

Tang Dynasty（TD）

软件手册

版本号: 4.4

上海安路信息科技有限公司

2019.02



安路公司发布此用户手册文档仅限于基于安路 FPGA/CPLD 器件的 TD 软件用户，其他个人未经安路公司书面同意不得以任何形式进行复制，分发，再版，下载，展示，其中的任何手段与形式包括但不限于：电子，机械，复印，录音等。安路公司不对任何人对此文档的使用承担任何责任。安路公司保留在任意时间更改此文档的权利，文档更改恕不另行通知。安路公司对于文档中错误的更正以及文档更正后的更新不承担任何义务。安路公司在技术支持和可能提供的信息援助中不承担任何责任。

目录

1 启动软件 7

软件要求 7

硬件要求 7

安装与卸载 TD 7

启动 TD 软件 8

获得帮助 8

2 项目管理 9

2.1 创建新项目 9

2.2 打开项目 12

2.3 转换工程 13

2.4 导出 tcl 脚本 16

2.5 源文件管理 18

2.5.1. 新建文件 18

2.5.2 创建 VHDL Package 19

2.5.3 添加和移除文件 22

2.5.4 编辑文件 23

3 IP 生成器 27

3.1 COMMON 模块 27

3.1.1 BUFG 模块 27

3.1.2 IDDR 模块 31

3.1.3 ODDR 模块 35

3.2 PLL 模块 39
3.2.1 创建 PLL 模块 39
3.2.2 例化 PLL 模块 47
3.3 DSP 模块 48
3.3.1 创建 DSP 模块 48
3.3.2 例化 DSP 模块 52
3.4 Divider 模块 53
3.4.1 创建 Divider 模块 53
3.4.2 例化 Divider 模块 56
3.5 BRAM 模块 57
3.5.1 创建 BRAM 模块 57
3.5.2 例化 BRAM 模块 70
3.6 FIFO 模块 71
3.6.1 创建 FIFO 模块 71
3.6.2 例化 FIFO 模块 75
3.7 RAMFIFO 76
3.7.1 创建 RAMFIFO 模块 76
3.7.2 例化 RAMFIFO 模块 80
3.8 DRAM 模块 81
3.8.1 创建 DRAM 模块 81
3.8.2 例化 DRAM 模块 84
3.9 SDRAM 模块 85

3.9.1 创建 SDRAM 模块 85
3.9.2 例化 SDRAM 模块 87
3.10 ADC 模块 89
3.10.1 创建 ADC 模块 89
3.10.2 例化 ADC 模块 92
4 用户约束 93
4.1 物理约束 93
4.1.1 添加 IO 约束 93
4.1.2 界面设置 IO 约束 96
4.2 时序约束 103
4.2.1 添加时序约束 103
4.2.2 界面设置时序约束 105
5 HDL2Bit 流程 123
5.1 读入文件 124
5.2 RTL 级优化 125
5.2.1 Synthesis Keep 127
5.2.2 Synthesis Directive 129
5.3 门级优化 133
5.4 布局优化 135
5.5 布线优化 136
5.6 生成位流文件 138
5.7 syn_ip_flow 141
5.8 Design Summary 145

5.8.1 RTL Summary 145
5.8.2 Gate Summary 147
5.8.3 Physical Summary 149
5.8.4 Timing Summary 150
5.8.5 Clock Intersection Summary 154

6 功能仿真 159

7 下载 163

7.1 下载流程简介 163

7.2 位流文件类型 166

7.3 下载模式 167

7.3.1 Dual Boot 169
7.3.2 Multi Boot 172

7.4 扩展功能 174

7.4.1 Create Flash File 174
7.4.2 Update BRAM Data 177
7.4.3 EF2 Encrypt 180

7.5 离线下载器 183

7.5.1 离线下载器的介绍 183

7.5.2 离线下载器的使用步骤 185

7.6 Device Chain 188

8 工具集 194

8.1 Schematic Viewer 194

8.2 Chip Viewer 200

8.3 ChipWatcher 207

8.4 BramEditor 222

8.5 ChipProbe 227

9 附录 230

9.1 ADC 约束说明 230

9.2 SDC 约束说明 232

9.3 ModelSim 仿真流程 239

9.3.1 添加仿真库 239

9.3.2 仿真 241

9.4 USB 下载驱动安装 247

9.5 安全声明 251

1 启动软件

软件要求

用户需要安装下面的软件以便使用此指南:

- TD

在 Linux 下 TD 运行的操作系统要求:

- Red Hat Enterprise 6.0 及以上版本

在 Windows 下 TD 运行的操作系统要求:

- Windows 7 sp1 及以上版本

硬件要求

用户的计算机硬件需要以下配置:

