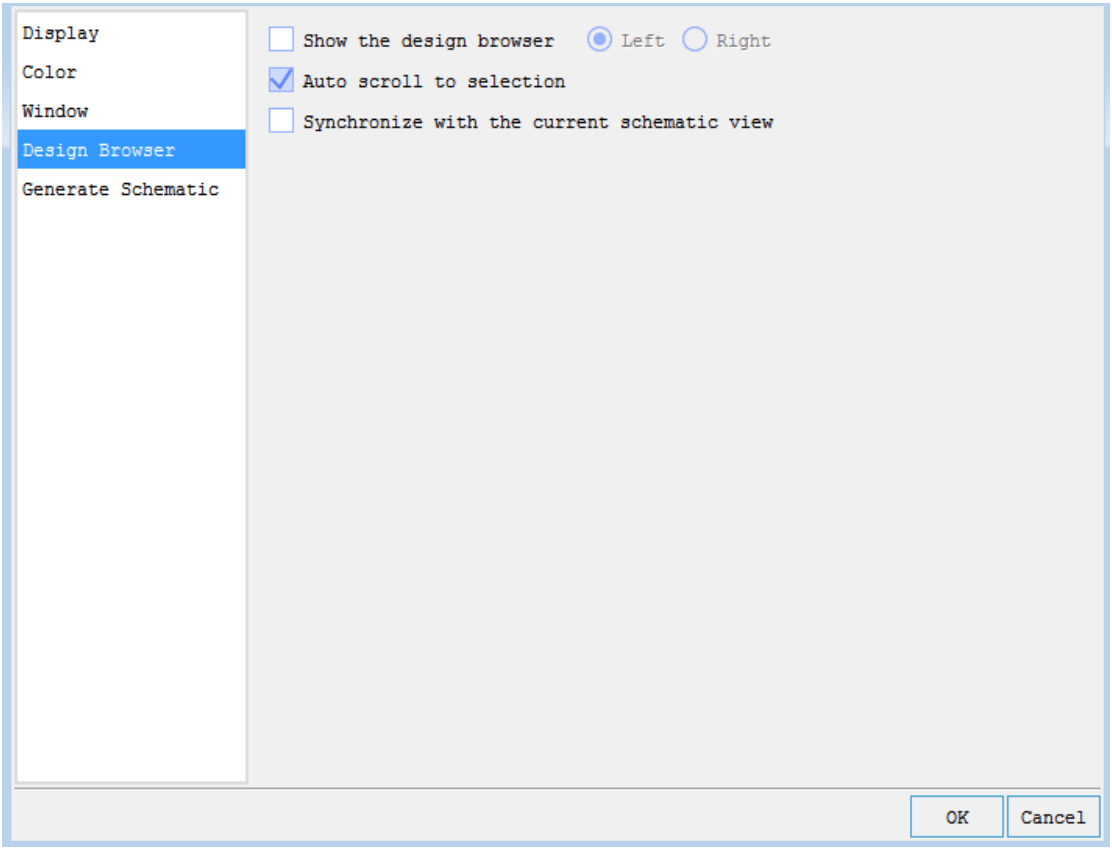

图 3-10 原理图 Preferences 设置对话框 Design Browser 部分

- Show the design browser：用于设置是否显示 Design Browser 及显示的位置。
- Auto scroll to selection：用于指定视图切换选中时，DesignBrowser 是否跟随选中并高亮。
- Synchronize with the current schematic view：用于指定当前视图模块层次切换时，是否将 Design Browser 节点同步

Generate Schematic 部分控制原理图生成过程中的一些属性，如图。

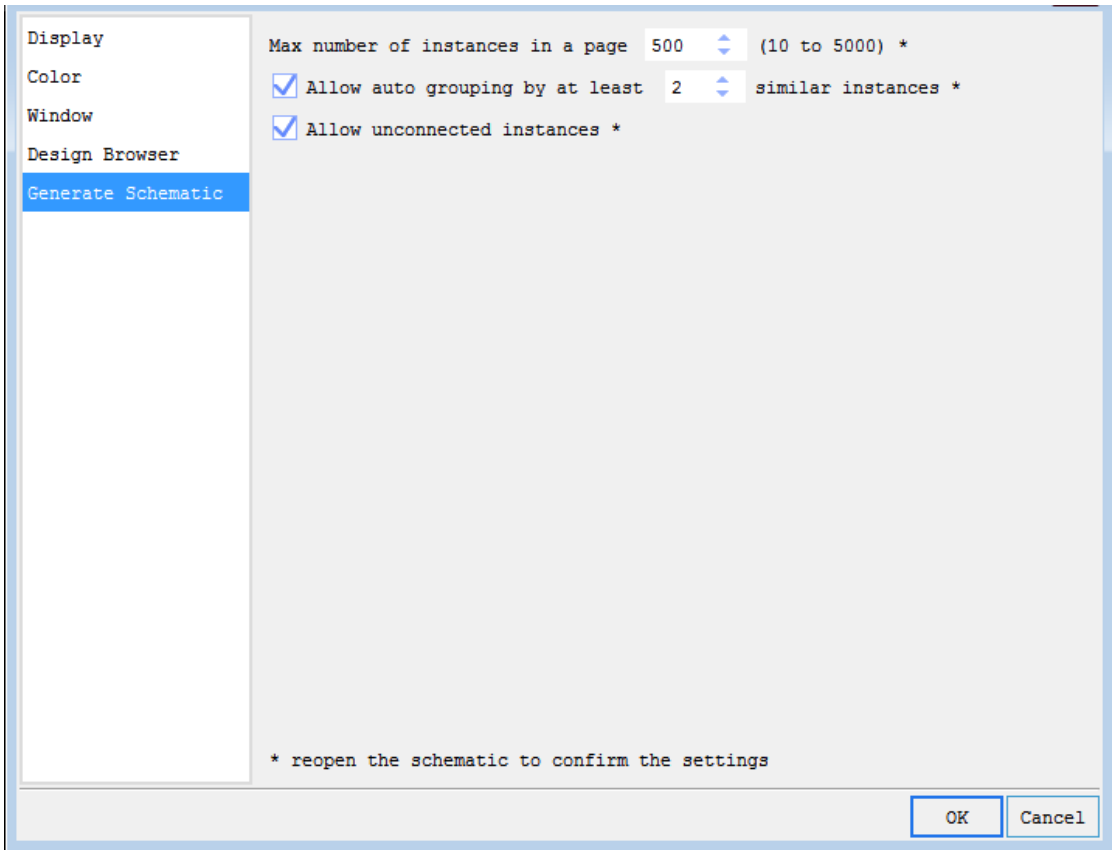

图 3-11 原理图 Preferences 设置对话框 Generate Schematic 部分

Max number of instances in a page：设置每个原理图页中最多可以放置的实体个数。

Allow auto grouping by at least ( ) similar instances：是否将达一定数量的拥有相同的单元且名称类似的实体自动合并成一个分组。

Allow unconnected instance *：是否显示没有连接的实体的原理图。

3. 6 RTL 和 Technology view 右键菜单功能介绍

在原理图中高亮或选中某个物体后，点击鼠标右键可以弹出下拉菜单，以进行一些操作。菜单的内容根据物体类型和状态而有所不同。菜单中定义的操作如下所列：

View Object in Source：原理图到文本的定位，打开对应的源代码文件并定位到相应的行。

View Module in Source：模块原理图到文本的定位，打开对应的源代码文件并定位到相应的行。

View Object in RTL/Tech View：RTL 视图与 Technology 视图的交互定位；如在 RTL 视图中选中某物体，则可定位到 Technology 视图，并高亮显示与之相对应的物体，反之亦可。

Edit Attributes：打开 UCE 窗口设置 attribute；该选项为 RTL Schematic 的功能选项，Technology Schematic 没有该选项。

Show Hierarchy：展示模块层级原理图。

Descend：下层显示；可以向下进入到所选中的 instance 或 group 物体中查看其细节。

Find：打开查找对话框。

Filter：将所选中的 instance 物体单独显示在窗口中。

Filter in New Window：将所选中的对象单独显示在新开的 Schematic 视图中。

Select Leaf Cells：仅选中当前选中列表下的 Leaf Cells，去除 Leaf Cell 类型外的选中。

Flatten：将当前原理图中的层次打散显示。

Expand：将与该 instance 相连接的物体高亮显示。

Expand to Port/Register：将与该 instance 连接的物体扩展至 port/register 并高亮显示。

Page Up：向前翻页；对于分为多页进行显示的设计，直接点击按键 PageUp 或点击右键菜单项Page Up 可以向前换一页，当到达第一页后则会从最后一页重新开始。

Page Down：向后翻页；对于分为多页进行显示的设计，直接点击按键 PageDown 或点击右键菜单项Page Down 可以向后换一页，当到达最后一页后则会从第一页重新开始。

Copy Name：复制当前高亮物体的名字到剪切板中，如果没有高亮物体复制当前打开的单元名。

Show Connectivity：将所选对象连接的 net/port/instance 高亮选中。

Timing：打开属性窗口查看时序信息，该选项为 Technology Schematic 的功能选项，RTL Schematic 没有该选项。

Save As PDF…：支持将 RTL 或 Technology 视图导出保存为 PDF 文件，可以自主选择要导出的视图范围、导出页面大小、页面方向，默认导出路径为工程文件同级目录。

Properties：查看当前高亮物体的属性。

3.6.1 Filtering Schematics

如下图，选中要 Filter 的 object，右击选择下拉菜单中的 Filter，即可在原理图中只显示选中的 instance。ADS 支持单个或多个 object 的 Filter，选中多个 object 的方式为 Shift+鼠标左键。

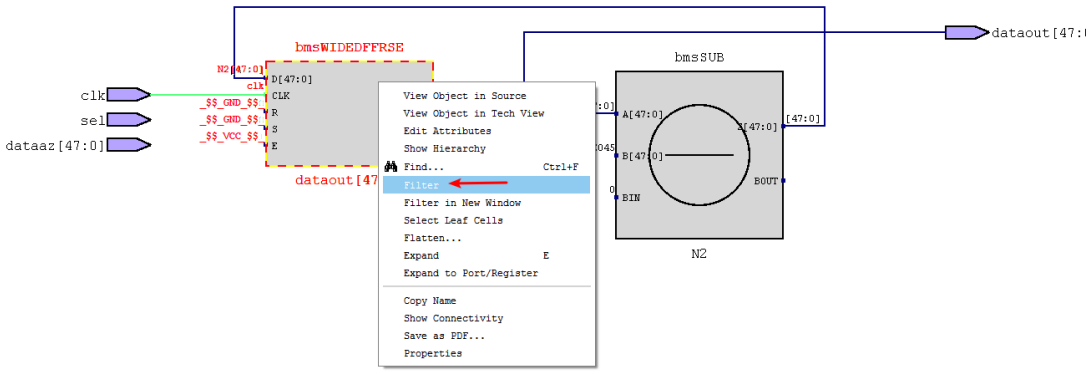

图 3-12 原理图 object Filter

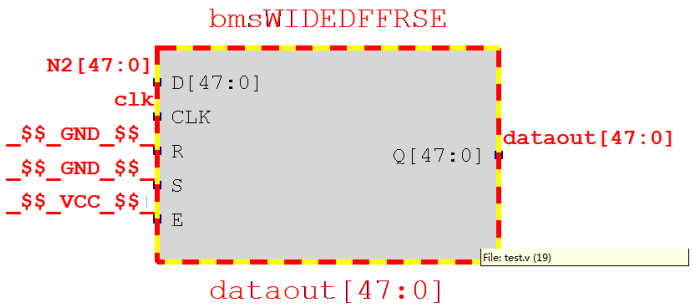

图 3-13 原理图 object Filter 效果

3.6.2 Expand and Expand to Port/Register

Expand 功能实现与所选 object 有连接关系的 object 的扩展，并将对应的 objects 边框高亮显示。如图 3-14，选中 N2 的 instance，右击选择下拉菜单中的 Expand，会将与 N2 连接的 instance 边框高亮显示。

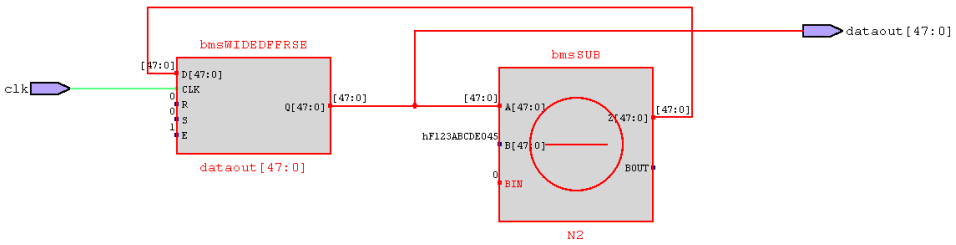

图 3-14 原理图 object Expand

进行网表 Expand 的选项除了上述的 Expand，还有 Expand to Port/Register。Expand to Port/Register 可以将与所选 object 有连接关系的 instance 边框高亮显示并 expand 到 port 和 register。

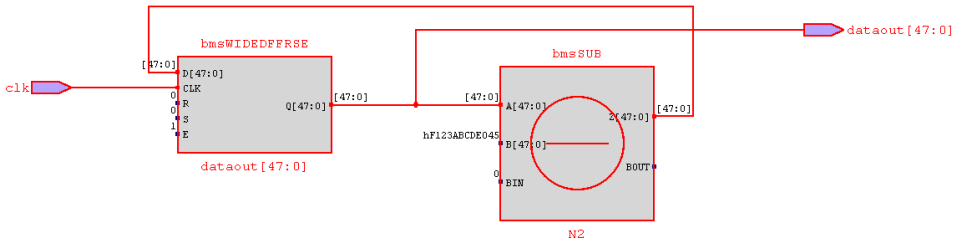

图 3-15 Expand to Port/Register

3.6.3 Flatten

打散显示当前的原理图，原来的带层次的 instance 都会被最底层网表代替。可以选择是否打散那些 instance 组（见下图）；注意，即使选择不打散 instance 组，那么这些组是否还会生成是由 Preferences 中的 Allow auto grouping by at least ( ) similar instances 选项控制的。

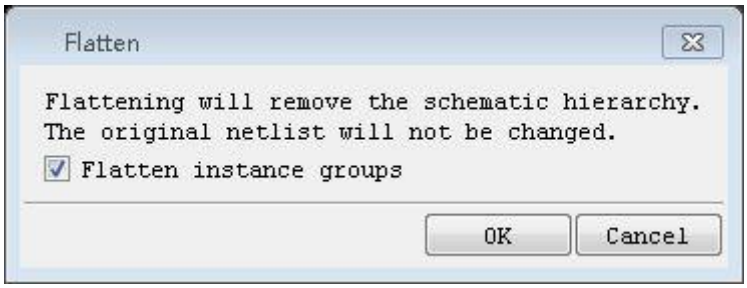

图 3-16 Flatten

3.6.4 查看属性

移动鼠标，选中视图中的某个物体，右键选择下拉菜单中的 Properties，弹出下图对话框：

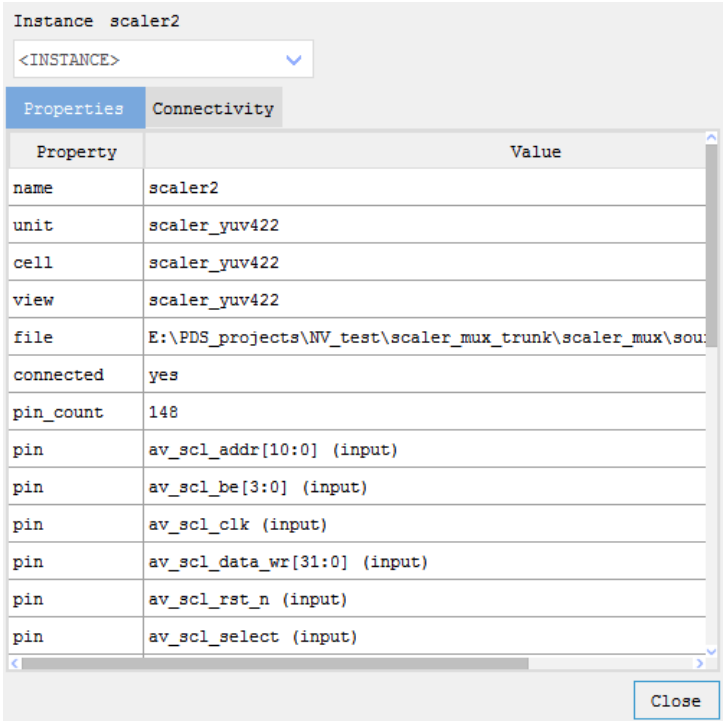

图 3-17 Properties 对话框

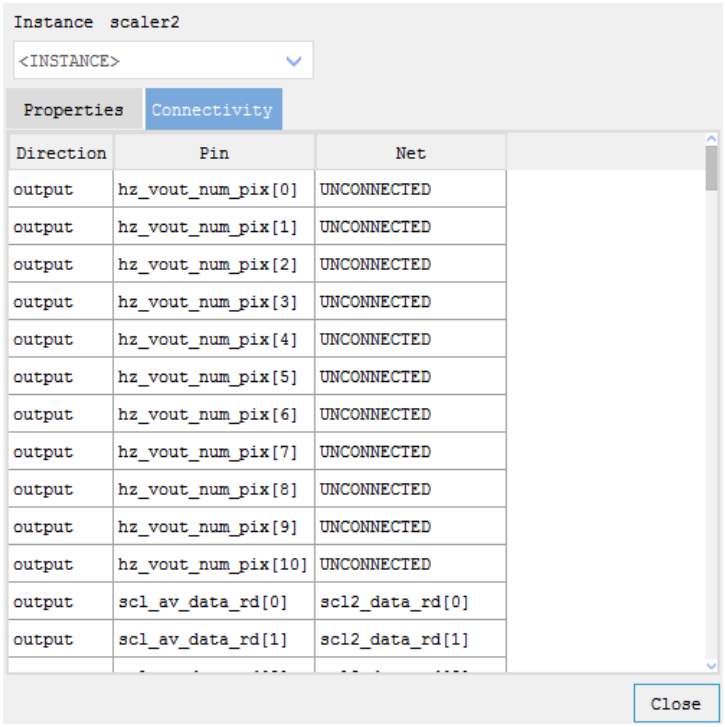

图 3-18 Connectivity 对话框

上述对话框可以查看一个物体的属性或者连接状态。对于 instance 物体，还可以选择查看它的端口；对于 bus 类型的 net 或 port 物体，则可以选择查看其中的一个子项。

可以使用鼠标进行单击或双击操作：

序号	界面名称	说明
1	双击列表中的 Net 对象	视图中定位并闪烁该 Net
2	双击列表中的 Pin 对象	视图中定位并闪烁该 Pin 所属的 Instance
3	双击列表中的 Instance 对象	视图中定位并闪烁该 Instance
4	双击列表中的 Port 对象	视图中定位并闪烁该 Port
5	双击列表中的 Group 对象	视图中定位并闪烁该 Group
6	单选、多选	支持（通过 Ctrl 或 Shift 多选），仅表格选中

属性窗口中有几个右键菜单，分别如下表：

序号	功能	说明
1	Focus	将视图定位到选定对象并闪烁（同双击效果）
2	Clear Focus	取消界面闪烁
3	Select	选中对象，表格、视图、Design Browser 均选中
4	Deselect All	取消界面选中
5	Filter	过滤选中对象
6	Copy	拷贝名称
7	All	选中当前属性窗口的全部单元格

属性窗口支持列排序，如下图：

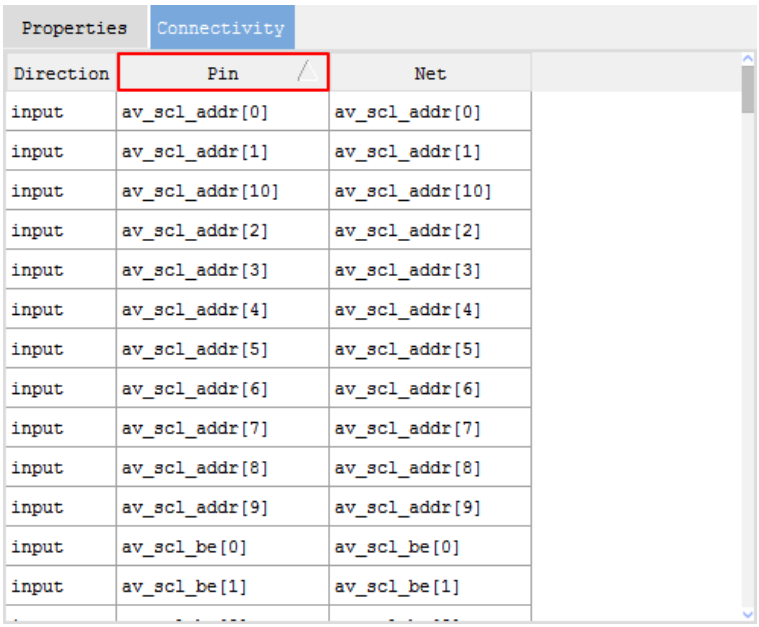

图 3-19 支持列排序

打开属性窗口的同时可以进行视图操作（注：一个 NV，同时仅能打开一个属性窗口）

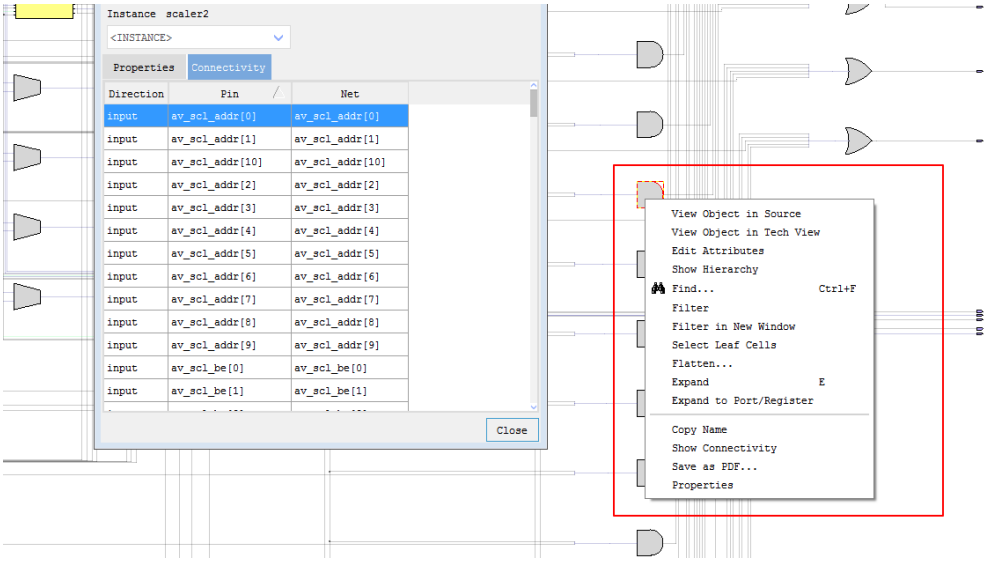

图 3-20 打开属性窗口同时操作视图

3.7 Schematic view 和 Verilog 源代码的交互定位

在 Schematic view 中选中某个 object 后，右击选择菜单中的 View Object in Source 可以在文本编辑器中打开对应的源代码，并定位到该 object 在源代码中对应的描述行。对应的描述在原文件中高亮显示。