- 处理器：1GHz 以上
- 内存：500M 以上
- 硬盘：100M 以上剩余空间

安装与卸载 TD

安装 TD,请双击 TD 安装盘中的 msi 文件，然后遵照安装步骤完成安装

在安装的过程中，若提醒安装 vc_redist.exe，请根据提示进行安装，安装后会继续安装 TD 软件，直至安装完成。若无法成功安装 vc_redist.exe，请更新 Windows 补丁。

卸载 TD,请点击 开始→控制面板→选择 TD→卸载

启动 TD 软件

在 Windows 下 TD 运行的操作系统要求：

- Windows7 及以上版本

Start → All Programs → TD → td.exe

在 Linux 下启动 TD：

- 界面模式：/td_install_dir/td -gui
- 命令模式：/td_install_dir/td

在 Windows 下启动 TD CMD 窗口：

Start → All Programs → td_commands_prompt

获得帮助

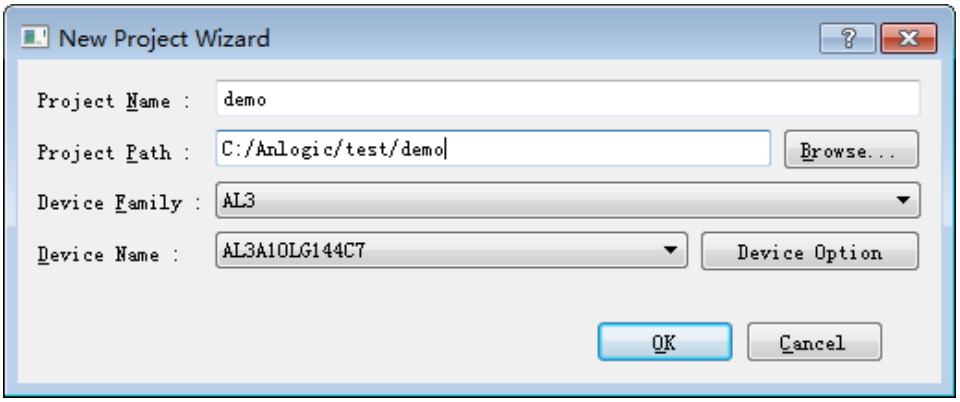
用户可发邮件至 support@anlogic.com 获得关于 TD 软件和相关工具的帮助。

2 项目管理

2.1 创建新项目

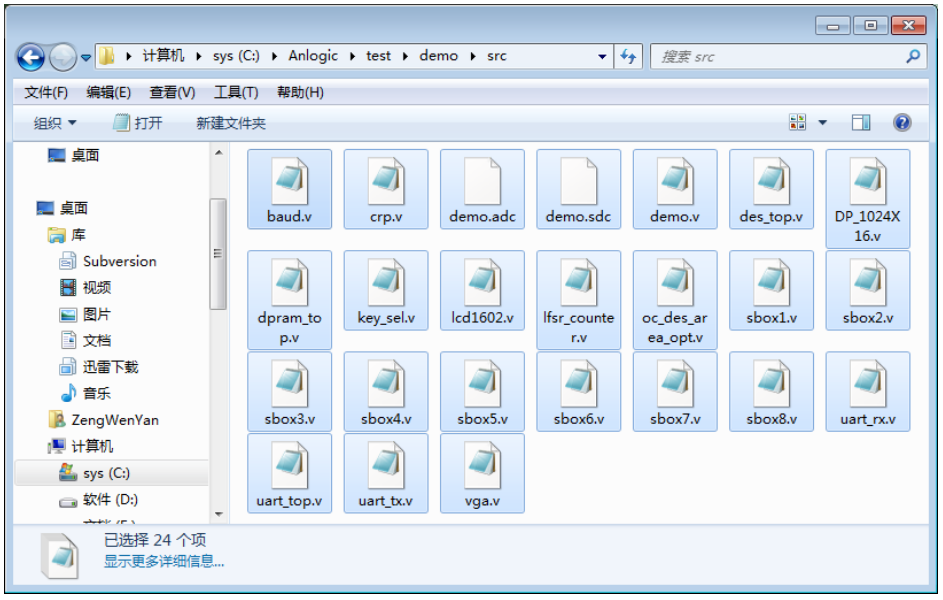
创建新项目:

1. 选择 **Project → New Project...** 此时会弹出新项目对话框
2. 指定所创建项目的存储路径并输入项目名称
3. 选择 **Device Family** 和 **Device Name**，默认为 AL3A10LG144C7。