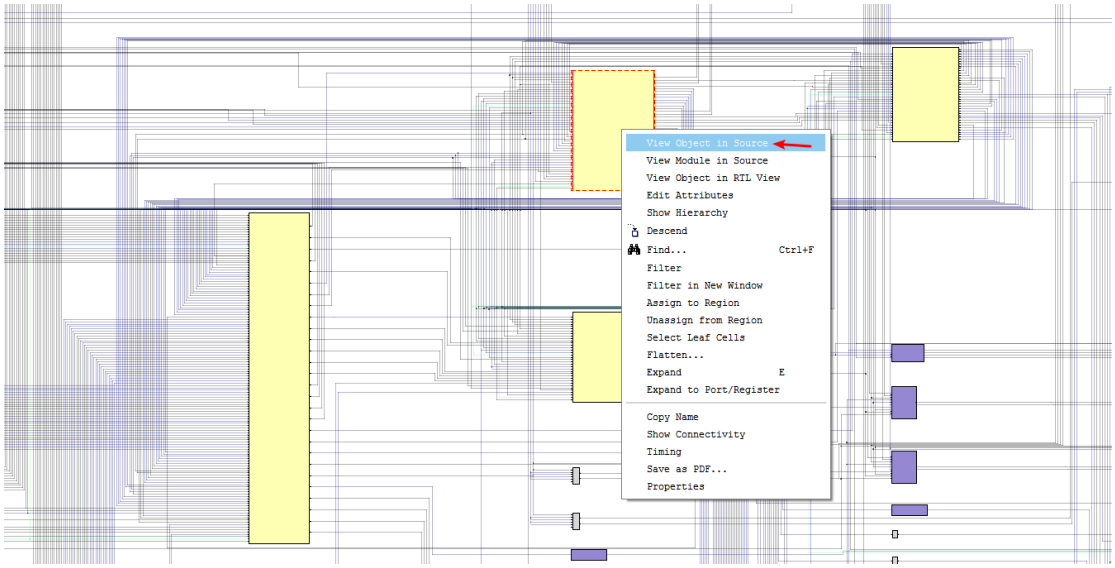

图 3-21 RTL 网表与文本交互定位

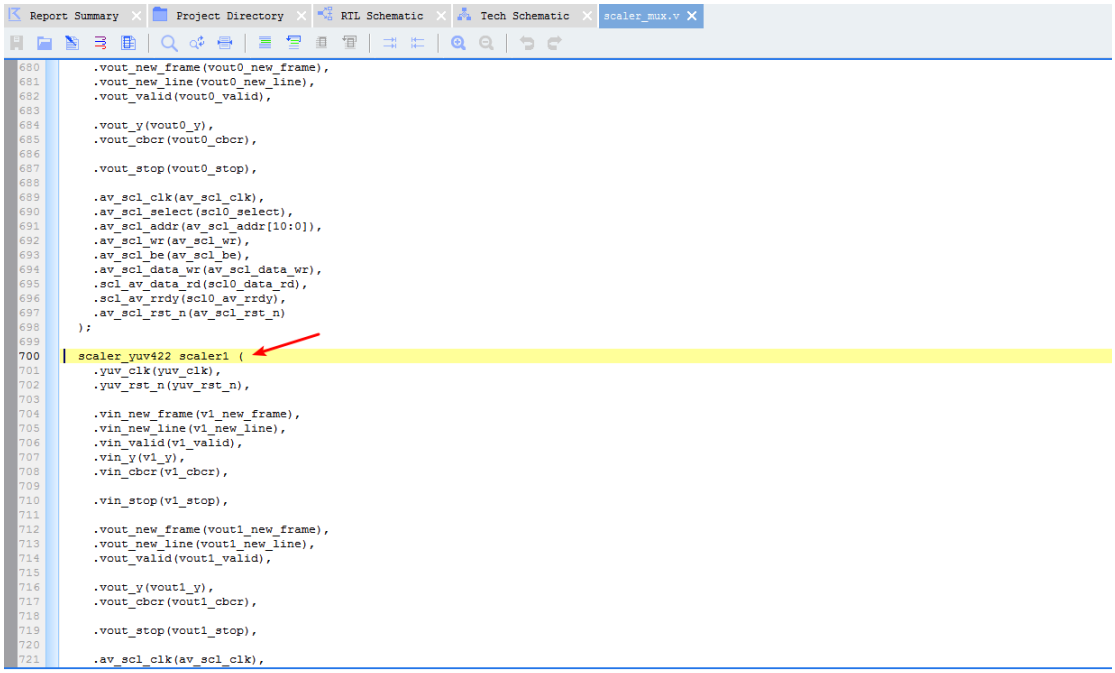

图 3-22 Verilog 文件显示结果

3.8 RTL Schematic View 和 Technology Schematic View 的交互定位

在原理图中选中某个 instance 后，右击选择菜单中的 View Object in RTL/Tech View 可以实现 RTL 视图与 Technology 视图之间交互定位。

如下图选中 RTL 中的 instance，右击选择 View Object in RTL View 会自动跳转到 Technology Schematic view 中，并将对应的 instance 突出显示。反之，在 Technology Schematic view 进行相同的操作，会跳转到 RTL Schematic view 并将对应的 instance 高亮显示外框。

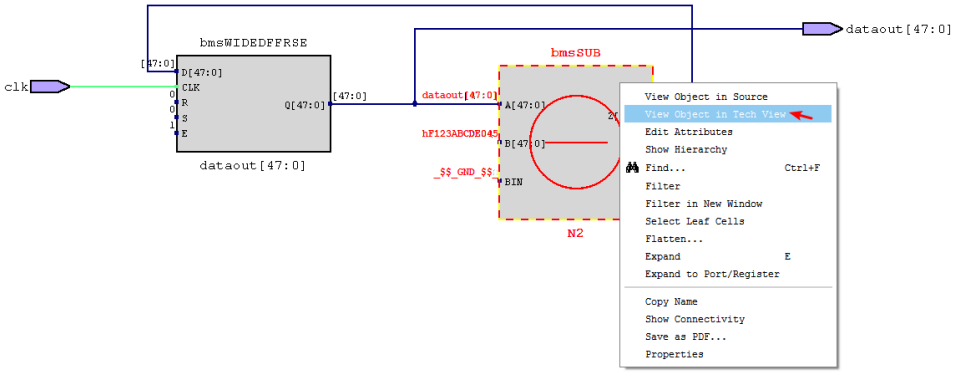

图 3-23 RTL 和 Technology 原理图交互定位

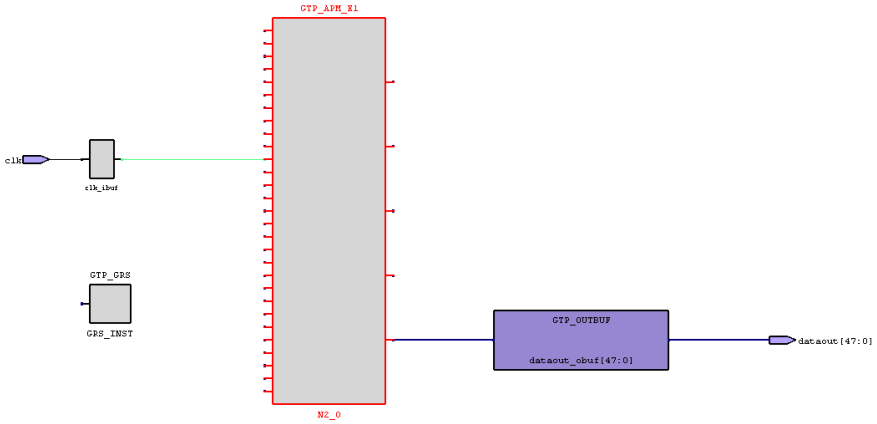

图 3-24 RTL 和 Technology 原理图交互定位

3.9 关键路径跳转原理图

在时序报告中，选中任意关键路径，右击选择右键菜单中的 View Timing Path In Schematic 可以跳转至 Schematic 查看时序路径。

如主界面 Synthesize 时序报告中，选中关键路径的 Start Piont/End Point 对象，右击选择右键菜单中的 View Timing Path In Schematic，如下图。

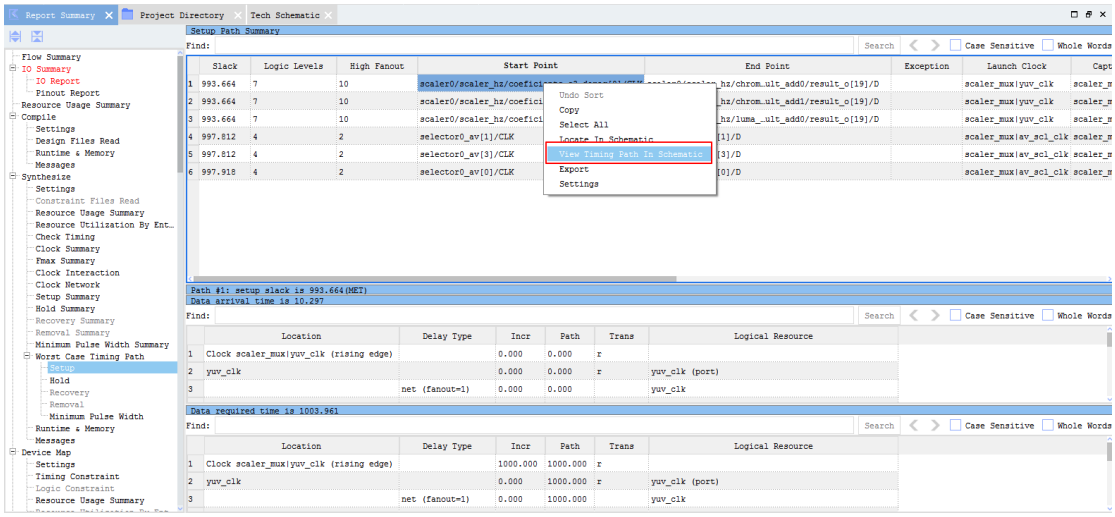

图 3-25 右键菜单选项 View Timing Path In Schematic

则会跳转主界面 Tech Schematic 显示时序路径原理图，如下图。

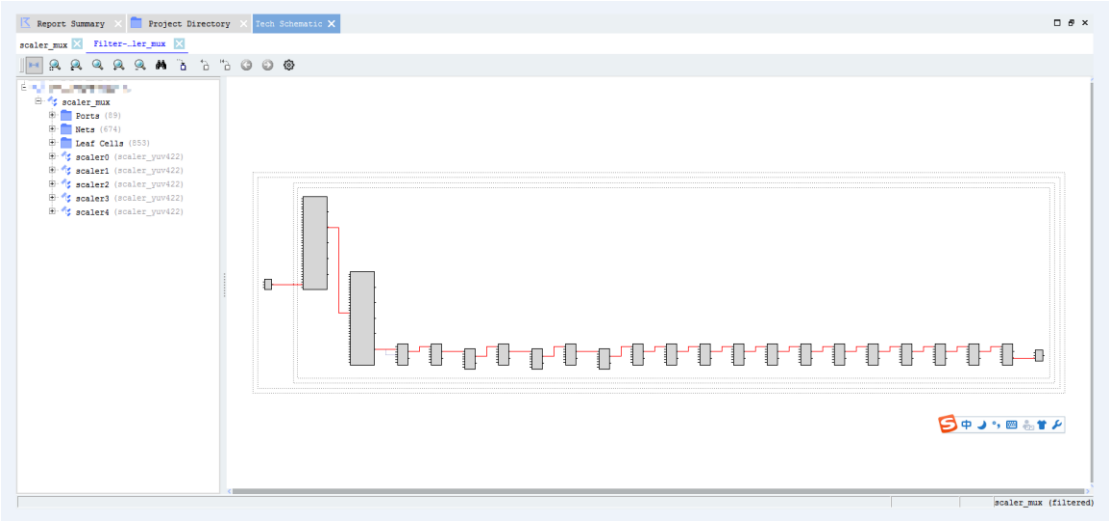

图 3-26 跳转 Tech Schematic 显示时序路径

如 TA 组件的 Report Timing 时序报告中，选中关键路径的 Start Piont/End Point 对象，右击选择右键菜单中的 View Timing Path In Schematic，则会跳转 Design Schematic 显示时序路径原理图，如下图。

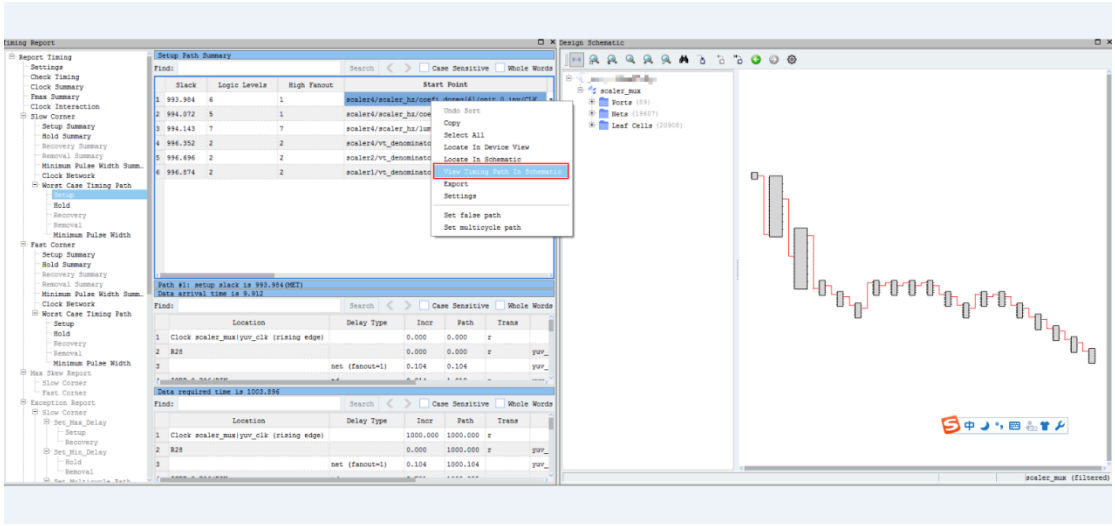

图 3-27 跳转 Design Schematic 显示时序路径

在时序报告中，选中任意关键路径中的 Logical Resource 对象，右击选择右键菜单中的 Locate In Schematic 可以跳转至 Schematic 查看该对象。

如主界面 Synthesize 时序报告中，选中关键路径中的 Pin/Port/Net 对象，右击选择右键菜单中的 Locate In Schematic，如下图。

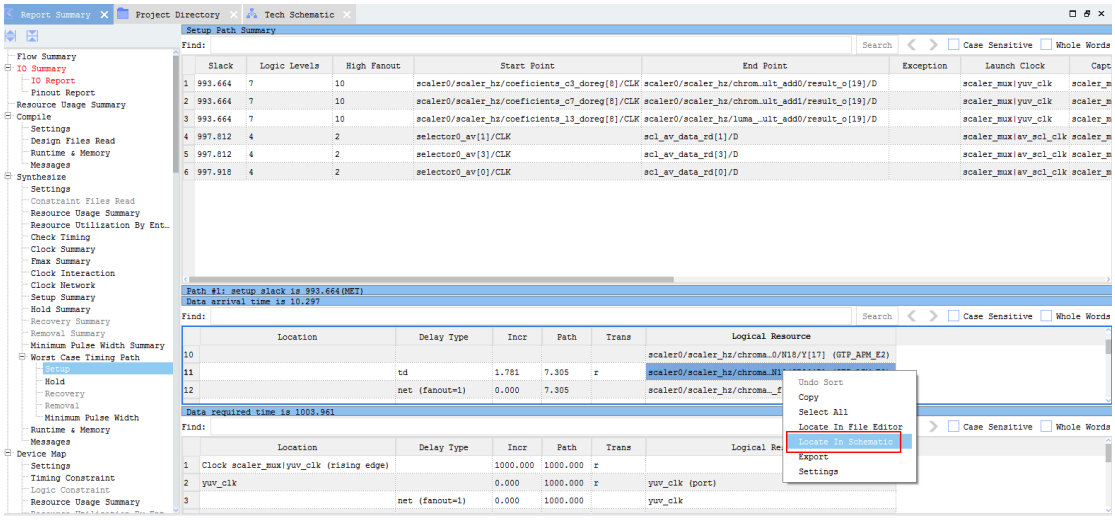

图 3-28 右键菜单选项 Locate In Schematic

则会跳转主界面 Tech Schematic 定位选中该对象，如下图。

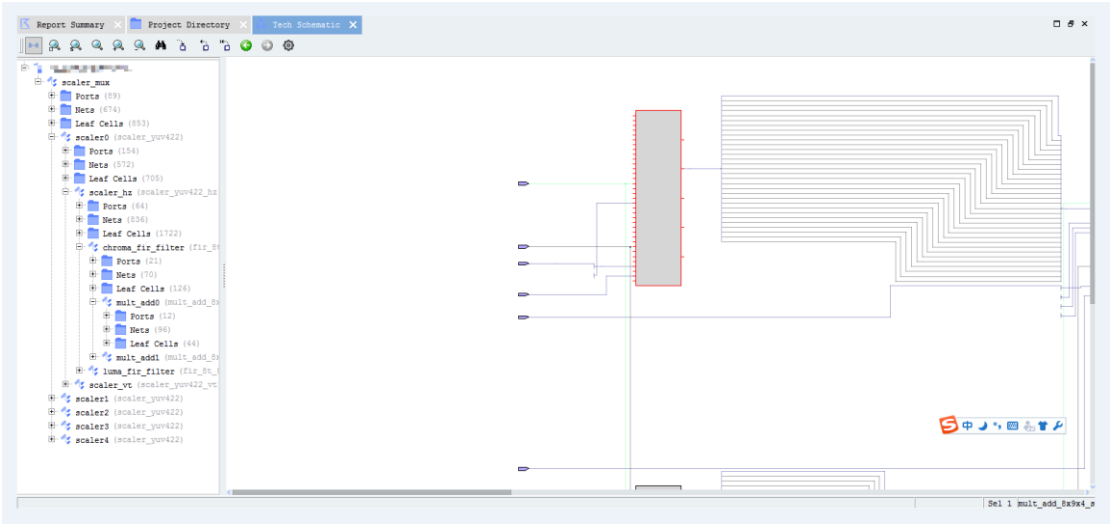

图 3-29 跳转 Tech Schematic 定位选中该对象

如 TA 组件的 Report Timing 时序报告中，选中关键路径中的 Pin/Port/Net 对象，右击选择右键菜单中的 Locate In Schematic，则会跳转 Design Schematic 定位选中该对象，如下图。

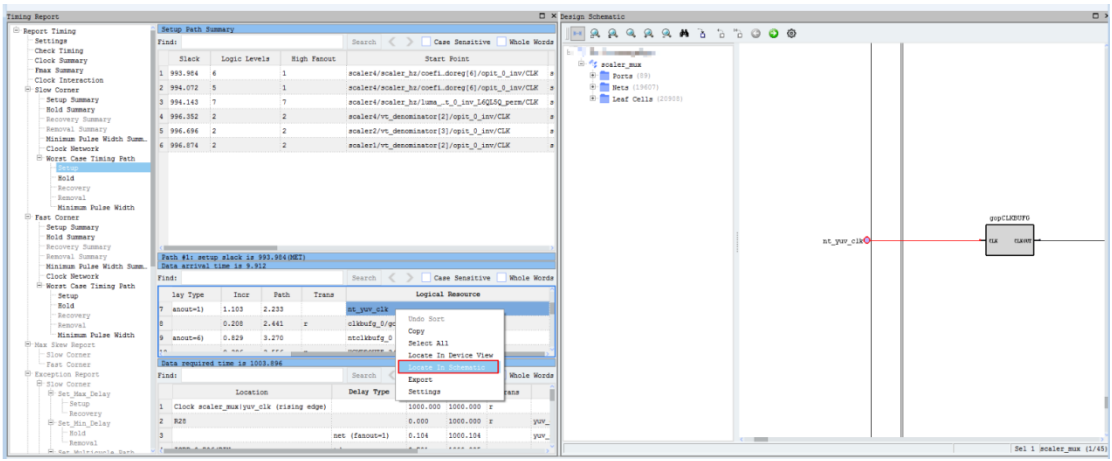

图 3-30 跳转 Design Schematic 定位选中该对象

4 shell 命令执行 ADS 综合

4. 1 使用 shell 命令执行 ADS 逻辑综合的流程

set_arch -family Compact [-device device -package package -speedgrade speedgrade]

add_constraint -logic -fdc fdc_file

add_design -verilog design_file

compile [-include_path include_path -top_module modulename]

synthesize -ads

set_arch family 可以设置为 Compact 且不能缺省，[]中的部分缺省会使用 default 设置；

add_constraint 命令完成约束文件的添加，约束文件为.fdc 类型；

add_design -verilog 命令完成需要综合的 verilog 文件的添加；

compile 命令完成 ADS 的 compile 阶段的流程，-include_path 和-top_module 可缺省；

synthesize -ads 命令完成 ADS 的 synthesize 阶段的流程，map 和 optimization 后生成并 technology 网表文件(.vm)，vm 网表文件可以用来使用第三方验证工具做验证工作。

下面为 shell 命令实现 ADS 逻辑综合的脚本示例，将 shell 命令写入如 MyScript.tcl(扩展名为.tcl)文件中，执行 pds_shell -file MyScript.tcl 命令完成 ADS 综合。

set_arch -family Logos -device PGL50H -package FBG256 -speedgrade sg7

add_constraint -logic -fdc design.fdc

add_design -verilog a.v

compile

synthesize -ads

4.2 Compile 和 Synthesize -ads Option

compile 命令完成 ADS 的 compile 阶段的流程，该阶段有相应的 option 供用户设置。compile 命令的 option 及对应的描述如下：

Option	Description
-include_path	include_path 用来设置 compile 的 include 路径，且该 option 后必须有参数；如果只有一个 include 路径，把路径名作为参数写到该 option 后面即可，如果有多个 include 路径，需要将 include 路径的 list 放到{}内
-top_module	Specify the name of Top-level Module，该 option 后须有参数，参数为 module 名；如果该项没有设置，ADS 会自动选择没有被实例化的 module 作为 top module，如果输入文件中有多个没有被实例化的 module，ADS 会根据字母排序的次序选择第一个没有被实例化的 module 作为 top 层
-system_verilog	设置后默认 Verilog 被设置成 System Verilog，该 option 后不需要参数
-allow_duplicate_modules	工程设计中一般不允许存在同名 module，不勾选本选项时，设计中存在同名 module 时会报错。当勾选本选项后，存在同名 module 可以编译成功，软件会按照 module 解析顺序将后一个 module 覆盖前一个 module，该 option 后不需要参数
-multi_file_compilation_unit	控制当前 rtl 文件中定义的变量，宏等，能否在其他 rtl 文件中生效的开关，switch 属性，不需要参数
-loop_limit	设置 for 语句循环次数的上限值。该 option 后须有参数，参数为循环上限值，须大于 2000 才会被写入
-fsm_compiler	FSM infer 的编码选项。该 option 后的值可以为 auto、off、one_hot、gray、sequential、safe、safe, one_hot、safe, gray、safe, sequential 、safe, original
-defined_parameters	在选项中设置的参数值会覆盖设计中的顶层参数值。参数名和参数值之间使用空格隔开，如果有多个参数，不同参数之间也是空格隔开，所有参数需要放到{}内，如{ADDR_WIDTH 8 DATA_WIDTH 4 ASYNC ”TRUE”}
-compiler_directives	用于设置宏定义参数，宏定义名和宏定义值间使用’=’，多个宏定义指令间使用空格隔开，如果有多个宏定义参数，需要将所有宏定义参数放到{}内，如{ADDR_WIDTH=8 DATA_WIDTH=4 ASYNC}。
-work	Specify the name of work library，后面须有参数，参数为用户要设置的 work lib 的 name

synthesize –ads 命令完成 ADS 的 synthesize 阶段的流程，该阶段有相应的 option 供用户设置。synthesize 命令的 option 及对应的描述如下：

Option	Description
-fanout_guide	设置 fanout 的最大值，option 后的值为整型变量
-rw_check_on_ram	Automatic Read/Write Check Insertion for RAM，switch 属性，不需要参数
-disable_io_insertion	Disable I/O Insertion，switch 属性，不需要参数
-enable_prepacking	LUT6D pack 开关，switch 属性，不需要参数
-resource_sharing	Resource Sharing 开关。option 后的参数可设置为 true/false，设置为 true 时，进行资源共享，设置为 false 时，不进行资源共享
-min_controlset_size	调整具有同一控制端口的 FF 的最小值，option 后的值为整型变量
-retiming	Retiming 开关，可在综合中通过移动寄存器的方式优化时序，switch 属性，不需要参数
--pipelining	Pipelining 开关，可在综合中通过移动寄存器的方式优化面积，switch 属性，不需要参数
-max_path	设置综合时序报告中报告的最差时序路径的数量，option 后的值为整型变量
-nworst	设置综合时序分析时，报出的每个 end point 最多能保存的 path 数量，option 后的值为整型变量
-path_logic_level_gt_than	综合时序分析时，报出的每个 setup/recovery 时序路径的 Logic Levels 必须大于设定值，小于该值的时序路径将不予显示，option 后的值为整型变量
-frequency	用于 infer clock 中，如果无法根据 PLL 的参数以及参考时钟计算正确的时钟频率，就会用这个 option 的设置值为时钟频率来创建时钟，需大于 1.0000 才会被写入
-use_clocked_period_for_unconstraint_io	为用户没有约束 IO delay 的 port 增加 IO delay 约束，提升时序约束的覆盖程度，switch 属性，不需要参数