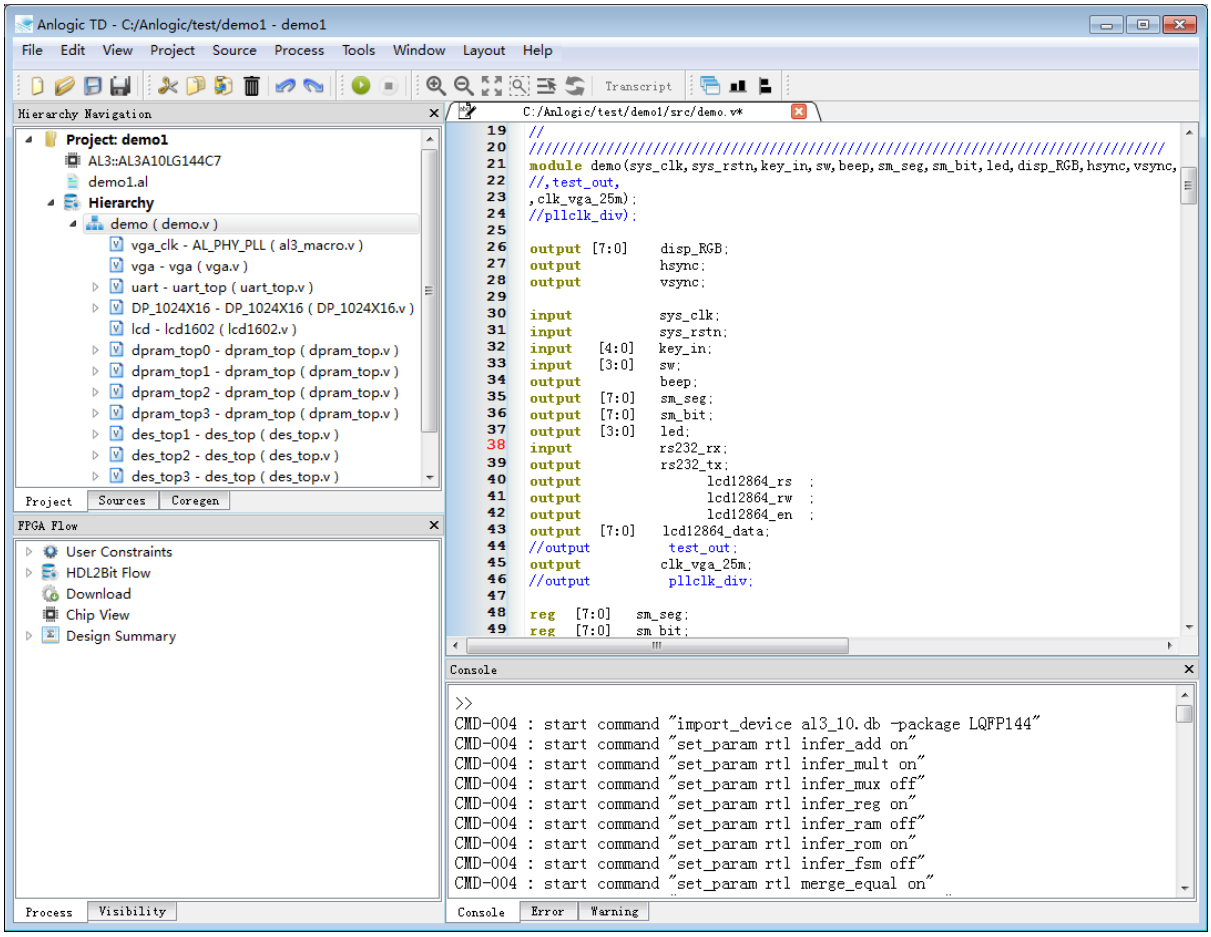


添加源文件:

1. 选择 **Source → Add Source...**
2. 选择需要添加的 HDL 源文件，点击打开。

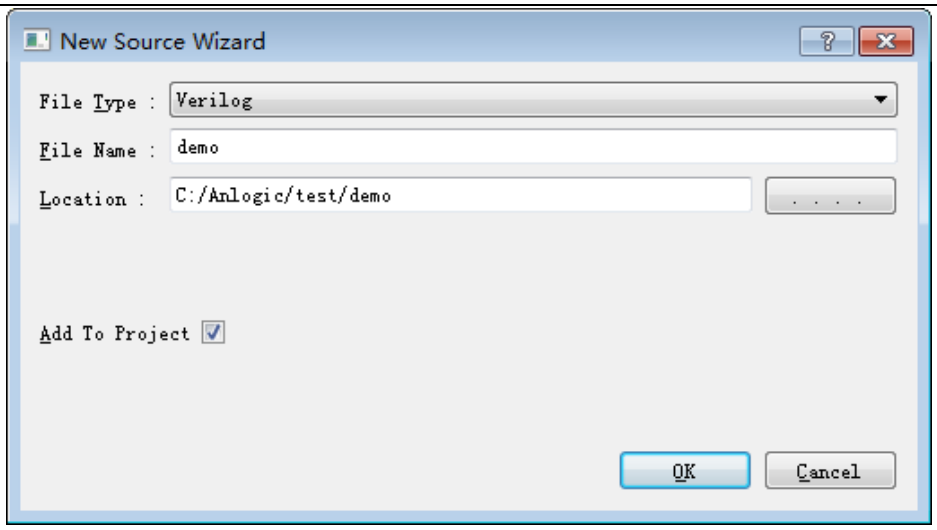


3. 此时，在 Hierarchy 中可看到添加的所有源文件，并可通过双击打开源文件。

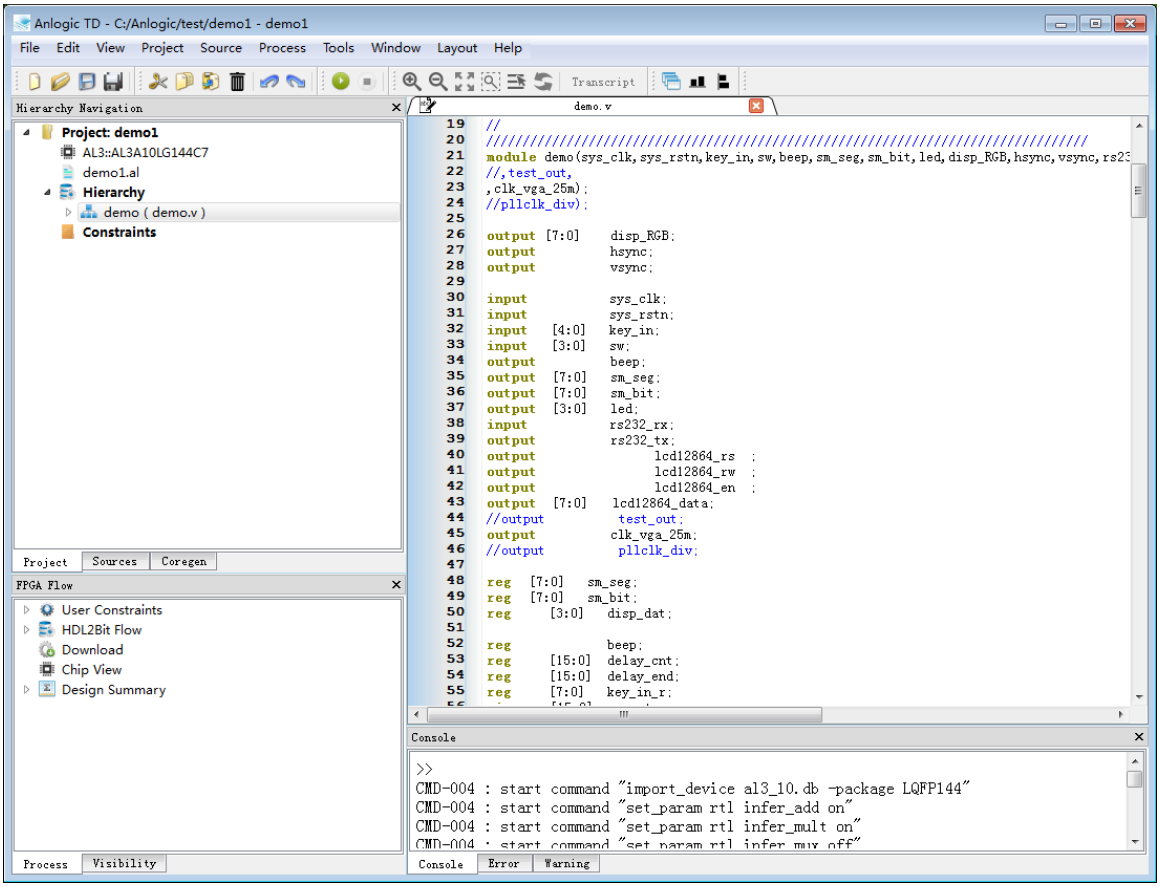


创建源文件:

1. 选择 **Source → New Source...**
2. 源文件类型默认为 Verilog .
3. 输入 **File Name**.
4. 确定已经勾选 **Add To Project**.
5. 点击 **OK**，完成创建