5 Attribute Reference

attribute 的设置方式有两种，一种是在 HDL 源文件中设置，一种是在 fdc 约束文件中设置。fdc 中设置的约束作用在 synthesize 阶段，若要设置的 attribute 在 compile 阶段有效，则需将 attribute 设置到 HDL 源文件中。

其中 ADS 已支持的 attribute 中只支持在 HDL 源文件中设置的 attribute 如下：

- syn_black_box
- syn_looplimit
- syn_keep
- syn_noprune
- syn_preserve

用户通过 fdc 文件设定 attribute，设定方式如下：

define_attribute <ObjectName> <AttributeName> <AttributeValue>

用户在 HDL 源文件中设定 attribute ，设定方式如下：

(Verilog-1995 meta-comments)

/*synthesis <AttributeName> = <AttributeValue> */

或者 (Verilog-2001 attribute)

(* <AttributeName> = <AttributeValue> *)

以下是 ADS 支持的 Synthesis Attributes 及说明。

5.1 syn_allow_retiming

用于指定 register 是否允许进行 Retiming 优化。

主要应用于在 Option 中打开全局的 Retiming 优化时，用该属性指定部分寄存器不进行 Retiming 优化。

syn_allow_retiming Values

Global	Object
Yes	Register

属性值	作用
1	允许进行 Retiming 优化，在综合配置选项中开启全局 Retiming 优化时该 register 可以进行优化。
0	不允许进行 Retiming 优化，在综合配置选项中开启全局 Retiming 优化时该 register 不进行优化。

Syntax Specification

FDC file	define_attribute {i:instance} syn_allow_retiming {0} 全局属性： define_global_attribute syn_allow_retiming {0}
Verilog	object /* synthesis syn_allow_retiming = 1 */;

Verilog Example

```
module test (a, rs, out, clk) ;
input [15:0] a;
input rs;
input clk;
output out;

reg [15:0] ar;

always @ (posedge clk)
   begin
        ar <= a;
  end

wire t;
assign t = |ar;

reg tr /* synthesis syn_allow_retiming = 0 */ ;
```

```
always @ (posedge clk)
    begin
        tr <= t;
    end

reg out;
always @ (posedge clk)
    begin
        if (rs)
            out <= 1;
    else
out <= tr;
end

endmodule
```

该属性必须搭配配置选项中的 Retiming 选项使用，即需先勾选上 Retming 的全局选项，如范例中勾选了 Retiming 选项且未应用该属性之前，在综合阶段会进行 retiming 优化。

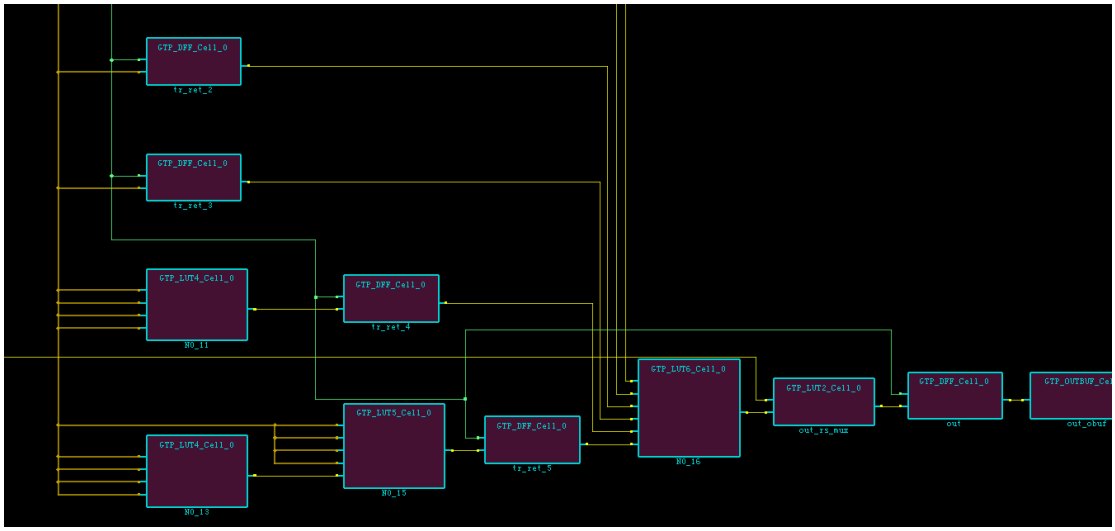

图 5-1 勾选 Retiming 进行优化

应用了 syn_allow_retiming = 0 即使勾选了 Retiming 选项之后，也不会进行 retiming 优化，如图，该属性主要用于控制局部的寄存器不做 retiming 优化。

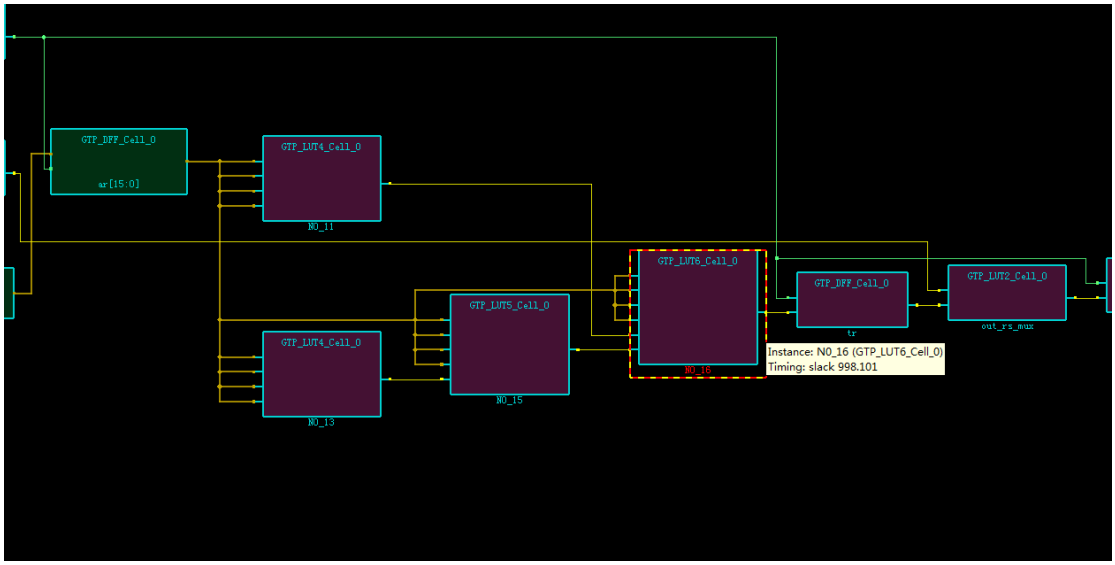

图 5-2 未进行 Retiming 优化

5.2 syn_allowed_resources

用于指定工程中允许使用的 DRM 的最大值，当使用的 DRM 资源超过设置的限制值后，超出的 inferred ram 综合会自动 map 到分布式 RAM 或寄存器上，实例化的 GTP ram 不会受影响。仅支持设置全局属性。

syn_allowed_resources Values

Global	Object
Yes	Top module

属性值	作用
blockrams=%d	当使用的 DRM 资源超过设置的限制值后，超出的 inferred ram 综合会自动 map 到分布式 RAM 或寄存器上

Syntax Specification

FDC file	define_global_attribute {syn_allowed_resources} {blockrams=100} 或 define_attribute {v:top module} {syn_allowed_resources} {blockrams=100}
Verilog	object /* synthesis syn_allowed_resources = "blockrams=10" */;

Verilog Example

```
module top# (
    parameter DATA_WIDTH = 10,
    parameter ADDR_WIDTH = 15
) (
    input CLK,
   input WE,
    input [ADDR_WIDTH-1:0] ADDR,
    input [DATA_WIDTH-1:0] DI,
    output wire [DATA_WIDTH-1:0] DO
)/* synthesis syn_allowed_resources = "blockrams=5" */;

sub sub1(CLK,WE,ADDR,DI,DO);

endmodule
```

```
module sub # (
    parameter DATA_WIDTH = 10,
    parameter ADDR_WIDTH = 15
) (
    input CLK,
    input WE,
    input [ADDR_WIDTH-1:0] ADDR,
    input [DATA_WIDTH-1:0] DI,
    output reg [DATA_WIDTH-1:0] DO
);

reg [DATA_WIDTH-1:0] mem [2**ADDR_WIDTH-1:0] ;

always @ (posedge CLK)
    if (WE) mem[ADDR] <= DI;
    else DO <= mem[ADDR];

endmodule
```

在范例中未应用该属性之前，综合后 inferred ram 会全部 map 到 DRM 上，资源报告如下:

Mapping Summary:

Total LUTs：0 of 243600（0.00%）

LUTs as dram：0 of 75400（0.00%）

LUTs as logic：0

Total Registers：0 of 487200（0.00%）

Total Latches：0

DRM36K/FIFO：

Total DRM = 10.0 of 480（2.08%）

应用了 syn_allowed_resources = "blockrams=5" 之后，工程中最多能使用 5 个 DRM，超出的部分会自动 map 到 Distributed RAM 和 FF，综合后资源报告中可以看到资源的汇总如下

Mapping Summary：

Total LUTs：2862 of 243600（1.17%）

LUTs as dram：2560 of 75400（3.40%）

LUTs as logic：302

Total Registers：5 of 487200（0.00%）

Total Latches：0

DRM36K/FIFO：

Total DRM = 5.0 of 480（1.04%）

5.3 syn_black_box

用于指定一个模块为 black box

综合时无需知道某个模块是如何实现的，只需要知道其输入输出接口即可，此时可以设置为 black_box。

syn_black_box Values

Global	Object
No	View

属性值	作用
1	object 被设置为 black_box
0	不是 black_box，与不设置效果一样
N/A	与设置为 1 效果一致，object 被设置为 black_box

Syntax Specification

Verilog	object /* synthesis syn_black_box = 1 */; 或 object /* synthesis syn_black_box */;

Verilog Example

```
module top(clk, in1, in2, out1, out2);

input clk;
input [1:0] in1;
input [1:0] in2;

output [1:0] out1;
output [1:0] out2;
```

```
add                 U1 (clk, in1, in2, out1);
black_box_add U2 (in1, in2, out2);

endmodule

module add (clk, in1, in2, out1);

input clk;
input [1:0] in1;
input [1:0] in2;

output [1:0] out1;
reg [1:0] out1;

always @(posedge clk)
    begin
        out1 <= in1 + in2;
    end
endmodule

module black_box_add(A, B, C) /* synthesis syn_black_box = 1 */;

input [1:0] A;
input [1:0] B;

output [1:0] C;

assign C = A + B;
```

endmodule

如范例中未应用该属性之前在 compile 网表中 black_box_add 模块内部是可以看到的，并且该模块与 add 模块重复，在综合阶段会被优化

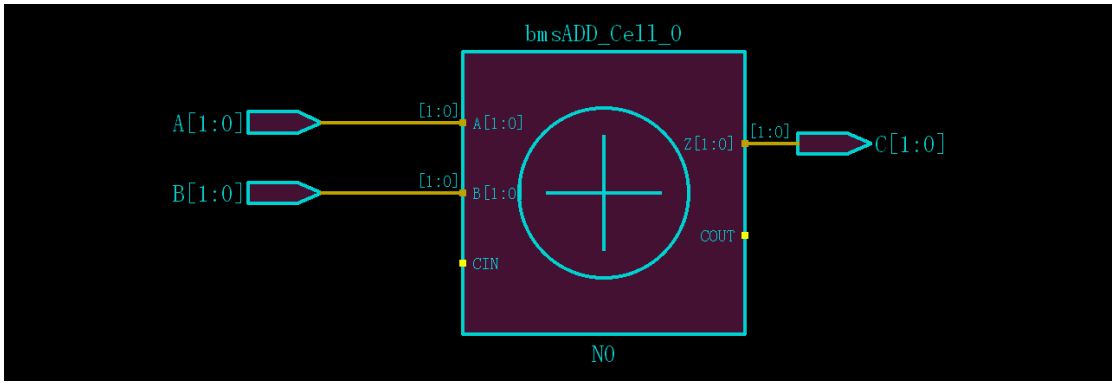

图 5-3 Compile 网表

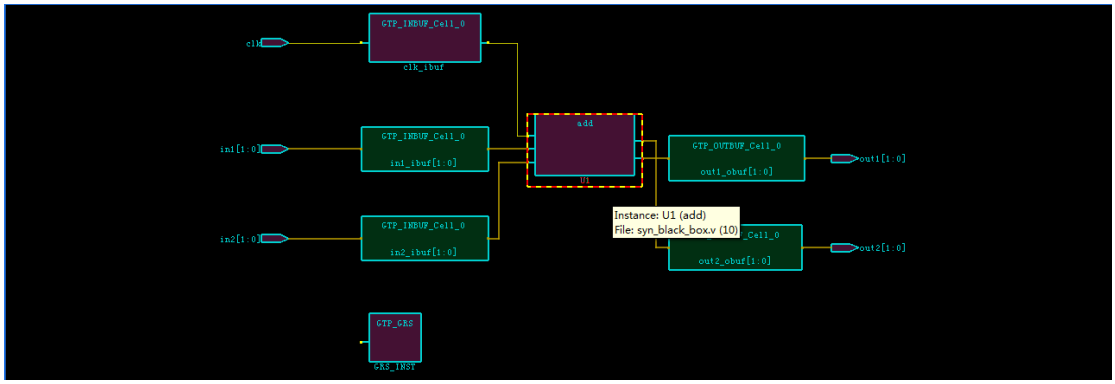

图 5-4 Synthesize 网表

应用了 syn_black_box = 1 之后 black_box_add 模块内部就变成了黑盒，只能看到端口信息，内部功能不可见，故不会被优化，综合将该黑盒模块传递下去。

注意：由于设置了黑盒无法被 flatten，此时在 Device Map 阶段会报错：

E: Flow-0088: Design instance U2 has not been completely flattened or contains black boxes.

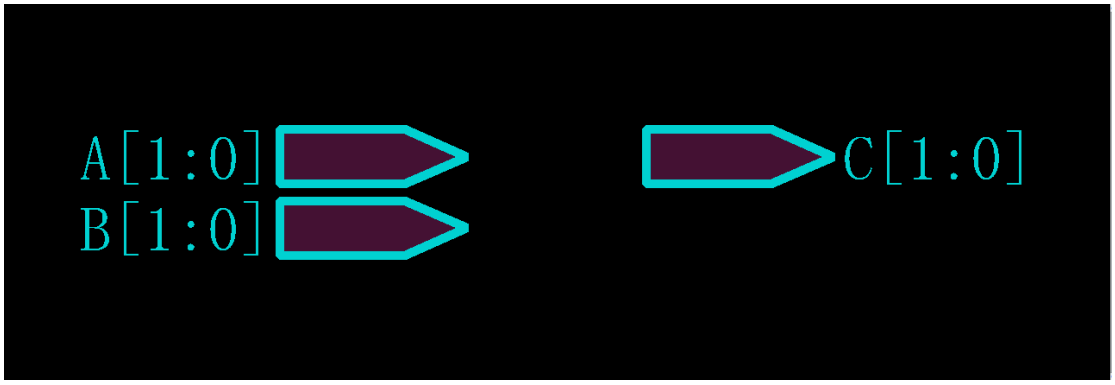

图 5-5 应用 syn_black_box 属性 compile 网表

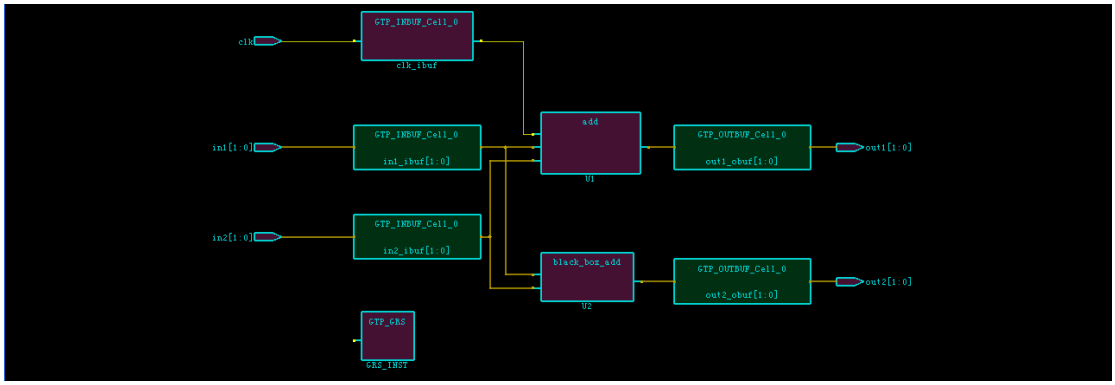

图 5-6 应用 syn_black_box 属性 synthesize 网表

5.4 syn_direct_enable

该属性用于指定直接控制 DFF 的 enable 信号。

支持穿透 inv 及设置有 syn_hier = "fixed/hard"的模块。

不支持 syn_direct_enable 在一个 DFF 中重复使用，且不支持在一个 DFF 中和 syn_direct_reset/ syn_direct_set 同时使用。

syn_direct_enable Values

Global	Object
No	port, net

属性值	作用
0	不是 direct enable 信号，与不设置效果一样
1	设置为 direct enable 信号

Syntax Specification

fdc file	define_attribute {object} {syn_direct_enable} {value}
verilog	object /* synthesis syn_direct_enable = value */;

Verilog Example

```
module direct_enable #(
    parameter size=5
    ) (
    input [size-1:0] d1,
    input clk,
    input e1,e2,
    input e3 /* synthesis syn_direct_enable = 1 */,
    output reg [size-1:0] q1
    );

    always @(posedge clk) begin
        if (e1 & e2 & e3)
            q1 = d1;
    end

endmodule
```

如范例中未使用该属性前，direct enable 信号由 e1, e2, e3 共同组成，如下图

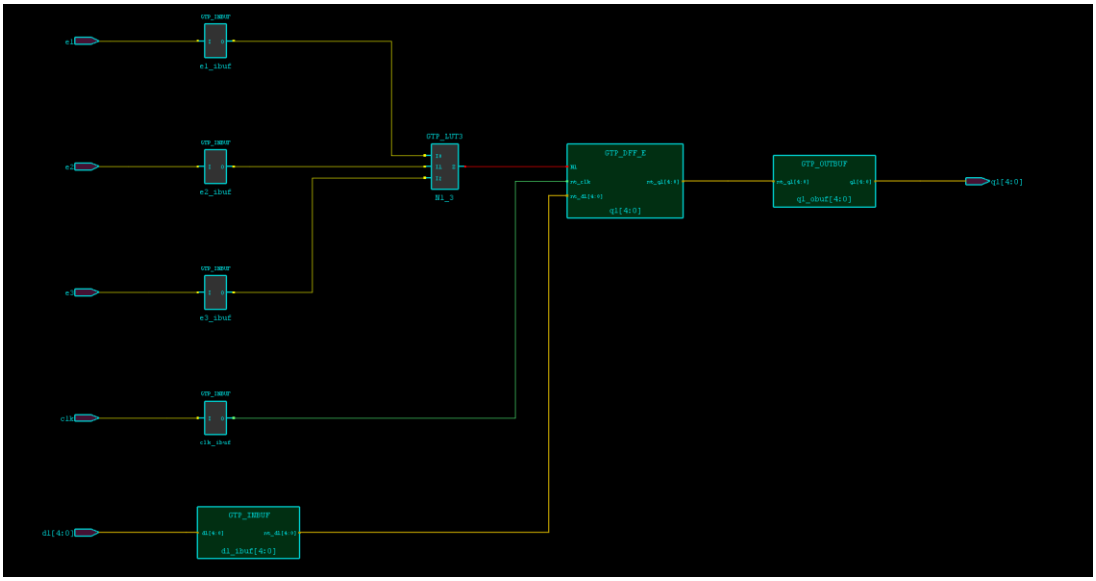

图 5-7 未应用 syn_direct_enable 属性

应用了 synthesis syn_direct_enable = 1 之后，e3 直接作为单独的 enable 信号，如下图

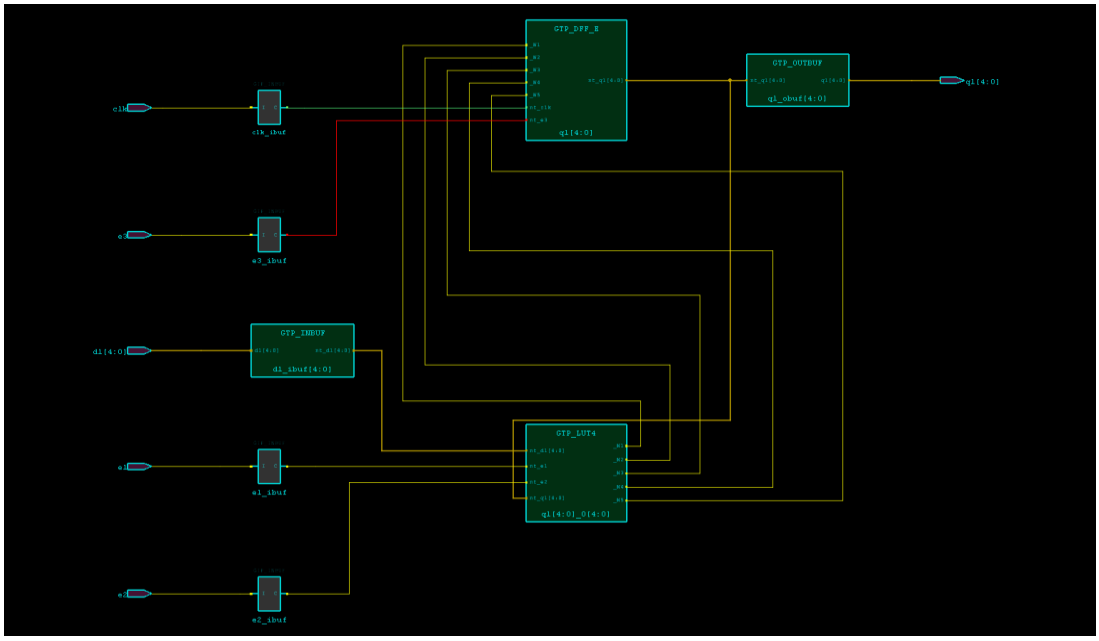

图 5-8 应用 syn_direct_enable 属性

5.5 syn_direct_reset

该属性用于指定直接控制同步 DFF 的 reset 信号。

支持穿透 inv 及设置有 syn_hier = "fixed/hard"的模块。

不支持 syn_direct_reset 在一个 DFF 中重复使用，且不支持在一个 DFF 中和 syn_direct_enable/ syn_direct_set 同时使用。

syn_direct_reset Values

Global	Object
No	port, net

属性值	作用
0	不是 direct reset 信号，与不设置效果一样
1	设置为 direct reset 信号

Syntax Specification

fdc file	define_attribute {object} { syn_direct_reset} {value}
verilog	object /* synthesis syn_direct_reset = value */;

Verilog Example

```
module direct_reset #(
    parameter size = 5
    ) (
    input a,
    input b,
    input c /* synthesis syn_direct_reset = 1 */,
    input clk,
    input [size-1:0] din,
    output reg [size-1:0] dout
    );

    always @(posedge clk) begin
```

```
        if (a == 1'b1 || b == 1'b1 || c == 1'b1)
            dout = 0;
        else
            dout = din;
    end

endmodule
```