6. 输入文件内容，选择 **File → Save** 保存文件

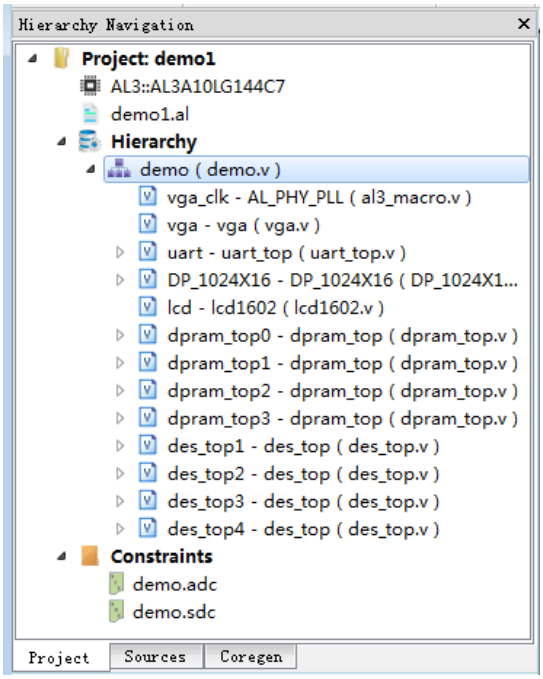


设置顶层模块

人工设置顶层模块是可选项。如果没有设置顶层模块，TD 软件将自动分析模块的层次结构选择最顶层模块。

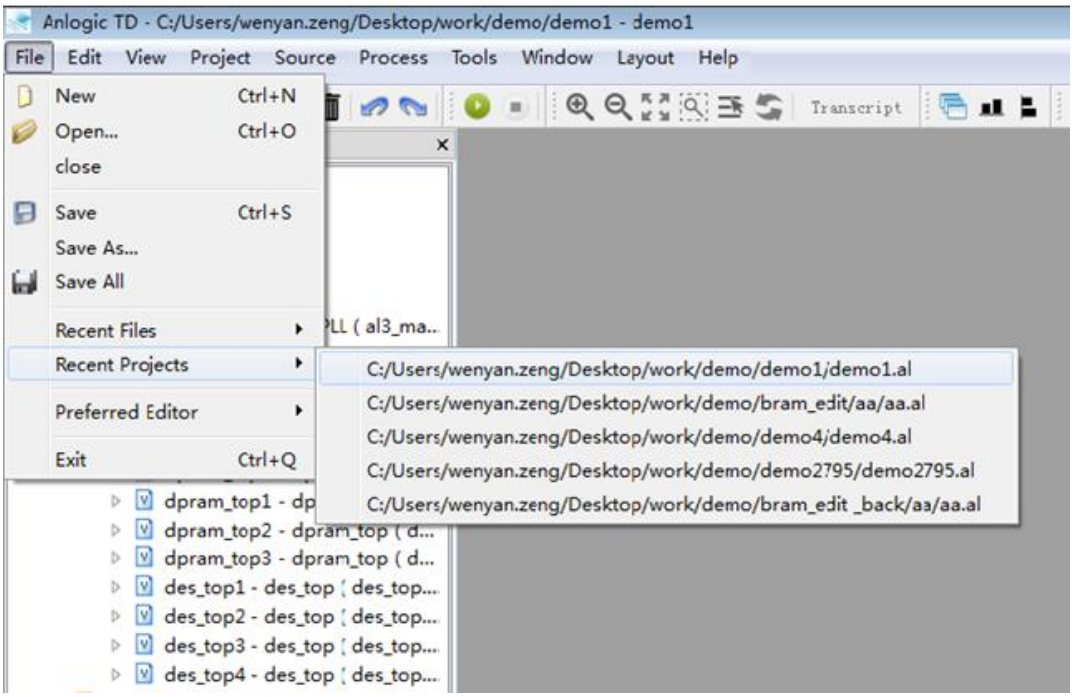
在 Hierarchy Navigation 窗口中，右键单击目标模块所在行，选择 **Set As Top**，在