如范例中未使用该属性前，direct reset 信号由 a, b, c 共同组成，如下图

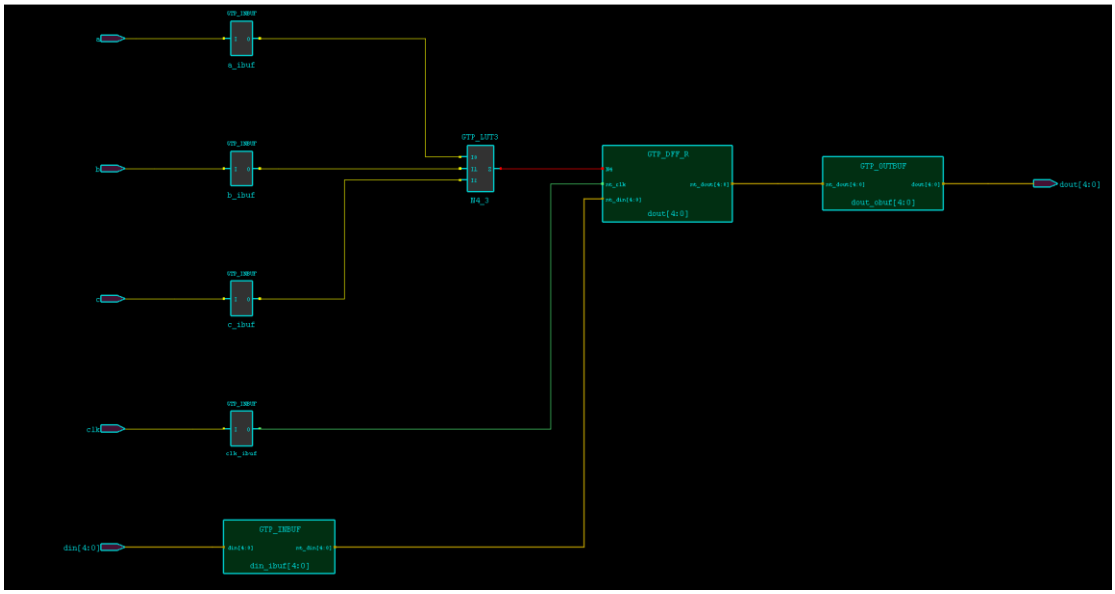

图 5-9 未应用 syn_direct_reset 属性

应用了 synthesis syn_direct_reset = 1 之后，c 直接作为单独的 reset 信号，如下图

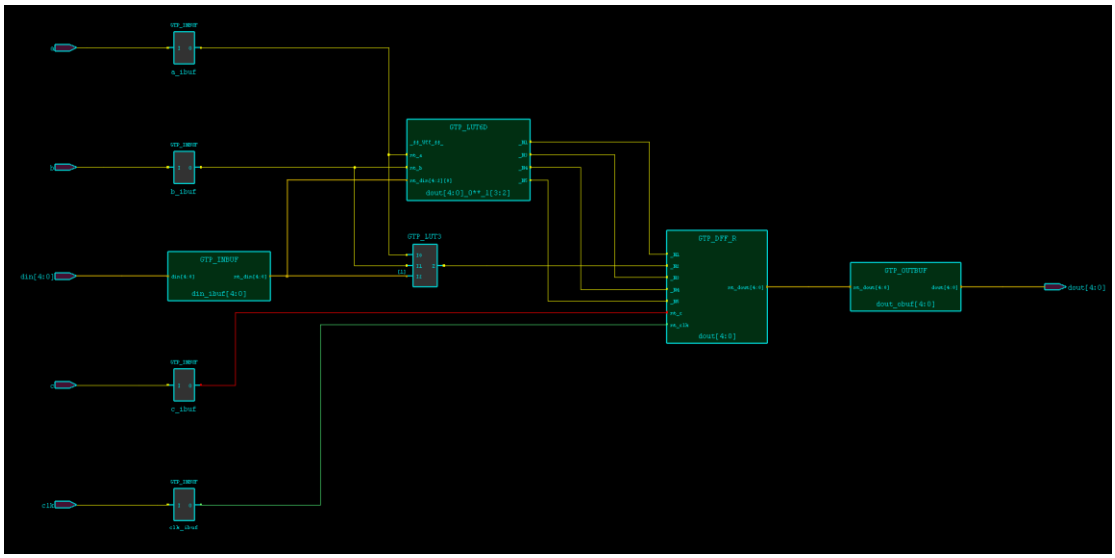

图 5-10 应用 syn_direct_reset 属性

5.6 syn_direct_set

该属性用于指定直接控制同步 DFF 的 set 信号。

支持穿透 inv 及设置有 syn_hier = "fixed/hard"的模块。

不支持 syn_direct_set 在一个 DFF 中重复使用，且不支持在一个 DFF 中和 syn_direct_enable/ syn_direct_reset 同时使用。

syn_direct_set Values

Global	Object
No	port, net

属性值	作用
0	不是 direct set 信号，与不设置效果一样
1	设置为 direct set 信号

Syntax Specification

fdc file	define_attribute {object} { syn_direct_set} {value}

verilog	object /* synthesis syn_direct_set = value */;

Verilog Example

```
module direct_set #(
    parameter size = 5
    ) (
    input a,
    input b,
    input c /* synthesis syn_direct_set = 1 */,
    input clk,
    input [size-1:0] din,
    output reg [size-1:0] dout
    );

    always @(posedge clk) begin
        if (a == 1'b1 || b == 1'b1 || c == 1'b1)
            dout = ~0;
        else
            dout = din;
    end

endmodule
```

如范例中未使用该属性前，direct set 信号由 a, b, c 共同组成，如下图

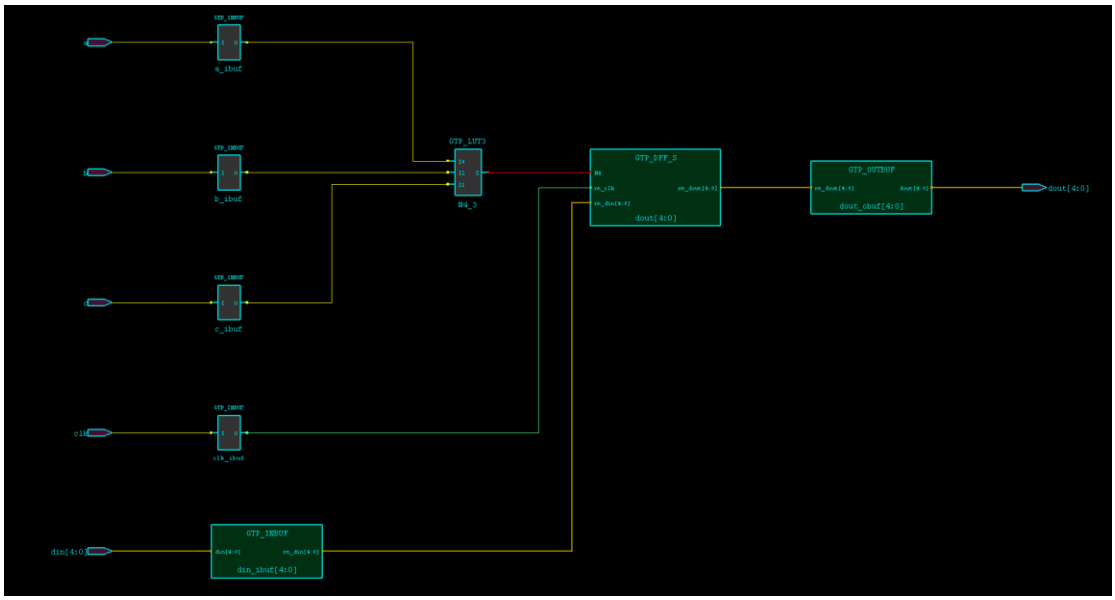

图 5-11 未应 syn_direct_set 属性

应用了 synthesis syn_direct_set = 1 之后，c 直接作为单独的 set 信号，如下图

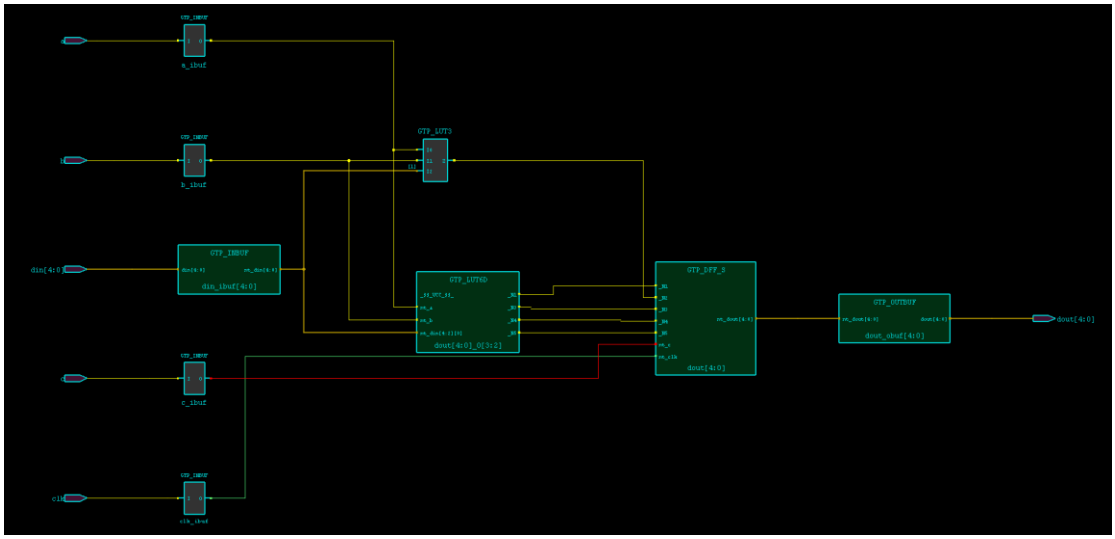

图 5-12 应用 syn_direct_set 属性

5.7 syn_dspstyle

用于指定乘法、加（减）法逻辑的实现方式

syn_dspstyle Values

Global	Object
Yes	Verilog module, operator

属性值	作用
block_mult	将乘法器、加（减）法器逻辑 map 到 APM
logic	将乘法器逻辑 map 到 LUT

Syntax Specification

FDC file	define_attribute {i:instance} syn_dspstyle {value} 全局属性： define_global_attribute syn_dspstyle {value}
Verilog	object /* synthesis syn_dspstyle = "block_mult"*/

Verilog Example

```
module test (clk,reset,enable,a,b,q);
    input clk;
    input reset;
    input enable;
    input signed [1:0] a;
    input signed[1:0] b;
    output signed [3:0] q;

    reg signed [1:0] a_reg;
    reg signed[1:0] b_reg;
    wire signed [3:0] q1 /*synthesis syn_dspstyle = "block_mult" */;
    reg signed [3:0] q ;

    assign q1 = a_reg * b_reg;

    always @(posedge clk)    begin
        if (reset)    begin
```

```
            a_reg <= 0;
            b_reg <= 0;
            q <= 0;
        end else if (enable) begin
            a_reg <= a;
            b_reg <= b;
            q <= q1 + q;
        end
    end
endmodule
```

如范例中未应用该属性之前，会使用普通逻辑实现乘法运算。

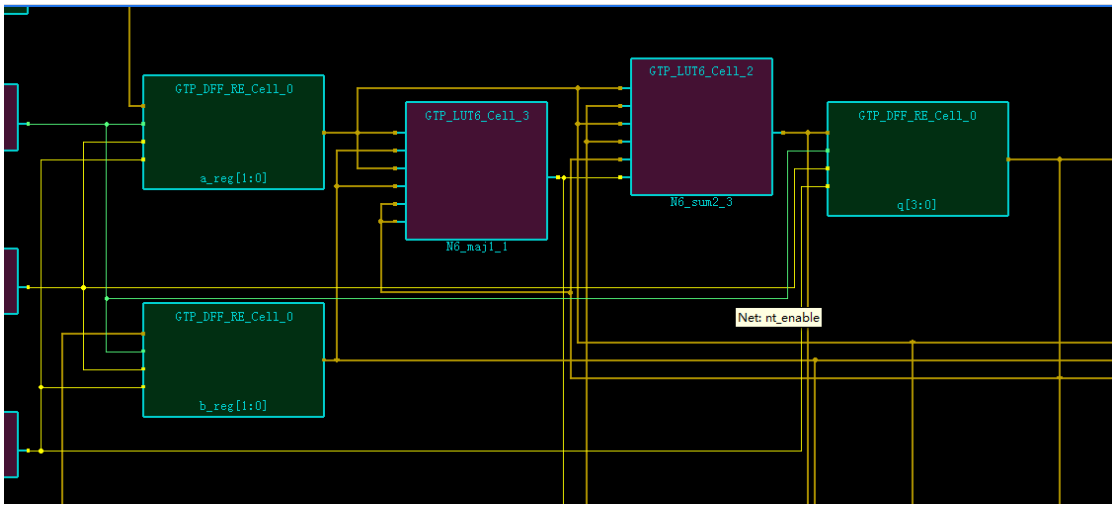

图 5-13 未应用 syn_dspstyle 属性

应用了 syn_dspstyle = "block_mult" 之后，会使用 APM 实现乘法运算。

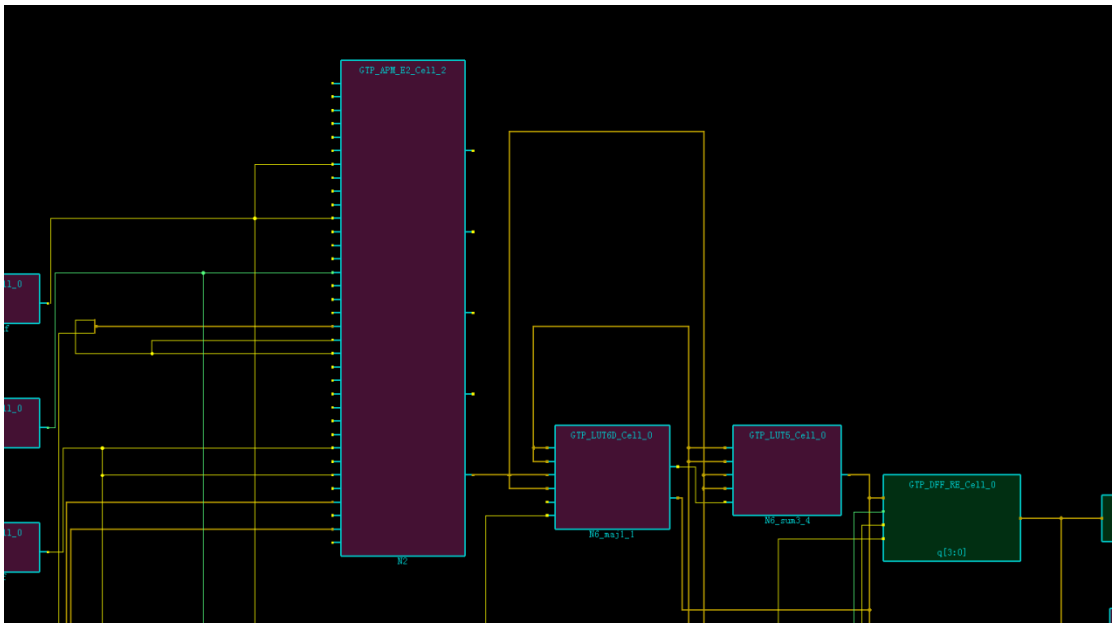

图 5-14 应用 syn_dspstyle 属性

5.8 syn_encoding

用于指定有限状态机编码类型

syn_encoding Values

Global	Object
Yes	状态寄存器

属性值	作用
onehot	每个状态只有一位为 1，可减少组合逻辑，速度最快，可优化时序，但使用较多寄存器
gray	相邻状态转换时只有一个状态发生翻转，与 onehot 相比，使用寄存器较少
sequential	自然数编码，相同条件下使用寄存器最少
original	保持源文件中的原始编码
safe	防止状态机被锁在非法状态，默认为 onehot,safe 编码方式。一旦 FSM 进入任意非法状态，可以通过 safe logic 第一时间将 FSM 跳转回合法状态，可增强设计的稳定性

safe,onehot	使用独热码编码方式，并防止状态机被锁在非法状态
safe,gray	使用格雷码编码方式，并防止状态机被锁在非法状态
safe,sequential	使用自然数码编码方式，并防止状态机被锁在非法状态
safe,original	保持源文件中的原始编码，并防止状态机被锁在非法状态

Syntax Specification

FDC file	define_attribute {i:instance} syn_encoding {value} 全局属性： define_global_attribute syn_encoding {value}
Verilog	object /* synthesis syn_encoding = "onehot"*/

Verilog Example

```
module test (clk,rst,data,out);

input clk,rst,data;

output out;

reg out;

reg [2:0] ps /* synthesis syn_encoding = "gray" */;

reg [2:0] ns;

parameter S0=3'b000, S1=3'b001, S2=3'b010, S3=3'b011, S4=3'b100,

 S5=3'b101, S6=3'b110;

always@(posedge clk or posedge rst)

begin

 if(rst)

  ps <= S0;

 else

  ps <= ns; end
```

```
always@(ps or data)
begin
 ns = S0;
 case (ps)
  S0: if(data)
  ns = S1;
  else ns = S0;
  S1: if(data)
  ns = S1;
  else ns = S2;
  S2: if(data)
  ns = S1;
  else ns = S3;
  S3: if(data)
  ns = S4;
  else ns = S0;
  S4: if(data)
  ns = S5;
  else ns = S2;
  S5: if(data)
```

```
    ns = S1;
    else ns = S6;
    S6: if(data)
    ns = S1;
    else ns = S3;
    default: ns = S0;
  endcase
end
always@(ps or data)
begin
  if((ps == S6) && (data == 1))
    out = 1;
  else
    out = 0;
end
endmodule
```

如范例中未应用该属性之前，有限状态机默认的编码方式为 onehot，并在 run.log 中打出如下信息：

```
I: Encoding type of FSM 'ps_fsm[2:0]' is: onehot.
I: Encoding table of FSM 'ps_fsm[2:0]':
I: from  ps[2] ps[1] ps[0]
```

```
I: to   ps_6 ps_5 ps_4 ps_3 ps_2 ps_1 ps_0
I: 000 => 0000001
I: 001 => 0000010
I: 010 => 0000100
I: 011 => 0001000
I: 100 => 0010000
I: 101 => 0100000
I: 110 => 1000000
```

需要注意的是，本例中状态机正常跳转下不会出现 ns 取值为 111 的情况。即使在设计中添加了”default: ns = S0;”语句，综合也不会将 111 添加到状态机逻辑当中，因此不能将 default 语句和属性中”safe”值等价起来。如果工程会受环境影响进入非法状态，请使用”safe”属性来防止状态机卡死在非法态。

应用了 syn_encoding = "gray" 之后，有限状态机的编码方式为 gray 码，并在 run.log 中打出如下信息：

```
I: Encoding type of FSM 'ps_fsm[2:0]' is: gray.
I: Encoding table of FSM 'ps_fsm[2:0]':
I: from   ps[2] ps[1] ps[0]
I: to   ps_2 ps_1 ps_0
I: 000 => 000
I: 001 => 001
I: 010 => 011
```

I: 011 => 010

I: 100 => 110

I: 101 => 111

I: 110 => 101

注意：syn_encoding 在 attribute 上设置的值的优先级是高于 option 中的 FSM Encoding 设置的值的。

5.9 syn_hier

综合工具默认会做跨层级优化，即综合过程中将层级打平再进行整体优化，可以打破模块的边界，在整个设计范围内进行逻辑优化，虽然综合时间会增加，但是可以优化逻辑，从而减少资源使用，提高时钟频率。

使用 syn_hier 属性可控制如何进行跨层级优化。

syn_hier Values

Global	Object
No	View

属性值	作用
fixed	保持原层次化结构，完全不做跨层次的优化，以模块为单位进行优化，可以减少综合所用的时间。
hard	保持原层次化结构，但不同层次之间可以做常量传递优化
remove	移除标记的当前模块的层次，只能移除 GTP 的 wrapper 层，并且该层级中只有一个 GTP instance（如 PLL IP 中的 GTP_PLL_WRAPPER）。
macro	保留当前层级的所有接口和内容，必须保证该层级没有例化其他 instance，否则会报错

Syntax Specification

FDC file	define_attribute {v:module} syn_hier {value}
Verilog	object /* synthesis syn_hier = "value" */;

Verilog Example

```
module sub(clk_sub, din_sub, dout_sub) /* synthesis syn_hier = "fixed" */ ;
   input clk_sub, din_sub;
   output reg dout_sub;

   always @(posedge clk_sub) begin
        dout_sub <= din_sub;
   end

endmodule

module test(clk, dout);
   input clk;
   output dout;

   wire din;

   assign din = 1'b1;

   sub u_sub(clk, din, dout);

endmodule
```

如范例中未应用该属性之前综合后的网表中 sub 会被优化，如图

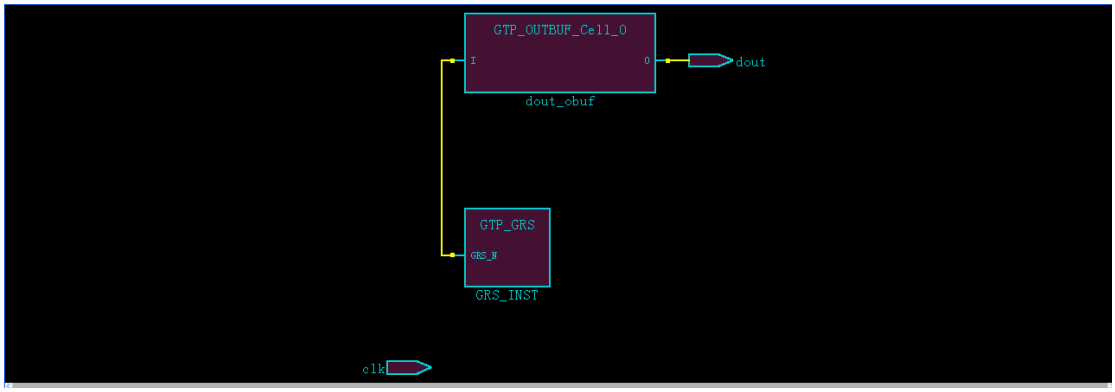

图 5-15 未应用 syn_hier 属性

应用了 syn_hier = "fixed"之后可以防止 sub 模块被优化

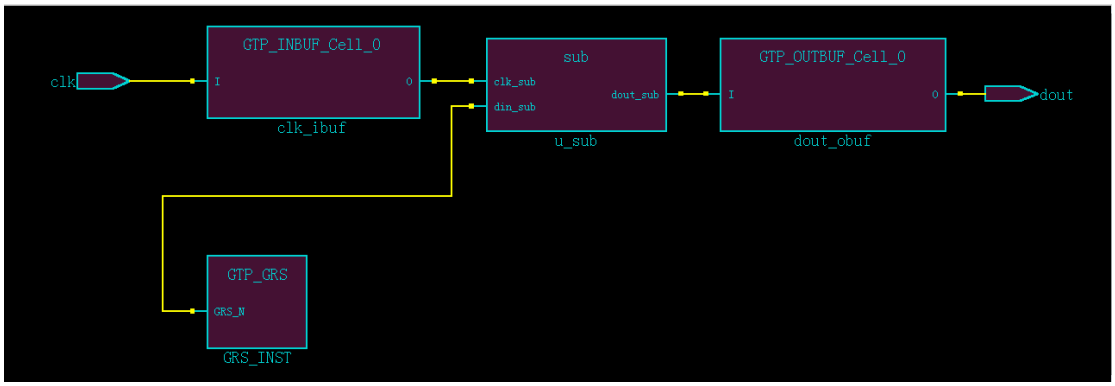

图 5-16 应用 syn_hier 属性

5.10 syn_insert_buffer

用于指定 clock net 使用的 clock buffer 的类型。

syn_insert_buffer Values

Global	Object
No	Verilog net, port

属性值	作用
GTP_CLKBUFG	在该对象后插入 GTP_CLKBUFG
GTP_CLKBUFR	在该对象后插入 GTP_CLKBUFR，Compact 系列不支持
GTP_CLKBUFX	在该对象后插入 GTP_CLKBUFX，当前仅支持 Logos2 和 Titan2 系列

GTP_CLKBUFM	在该对象后插入 GTP_CLKBUFX，当前仅支持 Logos2 和 Titan2 系列

Syntax Specification

Verilog	object /* synthesis syn_insert_buffer = "GTP_CLKBUFG" */

Verilog Example

```
module test(clk1,clk2,clk3,clk4,
            din1,din2,din3,din4,en1,en2,
            dout1,dout2,dout3,dout4);
  input clk1;
  input clk2 /* synthesis syn_insert_buffer = "GTP_CLKBUFG" */;
  input clk3;
  input clk4;
  input en1, en2;
  input [7:0]din1,din3;
  input din2;
  input din4;
  output [7:0]dout1,dout3;
  output dout2,dout4;
  reg [7:0]dout1,dout3;
  reg dout2,dout4;
  wire en;

  assign en = en1 & en2 ;

  always @(posedge clk1)
  begin
    if (en) dout1 <= din1;
  end
```

```
always @(posedge clk2)
begin
    if (en) dout2 <= din2;
end

always @(posedge clk3)
begin
    if (en) dout3 <= din3;
end

always @(posedge clk4)
begin
    if (en) dout4 <= din4;
end
endmodule
```

如范例中未应用该属性之前综合后的网表中 sub 会被优化，如下图

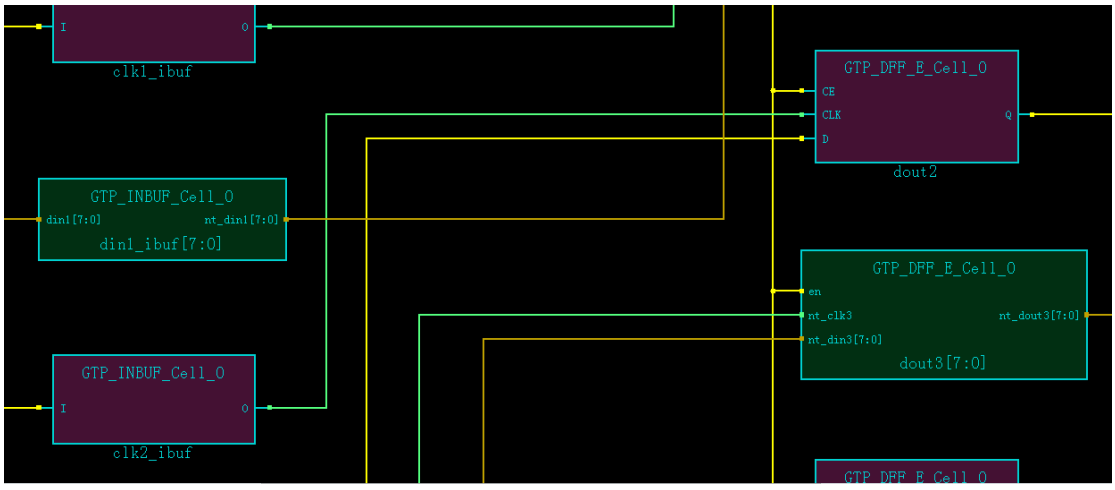

图 5-17 未应用 syn_insert_buffer 属性

应用了 syn_insert_buffer = "GTP_CLKBUFG"之后综合后在 clk2 之后会插入 GTP_CLKBUFG，如下图

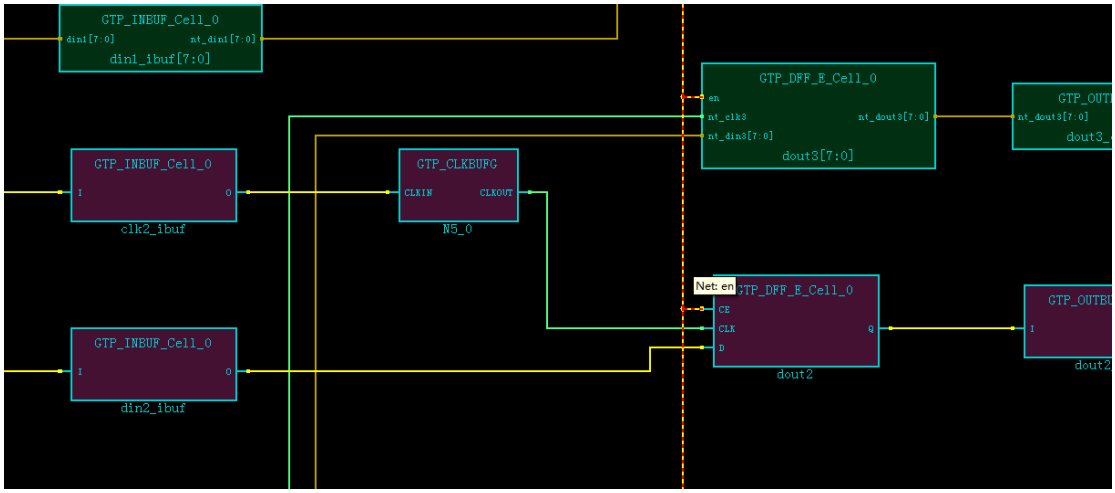

图 5-18 应用 syn_insert_buffer 属性

5.11 syn_keep

用于指定需要保留的 net，对带有 syn_keep 属性的 net，通过插入特定的 keepbuf 的方法，综合工具保证 net 不会被优化掉，并保持原名称出现在综合结果文件中。

syn_keep Values

Global	Object
No	Verilog net, port

属性值	作用
1	保持该 net 不被优化
0	允许该 net 被优化

Syntax Specification

Verilog	object /* synthesis syn_keep= 1 */

Verilog Example

```
module test(out1, out2, out3, clk, a,b,c);
    output out1, out2, out3;
    input clk;
    input a,b,c;
    wire w1 /* synthesis syn_keep=1 */;
    wire w2 /* synthesis syn_keep=1 */;
    wire w3 ;
    reg out1, out2, out3;
    assign w1=a&b;
    assign w2=a&c;
    assign w3=w1&w2;

    always @(posedge clk)
```

```
begin
    out3<=w3;
end
endmodule
```

如范例中未应用该属性之前综合后的网表中 w1 和 w2 两根 net 会被优化，如下图

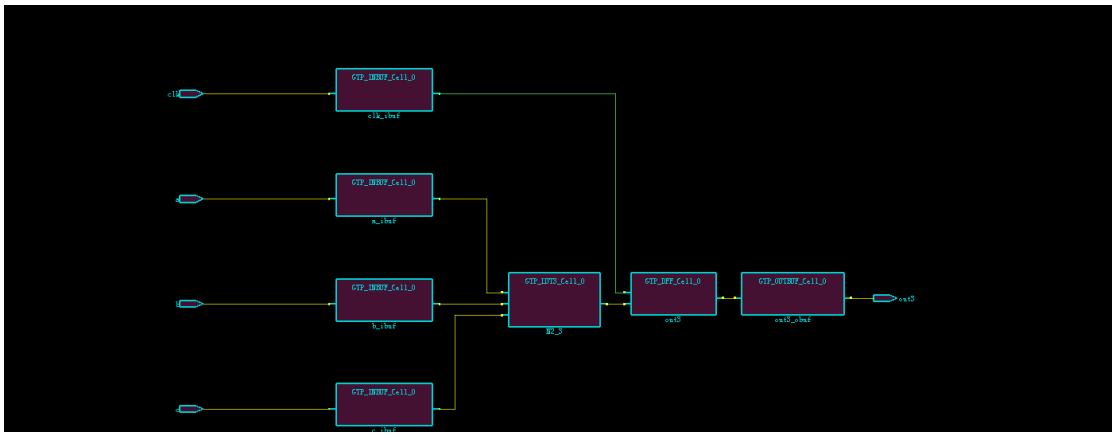

图 5-19 未应用 syn_keep 属性

应用了 syn_keep=1 综合后的网表中 w1 和 w2 两根 net 不会被优化，如下图

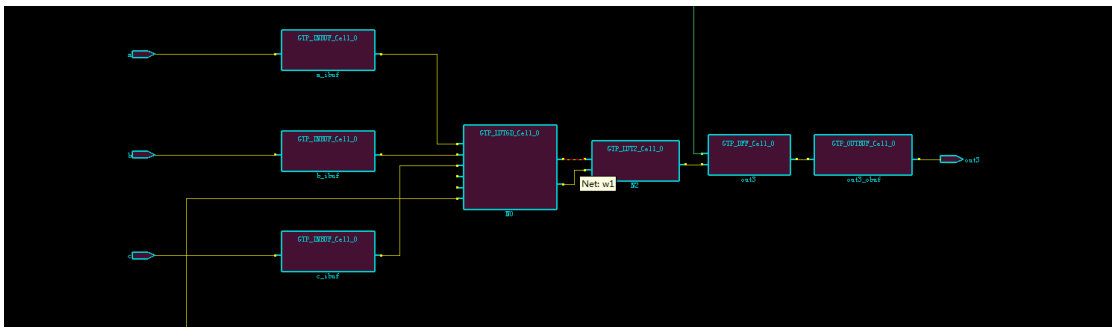

图 5-20 应用 syn_keep 属性

注意：若设置了 syn_keep 属性的 net 在设计中没有实际的连接，则即使在 compile 阶段插入了 keepbuf，在 Synthesize 阶段还是会被优化。

5.12 syn_looplimit

该属性用于指定 Verilog 循环语句（for 循环、while 循环或 repeat 循环）迭代次数的上限。

综合工具中循环最大迭代次数为 2000，若超过两千次迭代，需要设置最大迭代次数大于设计中的实际循环次数才能编译成功。

syn_looplimit Values

Global	Object
Yes	Verilog for-loop/while-loop/repeat-loop

属性值	作用
integer	指定 Verilog 循环语句（for 循环、while 循环或 repeat 循环）迭代次数的上限

Syntax Specification

verilog	object /* synthesis syn_looplimit = value */;

Verilog Example

```
module syn_looplimit_forloop_later (CLK, D, result);
    parameter N = 2003;
    input CLK;
    input D;
    output result;

    wire    register [0:N];

    GTP_DFF #(.INIT(1'b1))
    t1 (.CLK(CLK), .D(D), .Q(register[0]));

    genvar i;
    generate
        for(i = 0; i < N-1; i = i + 1) /*synthesis syn_looplimit = 4000 */
```

```
        begin
            GTP_DFF #(.INIT(1'b1))
            t2 (.CLK(CLK), .D(register[i]), .Q(register[i+1]))/* synthesis
syn_noprune=1 */;
        end
    endgenerate

    assign result = register[N-1];

endmodule
```

如范例中未使用该属性前，Compile 报错，提示循环超限，如下图

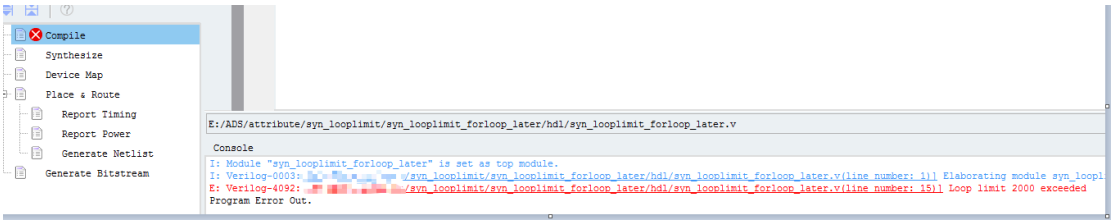

图 5-21 未应用 syn_looplimit 属性

应用了 syn_looplimit = 4000 之后，可综合成功，如下图

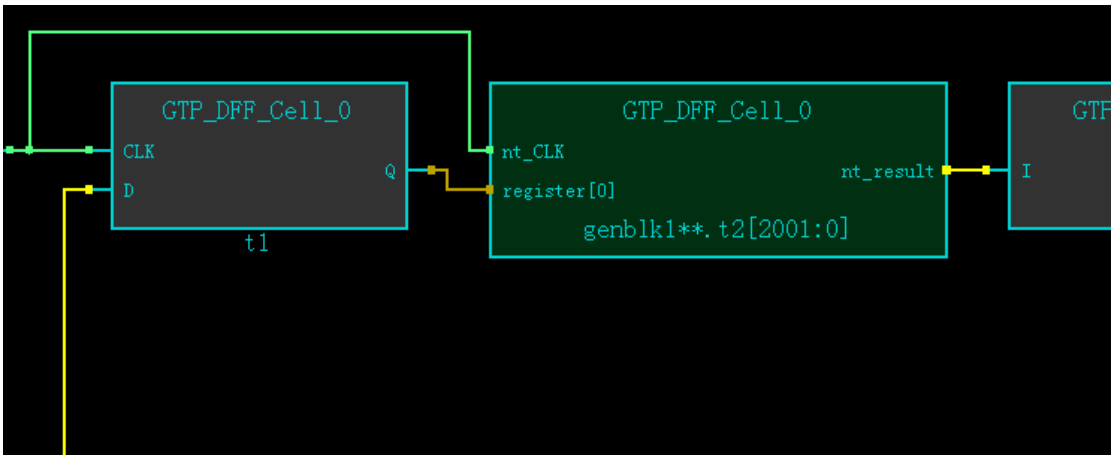

图 5-22 应用 syn_looplimit 属性

5.13 syn_maxfan

设置 fanout 最大值，与配置选项的 fanout guide 相似。使用 buffering 去减少输入端口的 fanout，通过复制逻辑减少被寄存器或组合逻辑驱动的 net。

syn_maxfan Values

Global	Object
Yes	module, port, net, instance,register

属性值	作用
integer	设置目标信号的 fanout 最大值

Syntax Specification

fdc file	define_global_attribute { syn_maxfan } {value} define_attribute {object} { syn_maxfan } { value }
verilog	object /* synthesis syn_maxfan = value */;

Verilog Example

```
module syn_maxfan_net (clk,a,b,c,q);
    input clk,a,b;
    input [7:0] c;
    output [7:0] q;

    wire en /* synthesis syn_maxfan = 4 */;
    reg [7:0] q;

    assign en = a & b;

    always @ (posedge clk) begin
        if (en)
            q = c;
    end
```

```
endmodule
```

如范例中未使用该属性前，综合后的网表中目标信号 fanout 不受限制，如下图

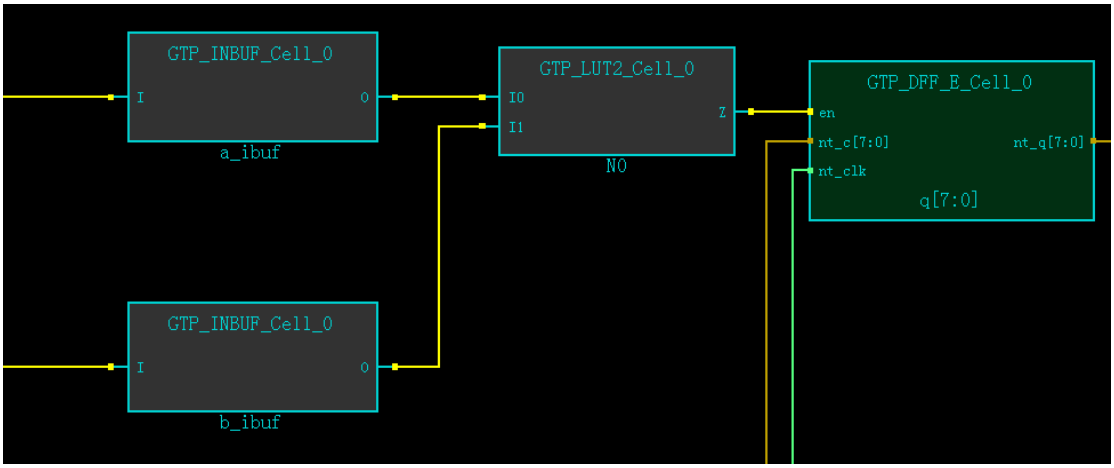

图 5-23 未应用 syn_maxfan 属性

应用了 syn_maxfan = 4 之后，综合后的网表中目标信号通过复制 LUT2 来实现 fanout 最大为 4，如下图

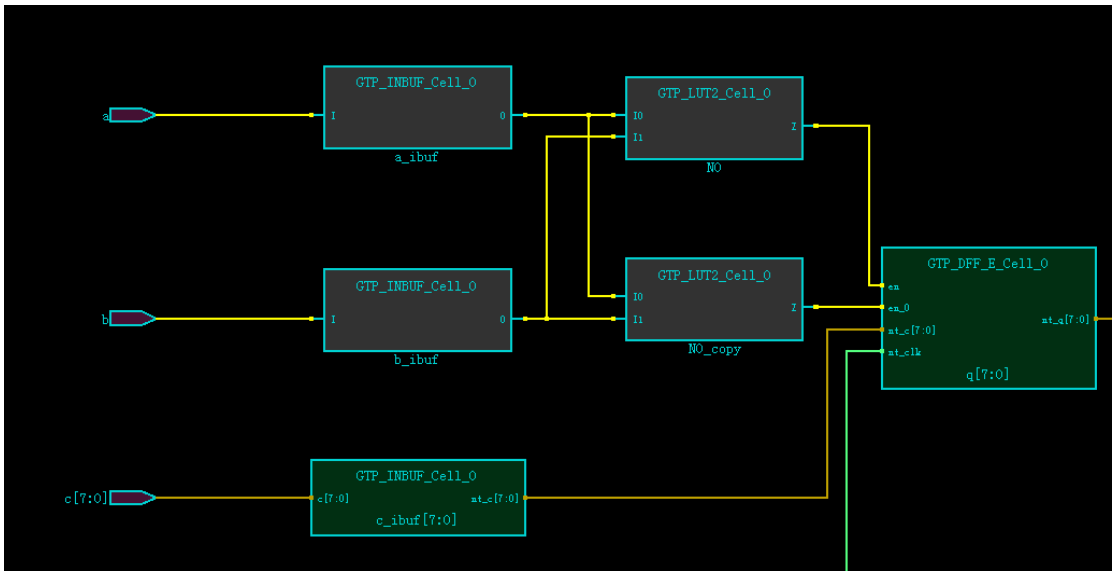

图 5-24 应用 syn_maxfan 属性

注意：syn_maxfan 属性设置在 CLK 信号和异步复位/置位信号上是无效的。

syn_maxfan 在 attribute 上设置的值的优先级是高于 option 中的 Fanout Guide 值。

5.14 syn_noprune

该属性用于指定不要删除的 instance.

对带有 syn_noprune 属性的 instance，综合工具不会对被约束的对象做优化，即使它的输入都悬空，或者输出没有驱动任何其他单元。

syn_noprune Values

Global	Object
No	module, instance, register

属性值	作用
0	不保留在综合过程中被删除的 instance
1	保留在综合过程中被删除的 instance

Syntax Specification

verilog	object /* synthesis syn_noprune = value */;

Verilog Example

```
module syn_noprune_1_ins (input a, b, output c);
    assign c=b;
    sub i1 (a);
endmodule

module sub (input a);
    leaf i2 (a,) /* synthesis syn_noprune=1*/;
endmodule

module leaf (input a, output b) ;
    assign b = a;
endmodule
```

如范例中未使用该属性前，综合后的网表中不会保留在综合过程中被删除的 instance(leaf)，如下图

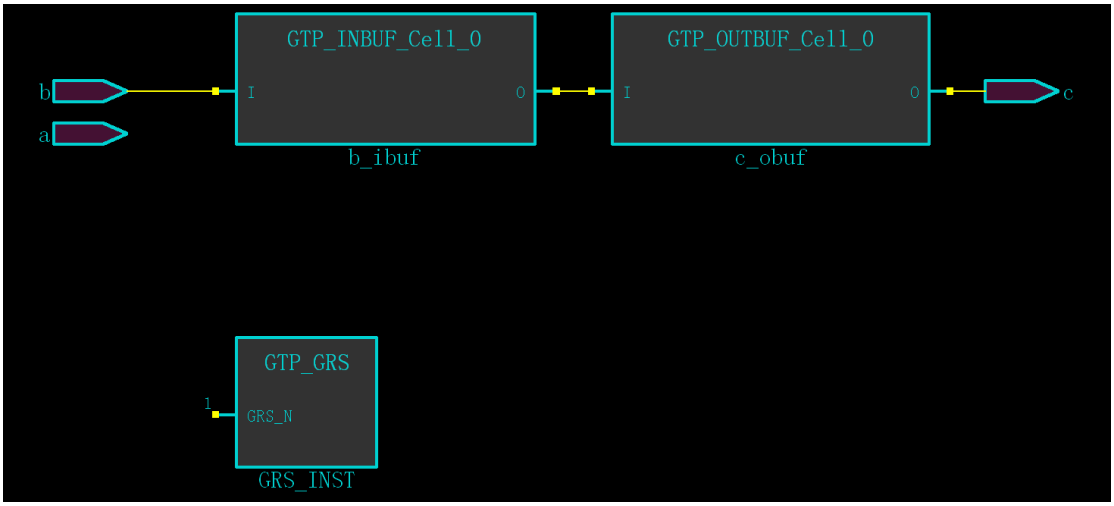

图 5-25 未应用 syn_noprune 属性

应用了 syn_noprune=1 之后，综合后的网表中会保留在综合过程中被删除的 instance(leaf)，如下图

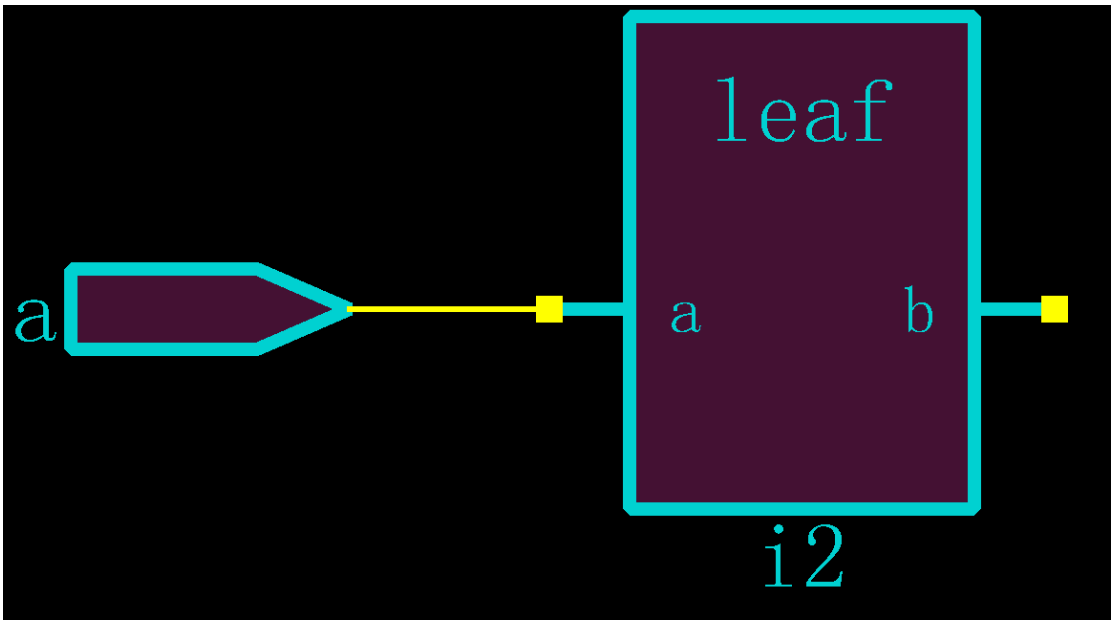

图 5-26 应用 syn_noprune 属性

注意： syn_noprune 只能将 inst 保持到综合阶段，若该 inst 没有连接，在 dev_map 还是会被优化。

5.15 syn_preserve

该属性用于控制是否保留在综合过程中被优化的 sequential (registers).

syn_preserve Values

Global	Object
No	module, port, register

属性值	作用
0	不保留在综合过程中被优化的 registers
1	保留在综合过程中被优化的 registers

Syntax Specification

verilog	object /* synthesis syn_preserve = value */;

Verilog Example

```
module syn_preserve_1_reg (out1,out2, data, rst, clk);
    output [7:0] out1,out2;
    input [7:0] data;
    input rst,clk;

    reg [7:0] out1;
    reg [7:0] out2 /* synthesis syn_preserve=1 */;

    // create the 8-bit register
    always @(posedge clk or negedge rst )
    begin
        if (!rst ) begin
            out1 <= 8'b0;
            out2 <= 8'b0;
        end
        else begin
```

```
            out1 <= data;
            out2 <= data;
        end
    end

endmodule
```