目标模块前将会出现紫色的顶层模块标记。

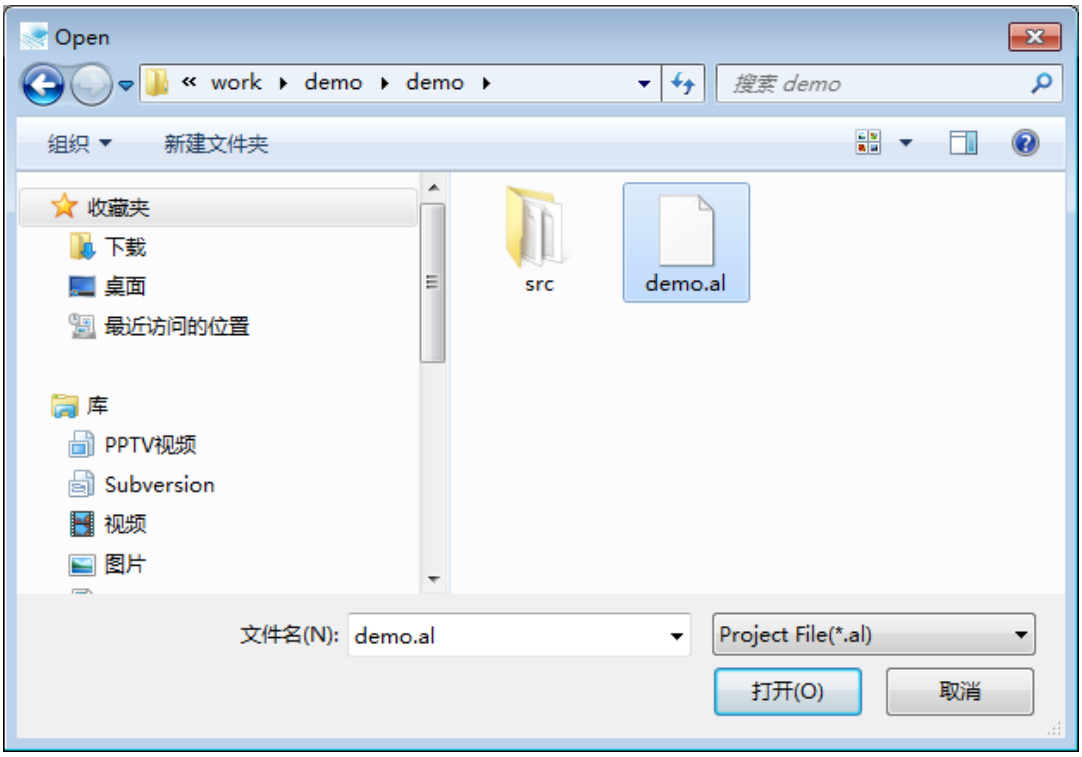


2.2 打开项目

TD 会根据项目打开的先后顺序为用户保留已打开过的项目和文件，用户可通过 **File → Recent Projects** 和 **File → Recent Files** 打开曾经打开过的项目或文件。



用户还可通过 **Project → Open Project** 选择 .al 文件来打开一个已存在的项目。

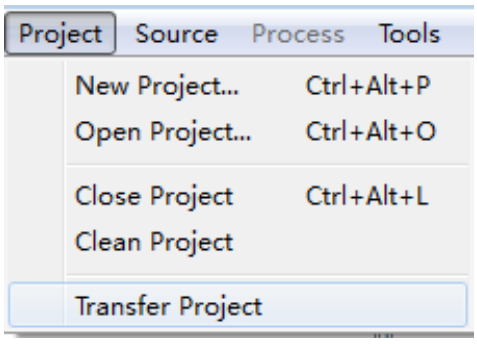


2.3 转换工程

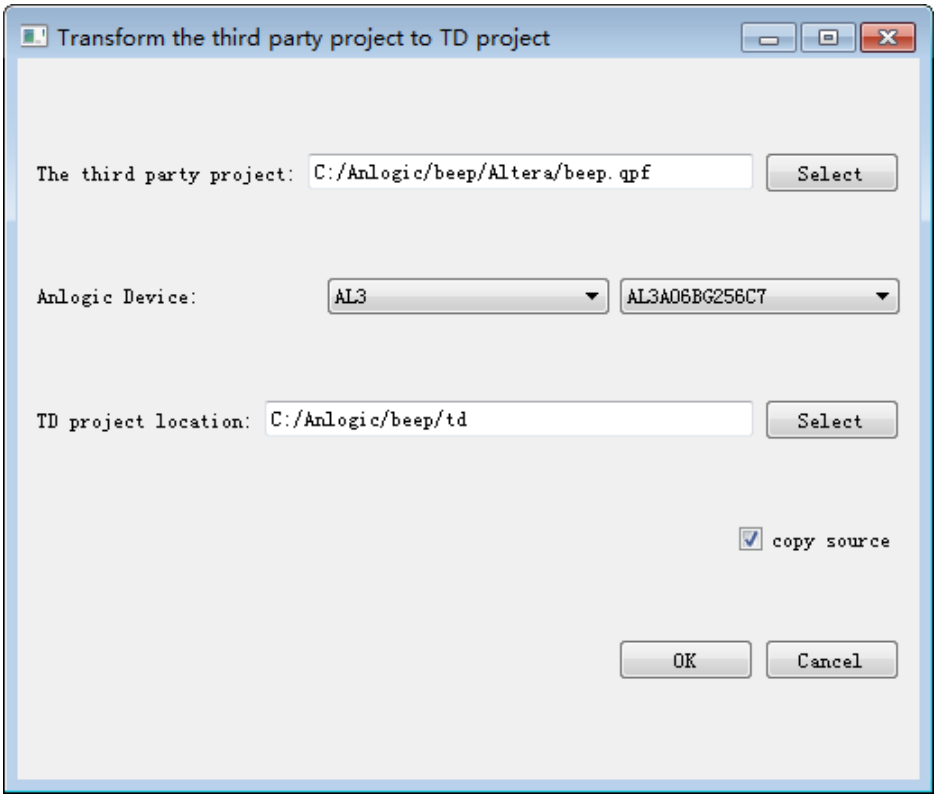
用户可将第三方工具(ISE、Quartus II、Diamond)创建的工程导入到 TD 软件中，在转换过程中，仅转换相应的 Source file、IO Constraint file、Timing Constraint file。只有当第三方器件与 Anlogic 器件在管脚定义兼容的情况下，才会转换 IO Constraint file。