如范例中未使用该属性前，综合后的网表中不会保留在综合过程中被优化的 register(out2[7:0])，如下图

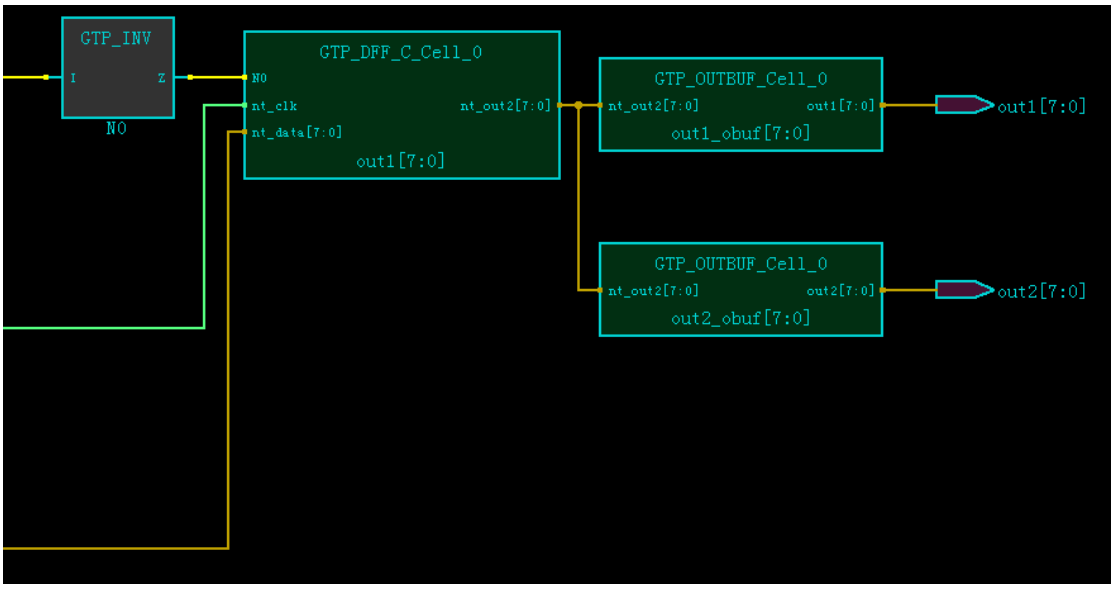

图 5-27 未应用 syn_preserve 属性

应用了 syn_preserve=1 之后，综合后的网表中保留在综合过程中被优化的 register(out2[7:0])，如下图

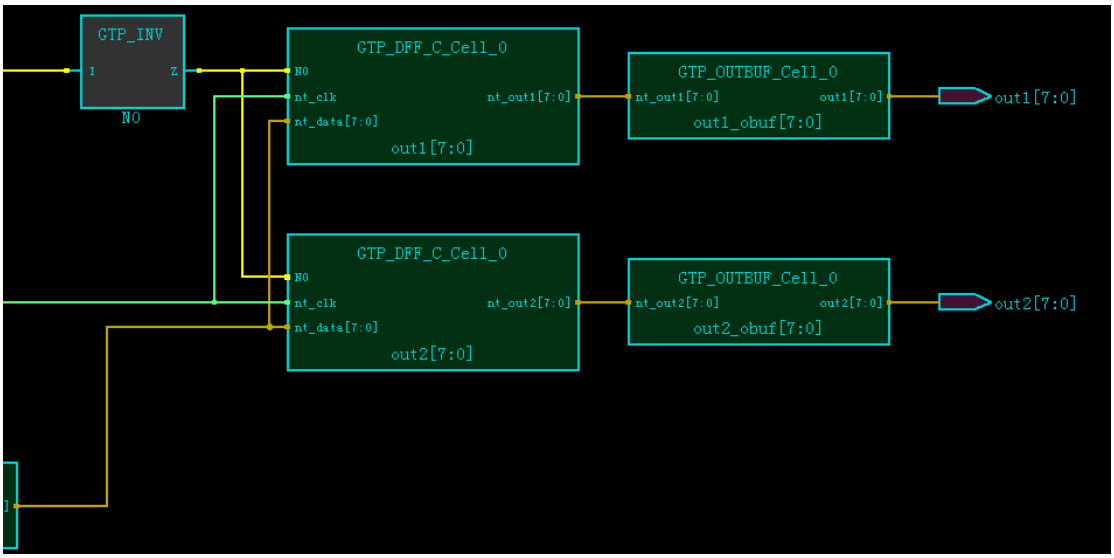

图 5-28 应用 syn_preserve 属性

5.16 syn_ramstyle

用于指定 inferred ram 的实现方式。

syn_ramstyle Values

Global	Object
Yes	module, ram-signal

属性值	作用
block_ram	指定 inferred RAM 可以 map 到 DRM 资源
lut_ram	指定 inferred RAM 可以 map 到 Distributed RAM 资源
registers	指定 inferred RAM 可以 map 到 Registers
no_rw_check	Infer RAM 时不插入 bypass 逻辑去做读写检查，可能会出现 RTL 仿真和综合后仿真的不一致的情况
block_ram,low_power	指定 inferred RAM 可以 map 到 DRM 资源，同时优先匹配 data width,再匹配 address

注：属性值 select_ram 与 lut_ram 等效

Syntax Specification

fdc file	define_global_attribute { syn_ramstyle } {value} define_attribute {object} { syn_ramstyle } { value }
verilog	object /* synthesis syn_ramstyle ="value" */;

Verilog Example

```
module syn_ramstyle_lut_ram_port (
```

```
    clk,we,waddr,raddr,din,q
    ) ;
    parameter ADDR_W = 8;
    parameter DATA_W = 8;

    input clk,we;
    input [ADDR_W - 1 : 0] waddr,raddr;
    input [DATA_W - 1 : 0] din;
    output [DATA_W - 1 : 0] q /* synthesis syn_ramstyle="lut_ram" */;

    reg [DATA_W - 1 : 0] q;
    reg [DATA_W - 1 : 0] mem [(2**ADDR_W) - 1 : 0];

    always @(posedge clk)
        if (we)
            mem[waddr] <= din;

    always @(posedge clk)
        q <= mem[raddr];

endmodule
```

如范例中未使用该属性前，综合后 infer 出 DRM，如下图

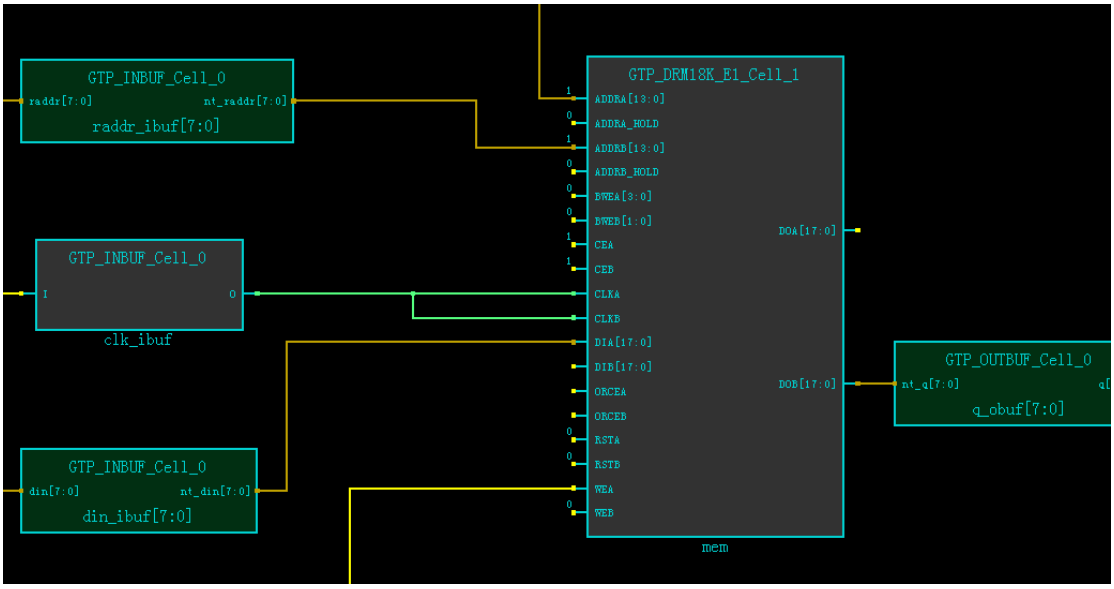

图 5-29 未应用 syn_ramstyle 属性

应用了 syn_ramstyle = "lut_ram"之后，综合后可以 map 到 Distributed RAM，如下图

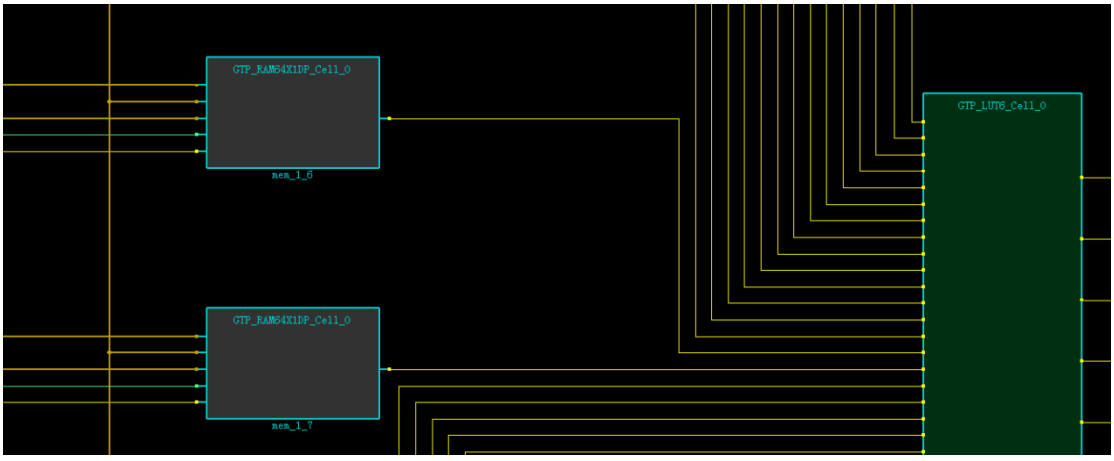

图 5-30 应用 syn_ramstyle 属性

5.17 syn_reduce_controlset_size

该属性用于指定具有同一控制端口的寄存器的最小值。与配置选项中的 Minimum Control-set Size 功能一致。

对带有 syn_reduce_controlset_size 属性的 register，当具有同一控制端口 CLK、CE、同步 SET/RESET 等的寄存器数小于设置的值时，会将对应的 FF 的部分或全部控制引脚移至 D 输入端。当大于等于设置值时，不进行优化。

Control set size 优化会忽略 IO-register，即和 port 相连、并且没有设置属性 /*synthesis syn_useioff=0*/的 DFF。

syn_reduce_controlset_size Values

Global	Object
Yes	module

属性值	作用
integer	当具有同一控制端口的寄存器数小于该值时，会将对应的 FF 的部分或全部控制引脚移至 D 输入端

Syntax Specification

fdc file	define_global_attribute { syn_reduce_controlset_size } {value} define_attribute {object} { syn_reduce_controlset_size } { value }
verilog	object /* synthesis syn_reduce_controlset_size = value */;

Verilog Example

```
module test (
  input clk,
  input en,
  input [3:0] d,
  output [3:0] q,
  output [3:0] q_e
  ) /* synthesis syn_reduce_controlset_size = 5 syn_useioff = 0*/;

  DFF U_DFF (
      .clk (clk),
      .d (d),
      .q (q)
      );

  DFF_E U_DFF_E (
      .clk (clk),
```

```
        .en (en),
        .d (d),
        .q_e (q_e)
        );

endmodule


module DFF (
   input clk,
   input [3:0] d,
   output [3:0] q
   );

   reg [3:0] q;

   always @ (posedge clk) begin
        q = d;
   end

endmodule


module DFF_E (
   input clk,
   input en,
   input [3:0] d,
```

```
output [3:0] q_e
);

reg [3:0] q_e;

always @ (posedge clk) begin
    if (en)
        q_e = d;
end

endmodule
```

如范例中未使用该属性前，综合后会 infer 出对应的 GTP_DFF_E，如下图

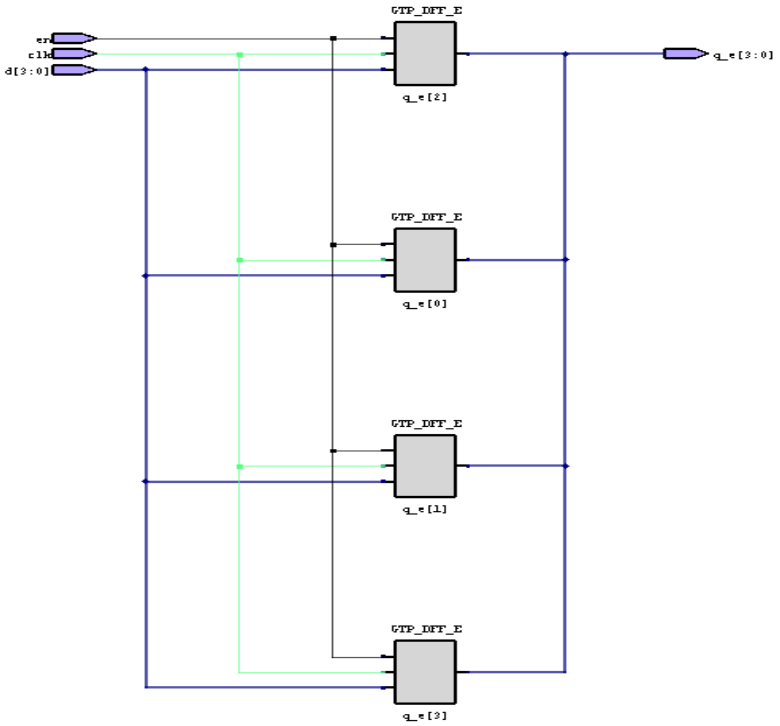

图 5-31 未应用 syn_reduce_controlset_size 属性

应用了 syn_reduce_controlset_size = 5 之后，综合后会将对应的 FF 的全部控制引脚移至 D 输入端，如下图

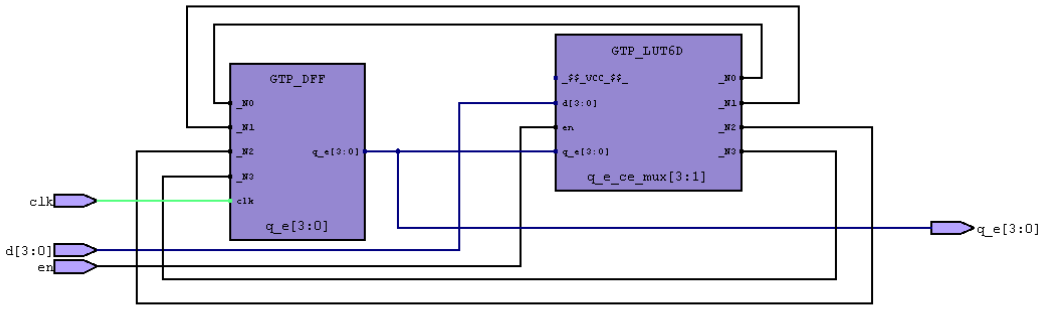

图 5-32 应用 syn_reduce_controlset_size 属性

5.18 syn_romstyle

用于指定 ROM 的实现方式。

syn_romstyle Values

Global	Object
Yes	module, rom-signal

属性值	作用
block_rom	指定 inferred ROM 可以 map 到 DRM 资源
lut_rom	指定 inferred ROM 可以 map 到 GTP_ROMxx 资源（实际也是占用 LUT 资源）
logic	指定 inferred ROM 可以 map 到普通逻辑资源

注：属性值 select_rom 与 lut_rom 等效

Syntax Specification

fdc file	define_global_attribute { syn_romstyle } {value} define_attribute {object} { syn_romstyle } { value }
verilog	object /* synthesis syn_romstyle ="value" */;

Verilog Example

```
module syn_romstyle_block_rom_reg (clk,addr,dout) ;
  input clk;
  input [4:0] addr;
  output [7:0] dout;
  reg [7:0] dout /* synthesis syn_romstyle = "block_rom" */ ;
  reg [4:0] addr_reg;

  always @(posedge clk)
  begin
      addr_reg<=addr;
      case (addr_reg)
          5'b00000: dout <= 8'b10000011;
          5'b00001: dout <= 8'b00000101;
```

5'b00010: dout <= 8'b00001001;

5'b00011: dout <= 8'b00001101;

5'b00100: dout <= 8'b00010001;

5'b00101: dout <= 8'b00011001;

5'b00110: dout <= 8'b00100001;

5'b00111: dout <= 8'b10110100;

5'b01000: dout <= 8'b11000000;

5'b01000: dout <= 8'b00011011;

5'b01001: dout <= 8'b10110001;

5'b01010: dout <= 8'b00110101;

5'b01011: dout <= 8'b01110010;

5'b01100: dout <= 8'b11100011;

5'b01101: dout <= 8'b00111111;

5'b01110: dout <= 8'b01010101;

5'b01111: dout <= 8'b00110100;

5'b10000: dout <= 8'b10110000;

5'b10000: dout <= 8'b11111011;

5'b10001: dout <= 8'b00010001;

5'b10010: dout <= 8'b10110011;

5'b10011: dout <= 8'b00101011;

5'b10100: dout <= 8'b11101110;

5'b10101: dout <= 8'b01110111;

5'b10110: dout <= 8'b01110101;

5'b10111: dout <= 8'b01000011;

5'b11000: dout <= 8'b01011100;

5'b11000: dout <= 8'b11101011;

5'b11001: dout <= 8'b00010100;

```
            5'b11010: dout <= 8'b00110011;
            5'b11011: dout <= 8'b00100101;
            5'b11100: dout <= 8'b01001110;
            5'b11101: dout <= 8'b01110100;
            5'b11110: dout <= 8'b11100101;
            default: dout <= 8'b01111110;
        endcase
    end
endmodule
```

如范例中未使用该属性前，综合后 infer 出 ROM，如下图

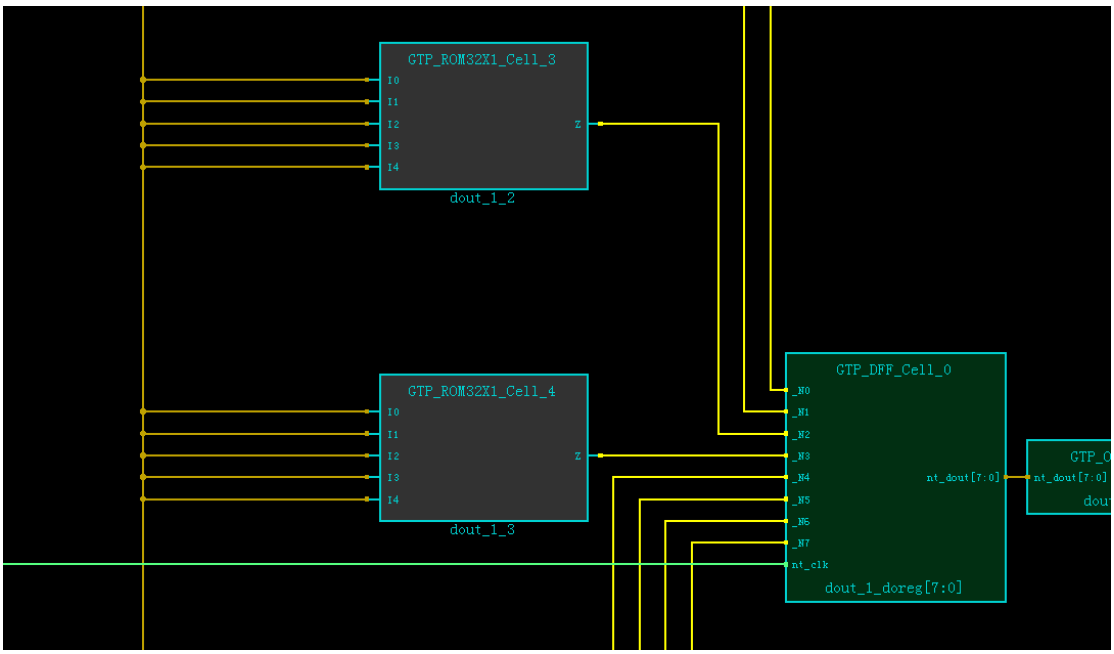

图 5-33 未应用 syn_romstyle 属性

应用了 syn_romstyle = "block_rom"之后，综合后可以 infer 出 DRM，如下图

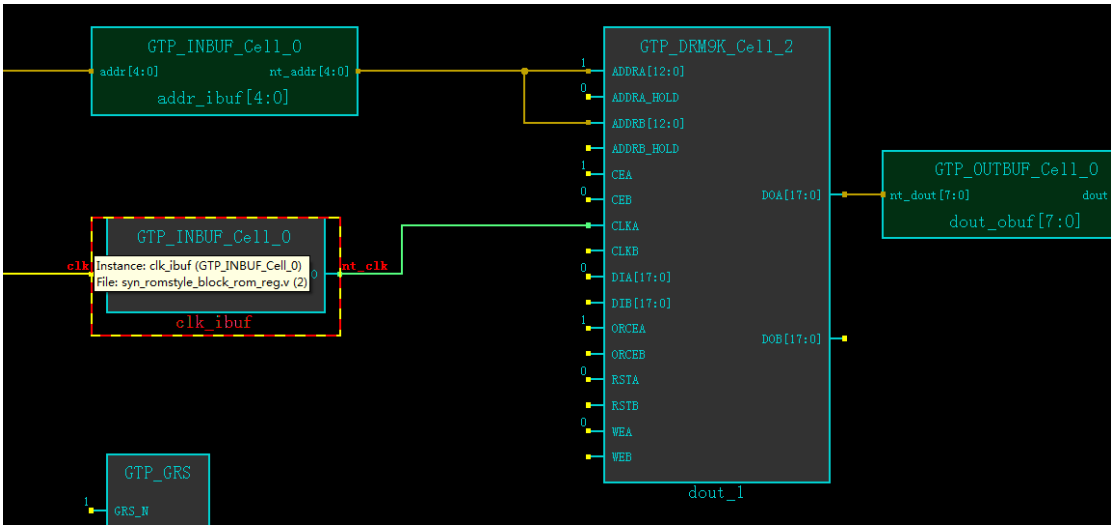

图 5-34 应用 syn_romstyle 属性

5.19 syn_srlstyle

该属性用于指定 inferred shift register 的实现方式。

syn_srlstyle Values

Global	Object
Yes	module, instance, register

属性值	作用
block_ram	指定 infered shift register 可以 map 到 DRM 资源
lut_ram	指定 infered shift register 可以 map 到 Distributed RAM 资源
registers	指定 infered shift register 可以 map 到 Registers

注：属性值 select_ram 与 lut_ram 等效

Syntax Specification

fdc file	define_global_attribute { syn_srlstyle } {value} define_attribute {object} { syn_srlstyle } { value }
verilog	object /* synthesis syn_srlstyle ="value" */;

Verilog Example

module test(

```
input din,
input en,
output dout1,
input clk
);

reg [3:0] pool /* synthesis syn_srlstyle = "block_ram" */;

always @(posedge clk)
begin
    if (en)
        pool <= {pool[2:0], din};
end

assign dout1 = pool[3];

endmodule
```

如范例中未使用该属性前，综合后 infer 出 Registers，如下图

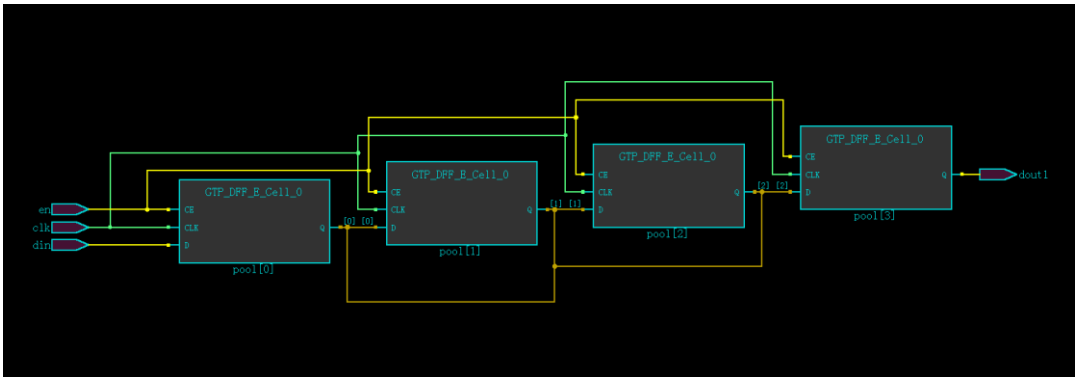

图 5-35 未应用 syn_srlstyle 属性

应用了 syn_srlstyle = "block_ram"之后，综合后可以 map 到 DRM，如下图

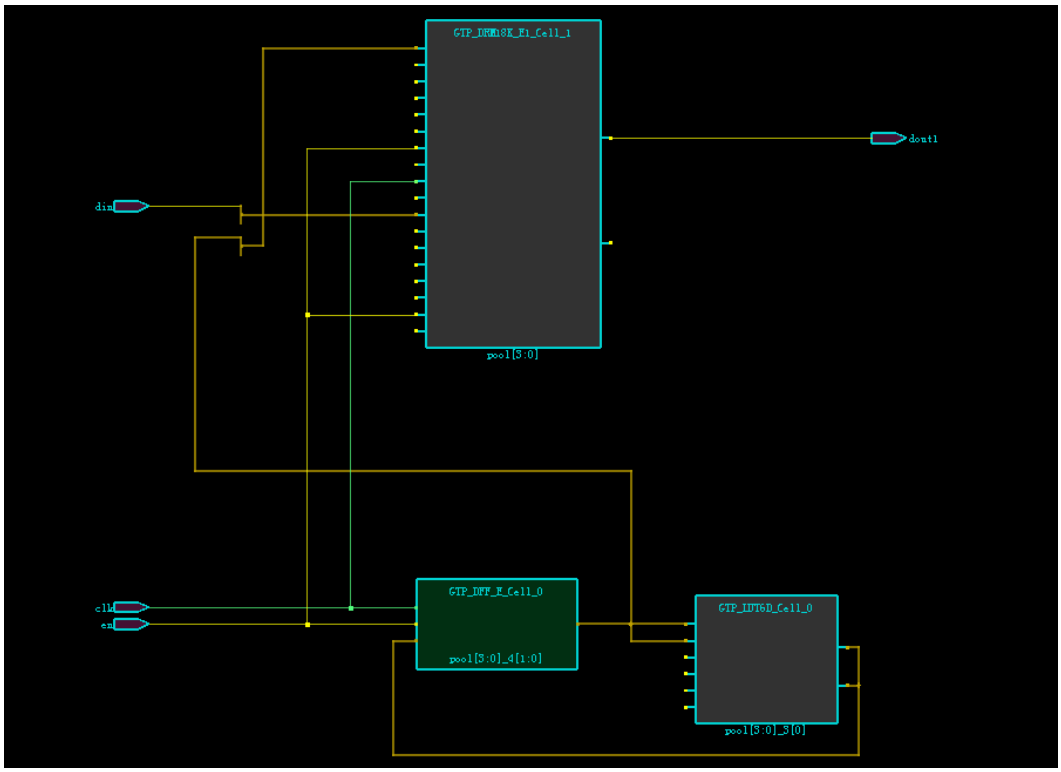

图 5-36 应用 syn_srlstyle 属性

5.20 syn_state_machine

用于指定寄存器是否 infer FSM

注意：1、当属性设置在 module 上时，仅支持设置在当前状态机实现代码的 module 上。

2、当寄存器上的 syn_state_machine 属性值未设时，由配置选项 fsm_compiler 决定是否 infer FSM.。当同时设置了 syn_preserve 时，syn_preserve 属性的优先级最高；syn_preserve=1，不会 infer fsm；syn_preserve=0，当寄存器的 syn_state_machine 属性值为 1 时(寄存器 上的属性值会覆盖 module 上的属性值）会 infer FSM; 当寄存器的 syn_state_machine 属性值为 0，不会 infer FSM。

syn_state_machine Values

Global	Object
No	module, register

属性值	作用
1	Compile 阶段推断出有限状态机
0	Compile 阶段不会推断出有限状态机

Syntax Specification

Verilog	object /* synthesis syn_state_machine = 1*/

Verilog Example

```
module test (clk,rst,data,out);

input clk,rst,data;

output out;

reg out;

reg [2:0] ps /* synthesis syn_state_machine = 0 */;

reg [2:0] ns;

parameter S0=3'b000, S1=3'b001, S2=3'b010, S3=3'b011, S4=3'b100,

 S5=3'b101, S6=3'b110;

always@(posedge clk or posedge rst)

begin

 if(rst)

   ps <= S0;

 else

   ps <= ns; end

always@(ps or data)

begin
```

```
ns = S0;
case (ps)
 S0: if(data)
 ns = S1;
 else ns = S0;
 S1: if(data)
 ns = S1;
 else ns = S2;
 S2: if(data)
 ns = S1;
 else ns = S3;
 S3: if(data)
 ns = S4;
 else ns = S0;
 S4: if(data)
 ns = S5;
 else ns = S2;
 S5: if(data)
 ns = S1;
 else ns = S6;
```

```
    S6: if(data)
    ns = S1;
    else ns = S3;
    default: ns = S0;
  endcase
end
always@(ps or data)
begin
  if((ps == S6) && (data == 1))
    out = 1;
  else
    out = 0;
end
endmodule
```

如范例中未应用该属性之前，默认是会 infer 出有限状态机，编码方式为 onehot，并在 run.log 中打出如下信息：

```
I: Encoding type of FSM 'ps_fsm[2:0]' is: onehot.
```

应用了 synthesis syn_state_machine = 0 后，则不会 infer 出有限状态机

5.21 syn_unconnected_inputs

该属性用于定义保持 instance 未连接的输入(unconnected inputs)在综合结果网表中保持悬空。

一般情况下，ADS 综合工具会保证 instance pin 有连接的 net，net 有 driver，但用户可以使用 syn_unconnected_inputs 属性来定义例外的 pin，如某 GTP 中的硬连线 pin 都设置此属性。

当设置到 module 上时，所有该 module 的 instance 未连接的 input 在综合结果网表中保持悬空。当设置到 instance 上时，该 instance 未连接的 input 在综合结果网表中保持悬空。

注意：unconnected input 指的是 input pin 未连接信号线，当 input pin 连接 net，但 net 没有 driver 时，综合结果网表中会将此 undriven net 优化为 GND.

syn_unconnected_inputs Values

Global	Object
No	module, instance

属性值	作用
@all	对于所有的未连接的输入在综合结果网表中将保持悬空
<PIN_NAME>	只有目标未连接的输入在综合结果网表中将保持悬空

Syntax Specification

fdc file	define_attribute {object} { syn_unconnected_inputs } { value }
verilog	object /* synthesis syn_unconnected_inputs ="value" */;

Verilog Example

```
module USER_I2C (
    output      SCL_OE_N,
    input       SCL_I,
    output      SCL_O,
    output      SDA_OE_N,
    input       SDA_I,
    output      SDA_O,
    output      IRQ
) /* synthesis syn_black_box syn_unconnected_inputs="SCL_I,SDA_I" */;

endmodule
```

```
module syn_unconnected_inputs_pins (
    output      SCL_OE_N,
    input       SCL_I,
    output      SCL_O,
    output      SDA_OE_N,
    input       SDA_I,
    output      SDA_O,
    output      IRQ
);

USER_I2C T_USER_I2C   (
    .SCL_OE_N(SCL_OE_N),
    .SCL_I(),
    .SCL_O(SCL_O),
    .SDA_OE_N(SDA_OE_N),
    .SDA_I(),
    .SDA_O(SDA_O),
    .IRQ(IRQ)
  );

endmodule
```

如范例中未使用该属性前，综合后的网表中 PIN "SCL_I"、PIN " SDA_I" 会接地，如下图

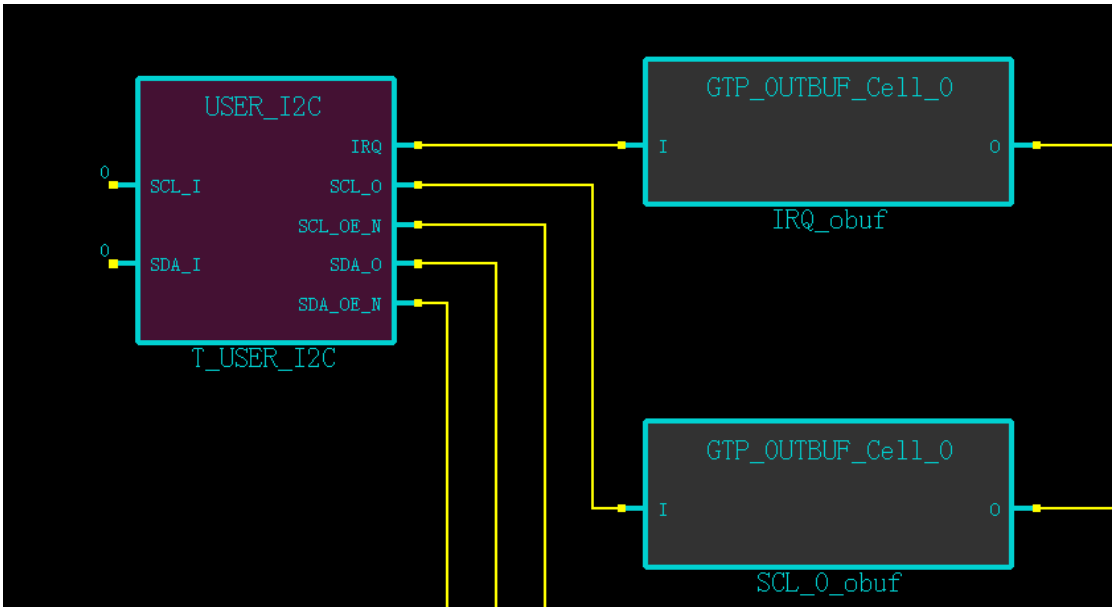

图 5-37 未应用 syn_unconnected_inputs 属性

应用了 syn_unconnected_inputs="SCL_I,SDA_I"之后，综合后网表中 PIN "SCL_I"、PIN " SDA_I"保持悬空，如下图

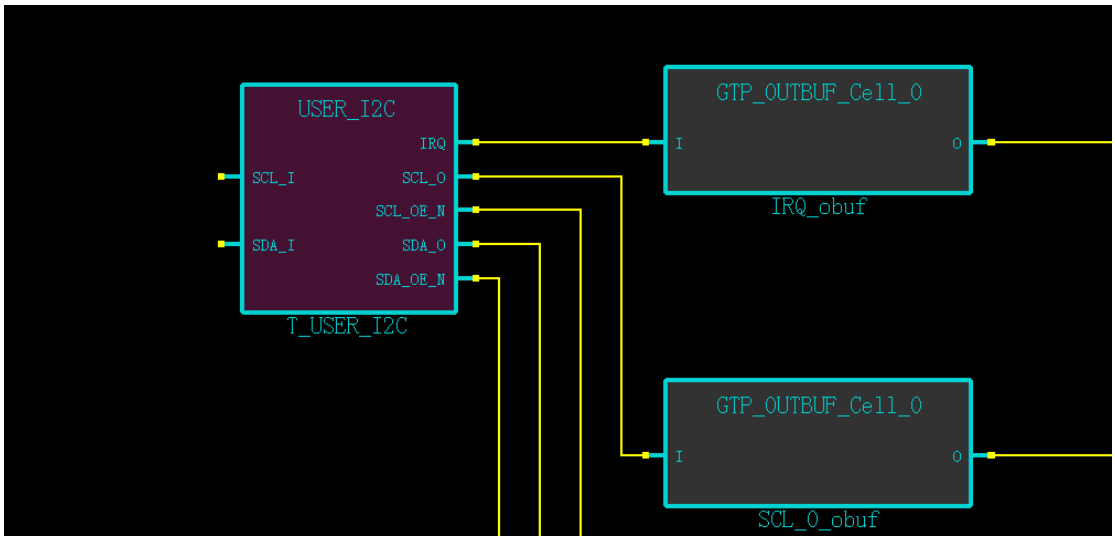

图 5-38 应用 syn_unconnected_inputs 属性

5.22 syn_useioff

该属性用于用于指定 FF 是否与 IO 进行 pack.既可以局部设置在特定的 port 和 register，也可以全局设置在 module.

对带有 syn_useioff =1 属性的 register 或 port ，综合后的 object 上会带有 PAP_IO_REGISTER 的属性，从而在 Device Map 阶段会将 IO 和 FF pack 在一起。当 syn_useioff =0 时，则不会对标记的 IO 和 REGISTER 进行 pack。当设置在 register 时，register 必须与 port 相连，该属性才能生效。

syn_useioff Values

Global	Object
Yes	port, register

属性值	作用
0	不会将 FF 与 IO 进行 pack
1	会将 FF 与 IO 进行 pack

Syntax Specification

fdc file	define_global_attribute {syn_useioff} {value} define_attribute {object} {syn_useioff} { value }
verilog	object /* synthesis syn_useioff = value */;

Verilog Example

```
module syn_useioff_1_reg (
    input clk,
    input d,
    output q
);

    test U (
        .clk(clk),
        .d(d),
        .q(q)
    );

endmodule
```

```
module test (
    input clk,
    input d,
    output q
);

    reg temp;
    reg qreg /* synthesis syn_useioff=1 */;

    assign q = qreg;

    always@(posedge clk) begin
        temp <= d;
        qreg <= temp;
    end

endmodule
```

如范例中未使用该属性前，Device Map 阶段不会将 q 和 qreg pack 到一起，如下图

图 5-39 未应用 syn_useioff 属性

应用了 syn_useioff=1 之后，Device Map 阶段会将 q 和 qreg pack 到一起，如下图

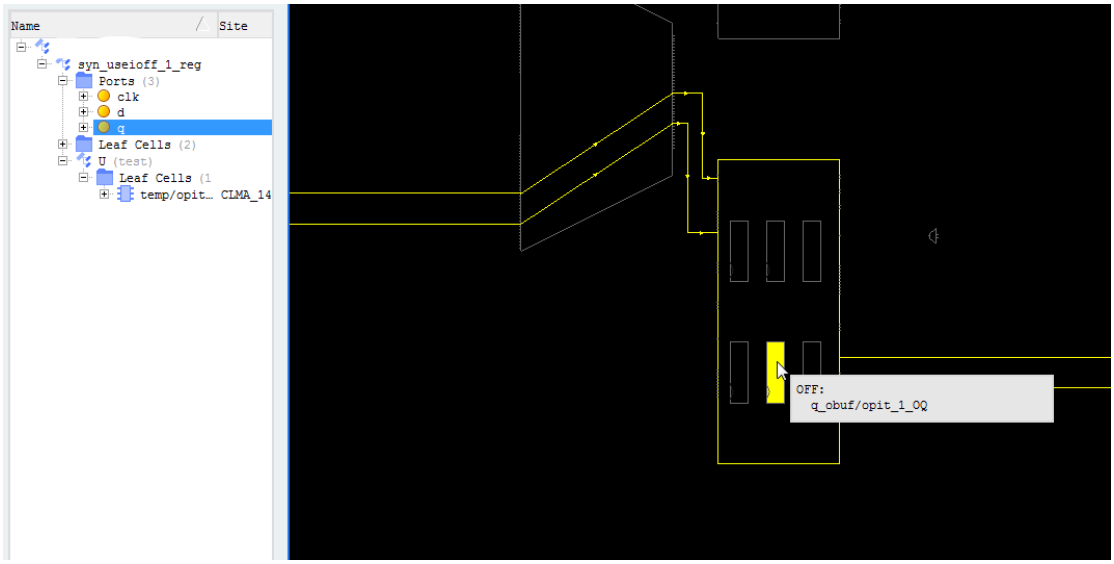

图 5-40 应用 syn_useioff 属性

5.23 full_case

表明在 case 语句中已经给出了所有可能的值，综合器不需要 infer 出锁存器来保留值。

full_case Values

Global	Object
No	A case,casex, or casez statement declaration

Syntax Specification

Verilog	object /* synthesis full_case */

Verilog Example

```
module test (out, a, b, c, d, select);
output out;
input a, b, c, d;
input [3:0] select;
reg out;

always @(select or a or b or c or d)
begin
casez (select) /* synthesis full_case */
4'b0001: out = a;
4'b001?: out = b;
4'b01??: out = c;
4'b1???: out = d;
endcase
end
endmodule
```

如范例中未使用该属性前，综合会 infer 出 latch，如下图

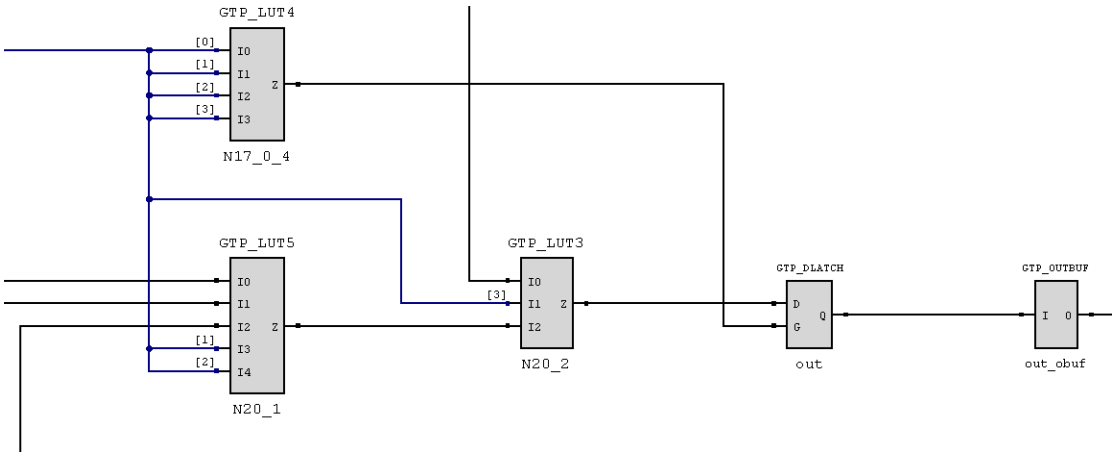

图 5-41 未应用 full_case 属性

应用了 full_case 属性之后，则不会 infer 出 latch。

注意：应用了 full_case 之后，可能导致 rtl 仿真和综合后仿真不一致。

5.24 parallel_case

仅适用于 Verilog 设计，强制使用并行多路复用结构，而不是优先级编码结构。

parallel_case Values

Global	Object
No	A case,casex, or casez statement declaration

Syntax Specification

Verilog	object /* synthesis parallel_case */

Verilog Example

```
module test (out, a, b, c, d, select);
output out;
input a, b, c, d;
input [3:0] select;
reg out;
```

```
always @(select or a or b or c or d)
begin
casez (select) /* synthesis parallel_case */
4'b???1: out = a;
4'b??1?: out = b;
4'b?1??: out = c;
4'b1???: out = d;
endcase
end
endmodule
```

如范例中未使用该属性前，是 infer 出优先级编码结构，如下图

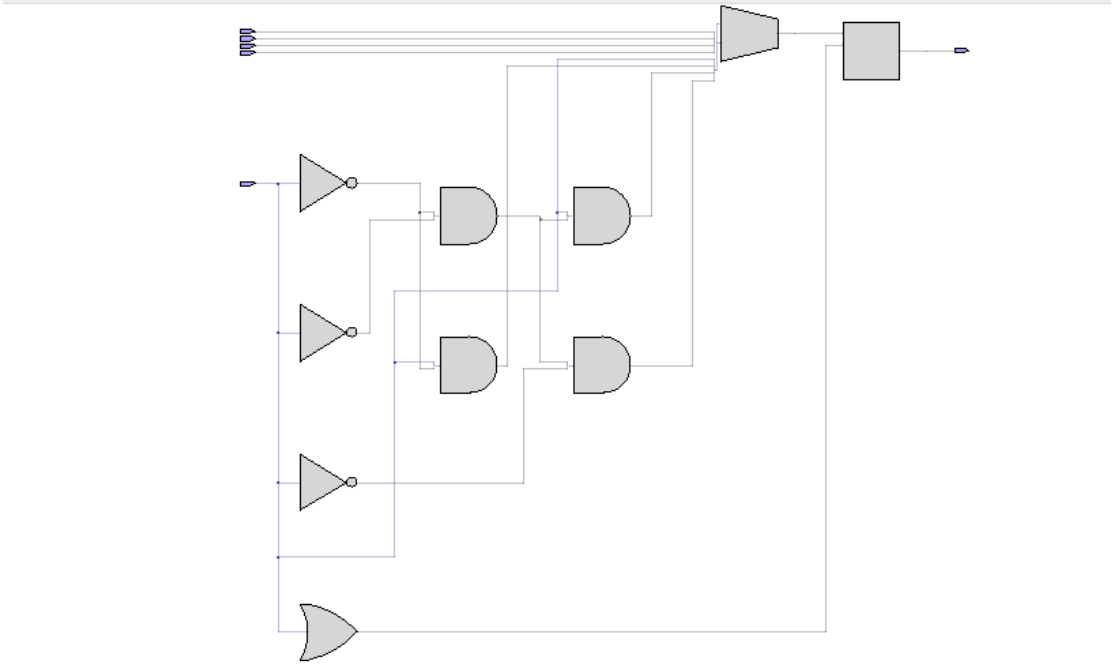

图 5-42 未应用 parallel_case 属性

应用了 parallel_case 属性之后，infer 出并行多路复用结构。

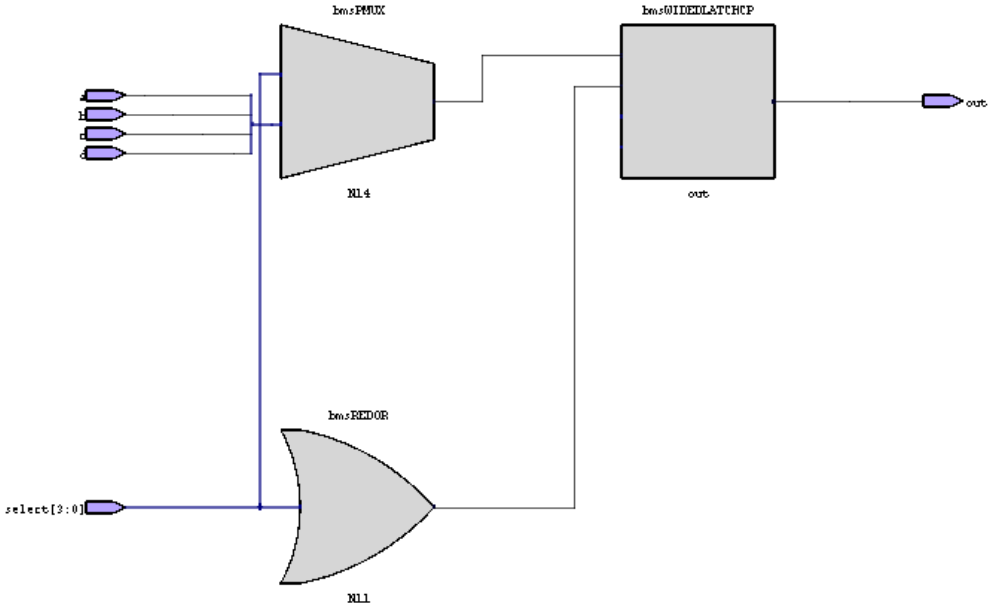

图 5-43 应用 parallel_case 属性

5.25 translate_off/ translate_on

用于指示编译器停止编译从/* synthesis translate_off */语句开始，/* synthesis translate_on */语句结束的 Verilog 描述。

Syntax Specification

Verilog	/* synthesis translate_off */
	/* synthesis translate_on */
	或
	// synthesis translate off
	// synthesis translate on

Verilog Example

```
module test(
input [5:0] a,b,
output [6:0] dout,
```

```
output [11:0] Nout);
assign dout = a + b;
/* synthesis translate_off */
assign Nout = a * b;
/* synthesis translate_on */
endmodule
```