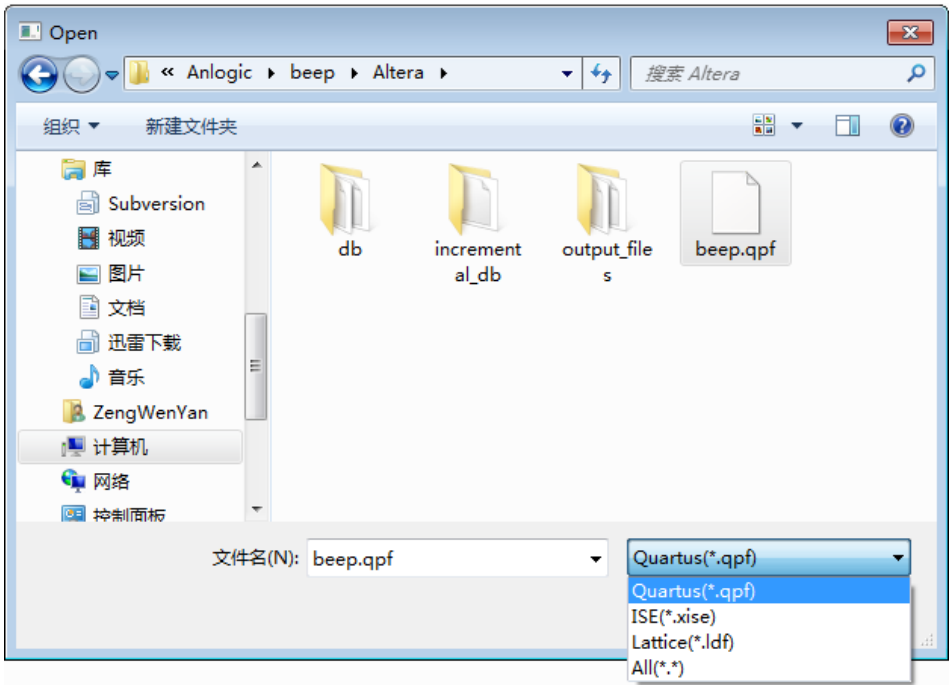
现以 Quartus II 工程转换为 TD 工程为例，介绍该功能：

1. **Project → Transfer Project**



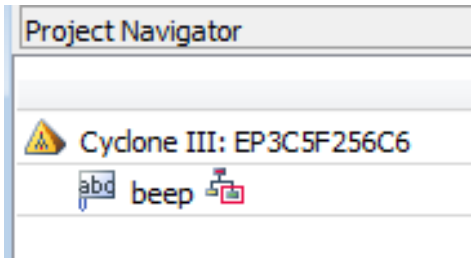
2. 选择需要转换的工程及转换后的工程目录，选择“**copy source**”可将第三方工程中的源文件复制到目标目录。