如范例中未使用该属性前，会 infer 出乘法器逻辑，如下图

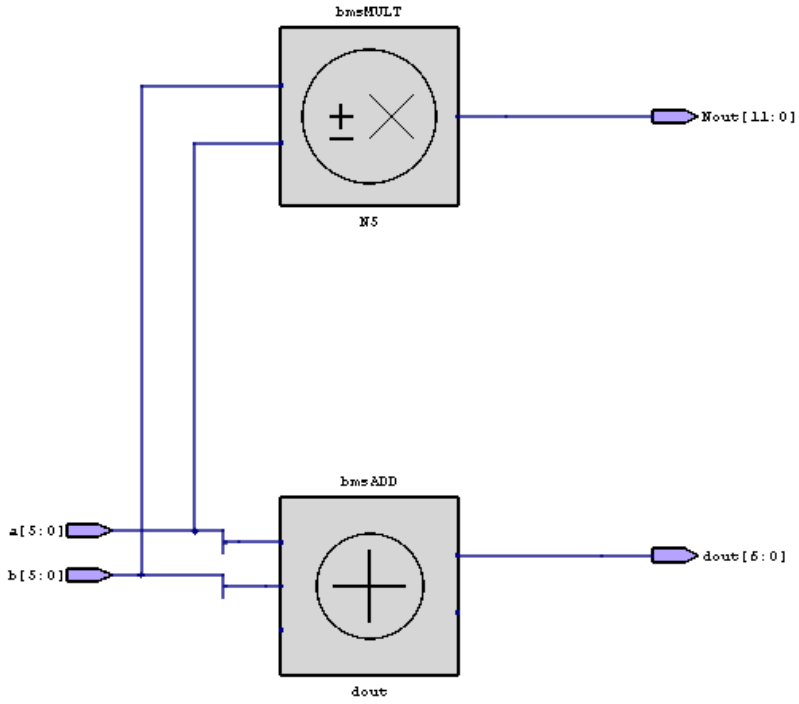

图 5-44 未应用 translate_off/on 属性

应用了 translate_off/translate_on 属性之后，不会 infer 出乘法器逻辑，Nout 信号悬空，如下图。

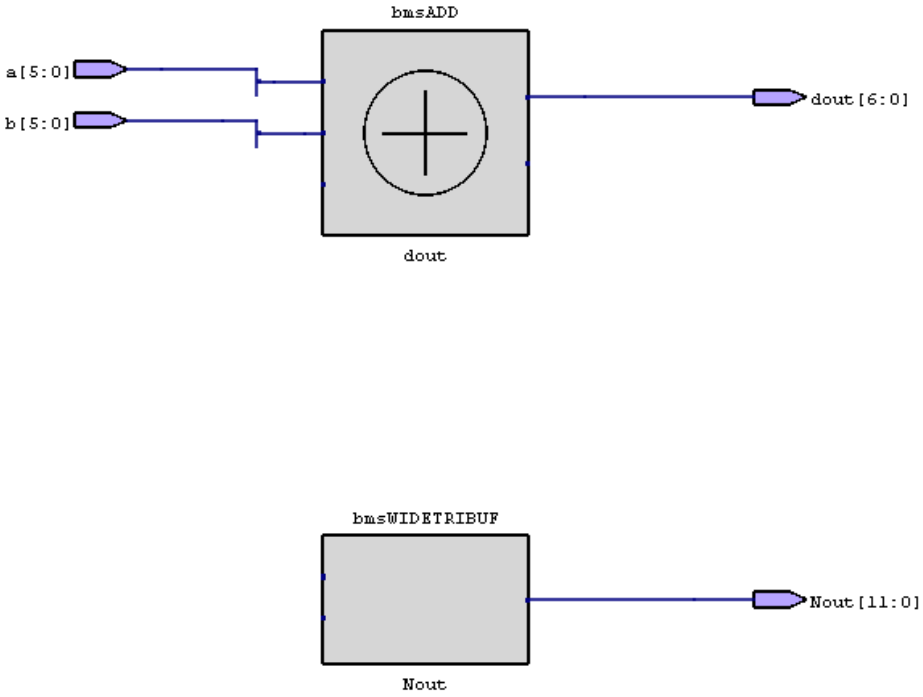

图 5-45 应用 translate_off/on 属性

6 Synplify Pro 到 ADS 切换指南

6. 1 如何选择 ADS 综合工具

新建 PDS Project 时，在“New Project Wizard”进行到选择“器件”型号时，在下方的“Synthesis Tool:”下拉菜单中选择“ADS”

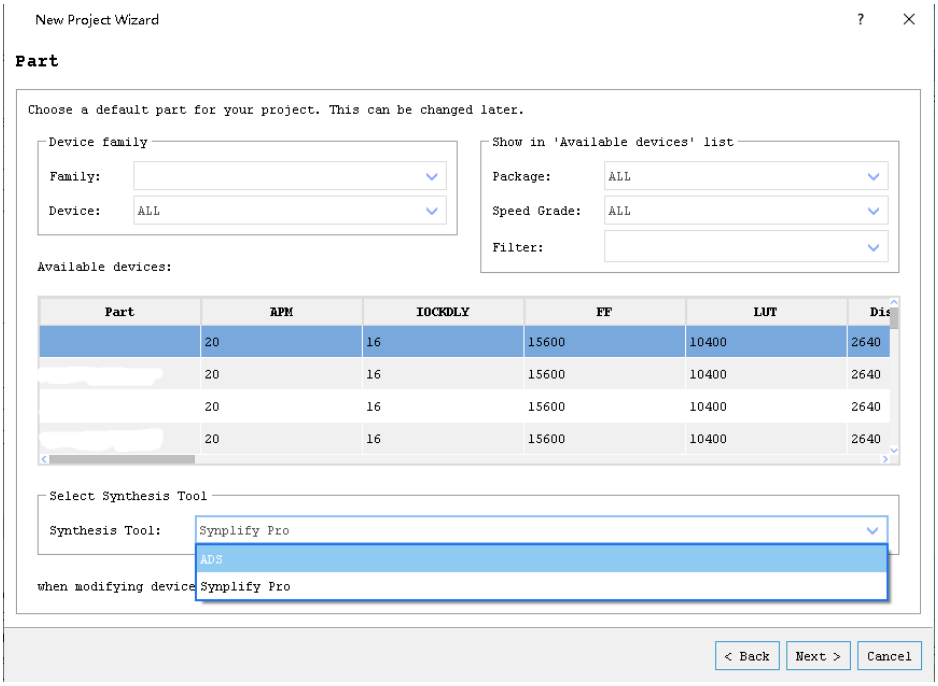

图 6-1 综合工具选择

将现有的 PDS 工程切换为“ADS”的方法：在“Navigator”下的器件名称上“右击”选择“Project Settings”，或者在“Flow”下的任意动作中“右击”选择“Project Settings”，然后选择左边的“Part”页面，在 Part 页面下方的“Synthesis Tool:”下拉菜单中选择“ADS”

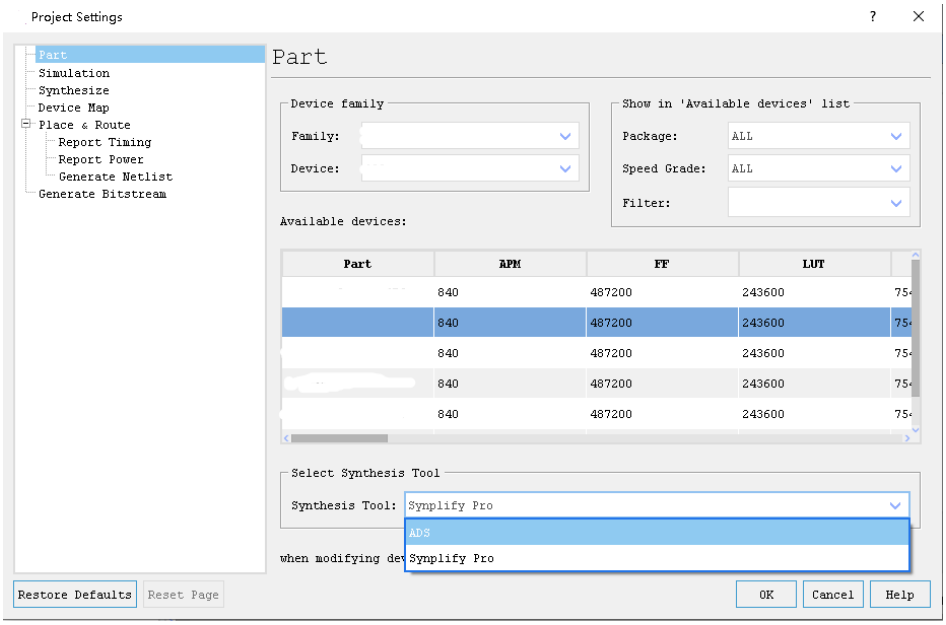

图 6-2 project setting 中综合工具选择

6. 2 综合 Flow 差异

Synplify Pro 作为第三方综合工具，在 PDS Flow 中只有 Synthesize 一个子流程，而 ADS 综合在 PDS Flow 主流程中有 Compile 和 Synthesize 两个子流程。

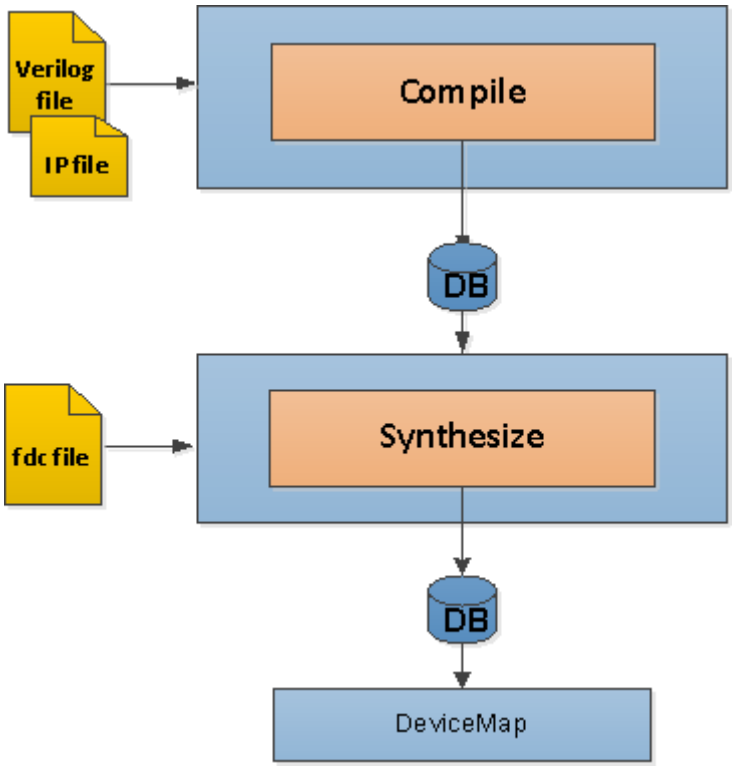

图 6-3 ADS 综合流程

ADS 综合分成 Compile 和 Synthesize 两个子流程，方便用户感知，fdc 约束文件是在 Synthesize 阶段读入，作用于 Compile 后生成的 RTL netlist 中的。

注意：不管是 Synplify Pro 还是 ADS，综合 fdc (FPGA Design Constraint) 约束文件都是**作用于 Compile 后的 RTL netlist 上**的。

6. 3 支持的 HDL 语言差异

Synplify Pro for Pango 是委托 Synopsys 开发的，成熟的 FPGA 综合工具，支持的 HDL 语言有 Verilog-2001, SystemVerilog 可综合子集，VHDL-1993, 部分 VHDL-2008 和 VHDL-2019 语言可综合子集。

ADS 是紫光同创自研的，开发中的 FPGA 综合工具。当前支持 Verilog-2001 可综合子集，及有限的 SystemVerilog 语法。

当前支持的 SystemVerilog 语法：**二维数组形式的 input/output 端口声明，$clog2, $signed, $unsigned 系统函数**。

PDS_2019.4 版本中，ADS 默认支持的语言为 Verilog-2001，当使用了 SystemVerilog 语法的工程，需要**手动打开 SystemVerilog 支持**。

“右击” Compile, 选择 “Project Settings”，将其中的 SystemVerilog 打勾。

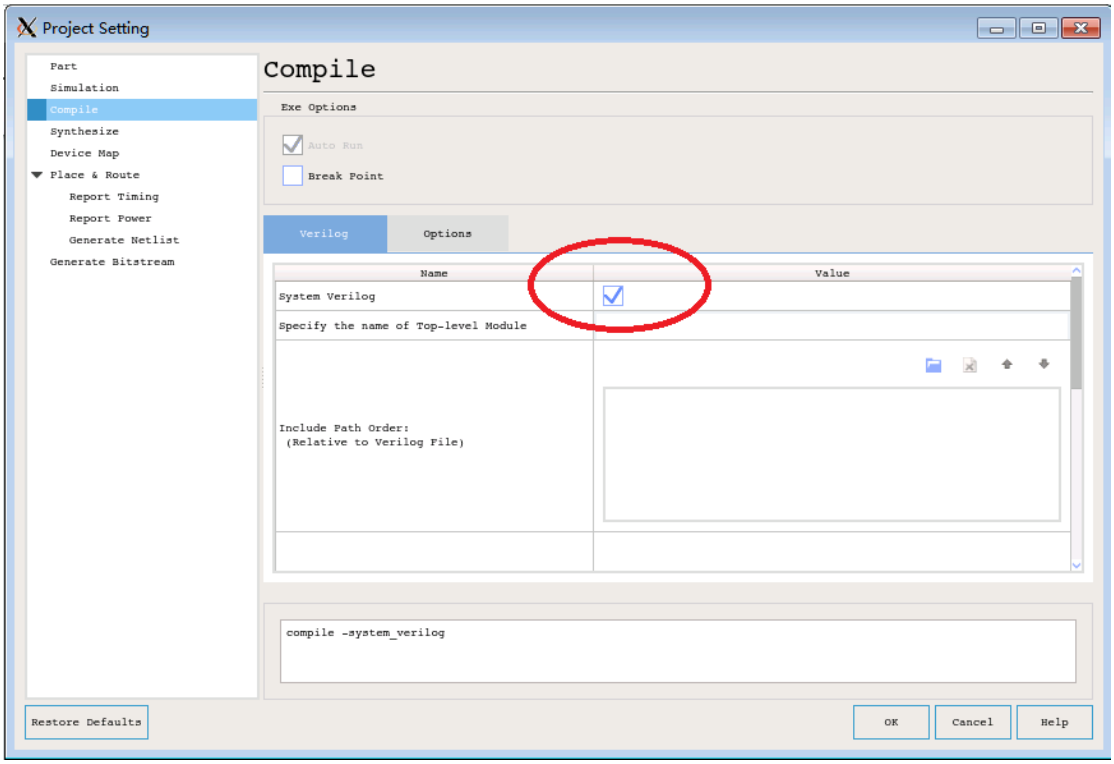

图 6-4 SystemVerilog 支持

6. 4 可能的 FDC 约束文件修改

如第 2 节“综合 Flow 差异”中描述的，FDC 约束文件是作用于 Compile 后的 RTL netlist 上的，而 Synplify Pro 和 ADS 做为两个独立开发的综合工具，生成的 Compile 后 RTL netlist 会存在差异，导致 FDC 约束文件被约束的 design object (如 instance 名称) 找不到。这时，需要手动修改 FDC 约束文件来适配 ADS 综合工具。

ADS 提供了 FDC 约束检查工具，右击 “FDC 约束文件”，弹出菜单中选择 “**Constraint Check**” 会报告出约束不成功的 FDC 命令，并生成约束检查报告 (.ccr)。

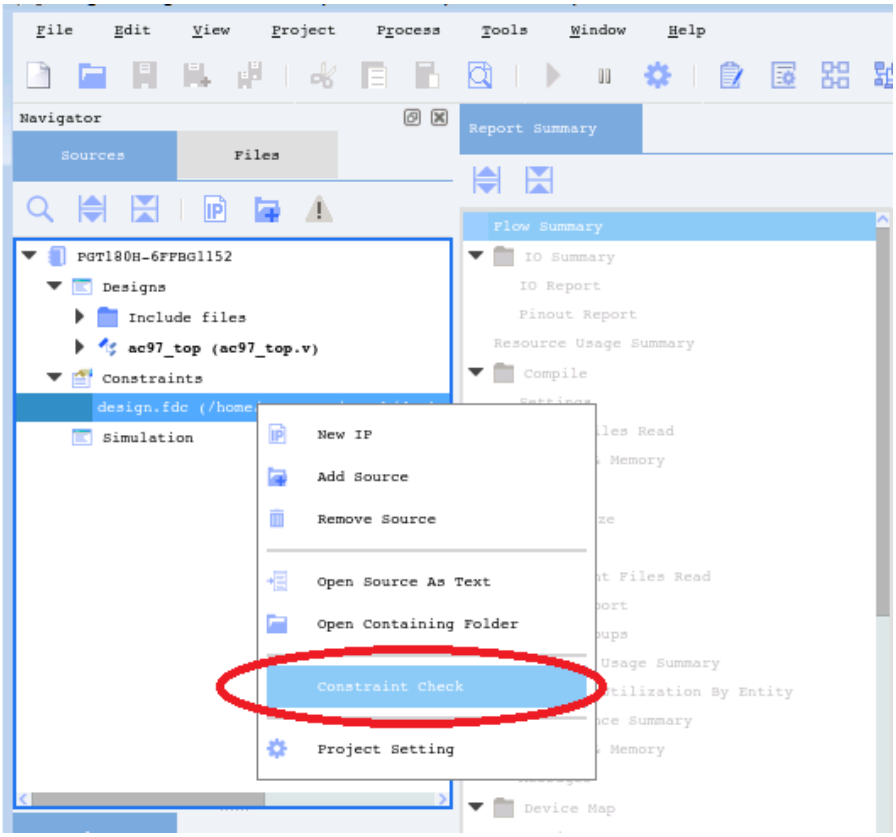

图 6-5 Constraint Check 约束检查

可以使用“RTL Schematic”工具，辅助定位要被约束的 design object 的名称，打开“RTL Schematic”有两种方法。

一是在“Tools”下的“Schemativ View”子菜单中选择“View RTL Schematic”，如下图所示。

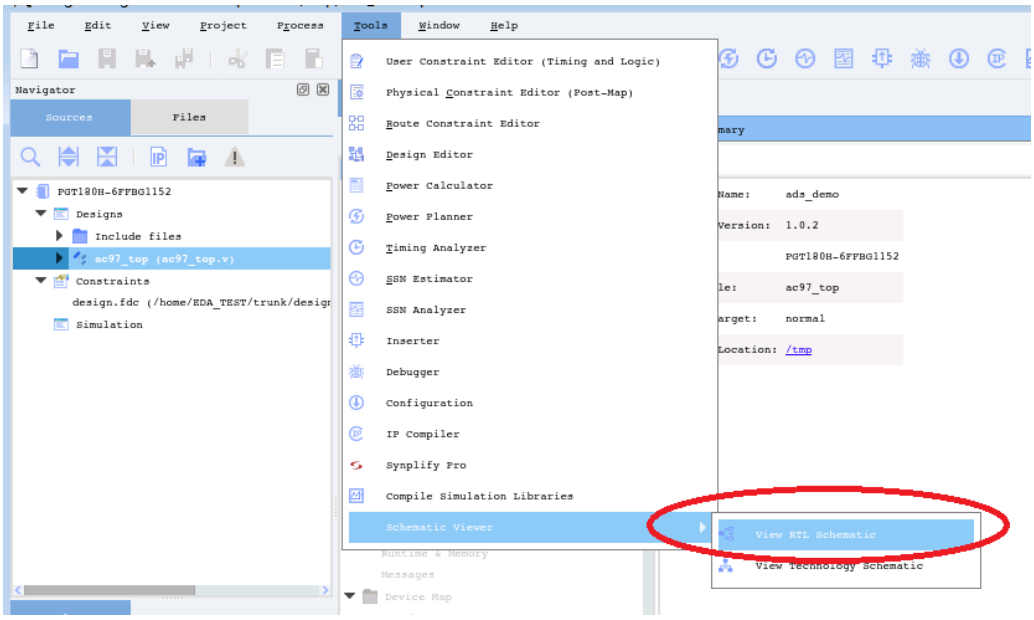

图 6-6 RTL Schematic 工具

二是在工具栏图标中，单击如下红框中的图标。

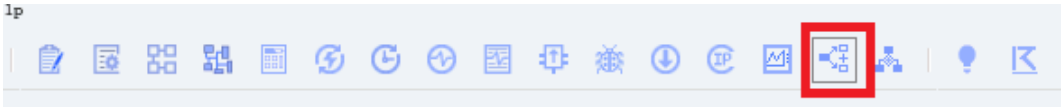

图 6-7 工具栏中选择 RTL Schematic

打开“RTL Schematic”后，可以通过“Design Browser”和 搜索功能查找需要被约束的“design object”。

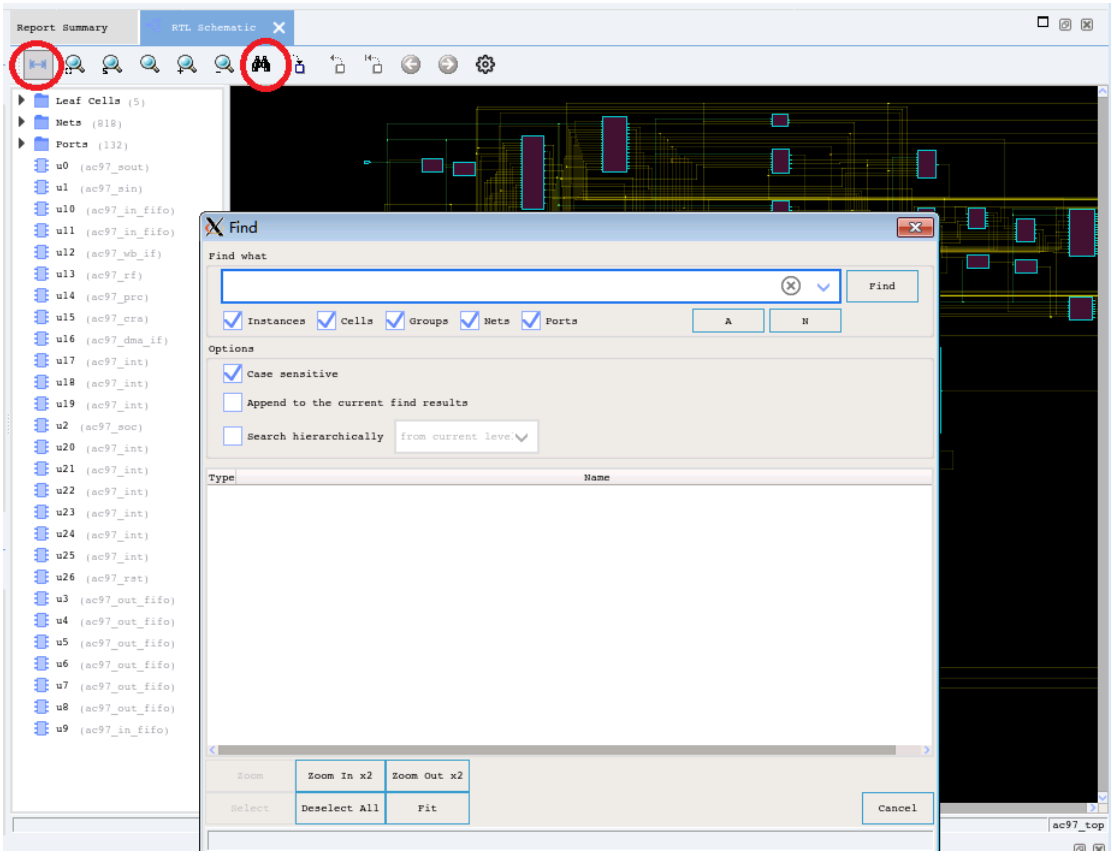

图 6-8 Design Browser 功能

6. 5 注意事项

6.5.1 错误 E: Array port <xx> is not supported. Enable SystemVerlog features first.

二维数组 port 声明是 SystemVerilog 语法，需要先打开 SystemVerilog

说明：Synplify Pro 可能是随着 SystemVerilog 的开发，混淆了 Verilog-2001 和 SystemVerilog 语法的边界，ADS 则严格按标准来区别。

6.5.2 错误 E: Verilog-4088: Index <xx> of <xx> is out of range

ADS 严格检查了 vector 的 bit-select, part-select, 当常量的 bit-select 或者 part-select 超出了变量定义的 range 时，会报错。

Synplify Pro 没有严格检查“out of range”的报错，有的情况下会报错，有的情况下则不报错。

Synplify Pro 为: CS101:@E: index <xx> is out of range for variable <xx>

6.5.3 错误 E: DRC-0006: The pin <pin-name> of inst <inst-name> is floating

ADS 的设计会检查，每个 input pin 上是否连接着 net，每条 net 是不是有 driver。不符合规则的，会报告此错误消息。

对于明确需要保持 floating 的 pin，需要显示使用 syn_unconnected_inputs 进行说明。

6.5.4 Unnamed generate block 名称

根据 Verilog-2001 标准，若 generate block 没有显示指定名称，为了能访问 generate block 中的 design object，规定按同一层次 generate block 中的出现顺序，自动命名为 genblk<n> (其中 n 为同一层中出现的第几个 generate)

以下内容摘抄自 IEEE 1364-2005 Verilog HDL Reference Manual

12.4.3 External names for unnamed generate blocks

Although an unnamed generate block has no name that can be used in a hierarchical name, it needs to have a name by which external interfaces can refer to it.

Each generate construct in a given scope is assigned a number. The number will be 1 for the construct that appears textually first in that scope and will increase by 1 for each subsequent generate construct in that scope. All unnamed generate blocks will be given the name "genblk<n>"

图 6-9 Unnamed generate block 命名标准

免责声明

版权声明

免责声明

1、本文档仅提供阶段性信息，所含内容可根据产品的实际情况随时更新，恕不另行通知。如因本文档使用不当造成的直接或间接损失，本公司不承担任何法律责任。

2、本文档按现状提供，不负任何担保责任，包括对适销性、适用于特定用途或非侵权性的任何担保，和任何提案、规格或样品在他处提到的任何担保。本文档在此未以禁止反言或其他方式授予任何知识产权使用许可，不管是明示许可还是暗示许可。

3、公司保留任何时候在不事先声明的情况下对公司系列产品相关文档的修改权利。