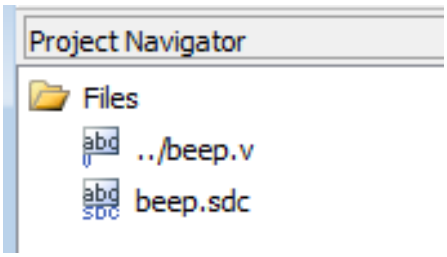




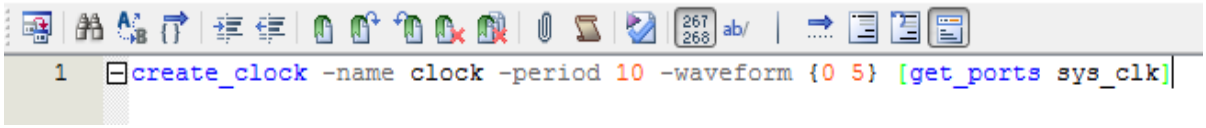
3. TD 会默认打开转换后的工程，以下为转换前后两工程的对比

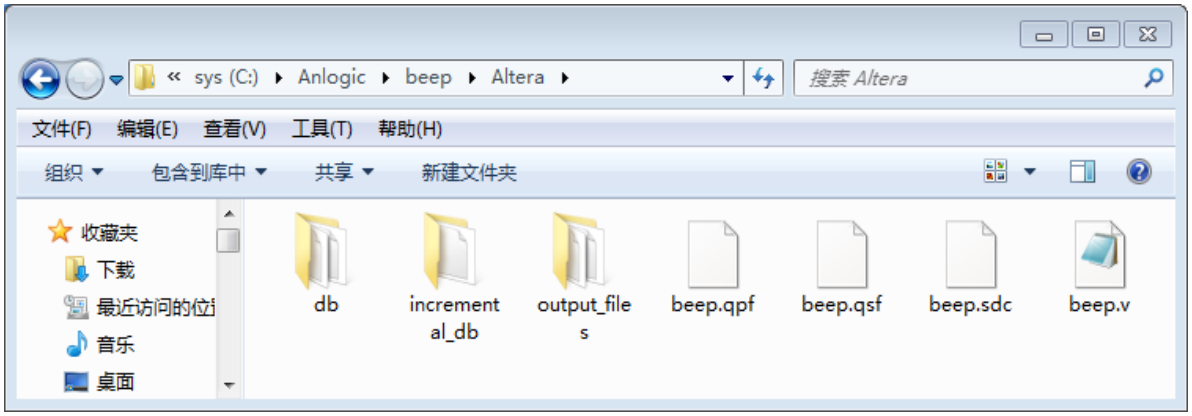
Quartus 工程：



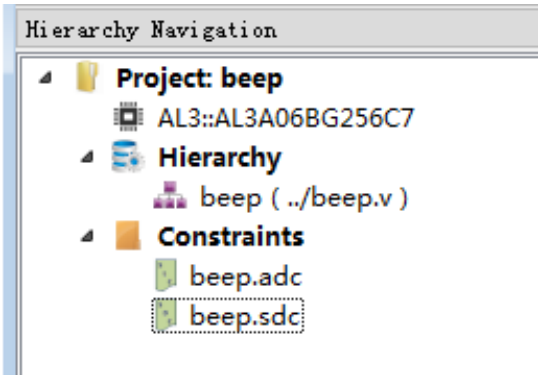


Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate
beep	Output	PIN_A7	8	B8_N0	PIN_A7	2.5 V (default)		8mA (default)	2 (default)
sys_clk	Input	PIN_E1	1	B1_N0	PIN_E1	2.5 V (default)		8mA (default)	
sys_rstn	Input	PIN_T5	3	B3_N0	PIN_T5	2.5 V (default)		8mA (default)	





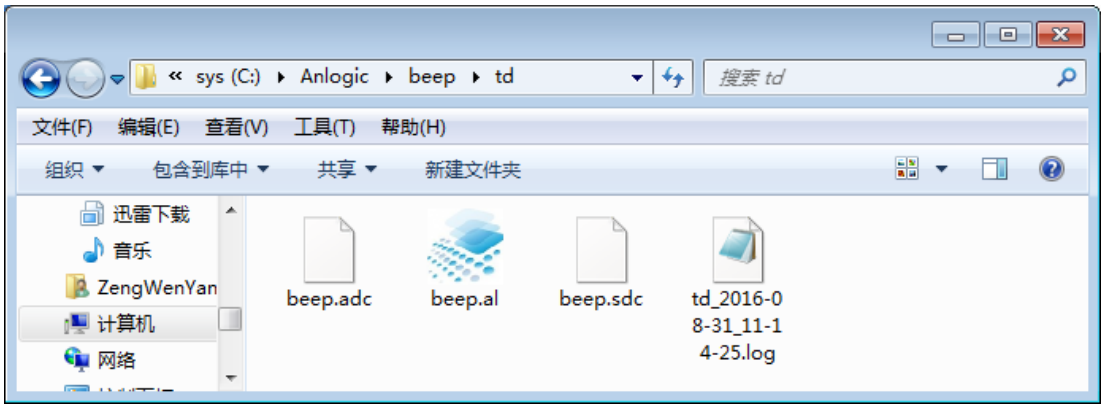
TD 工程：



	Name	Direction	Bank	Location	PullType	IOStandard	SlewRate	DriveStrength	VREF	DiffResistor
1	beep	output	bank8	A7	NONE	LVCMOS25	MED	8	NONE	NONE
2	sys_clk	input	bank1	E1	PULLUP	LVCMOS25	MED	8	NONE	NONE
3	sys_rstn	input	bank3	T5	PULLUP	LVCMOS25	MED	8	NONE	NONE

beep.sdc

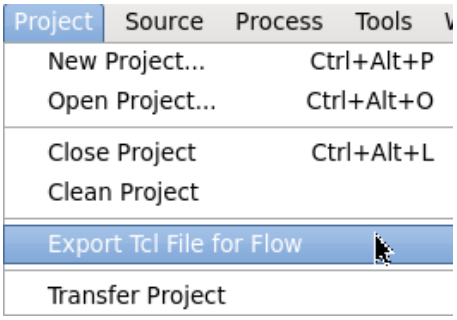
```
1  create_clock -name clock -period 10 -waveform {0 5} [get_ports sys_clk]
```



2.4 导出 tcl 脚本

TD 软件支持使用 tcl 脚本运行 Flow，可减少用户界面操作。

单击 Project ->Export Tcl File for Flow，将会在工程目录中生成 prj_name.tcl 文件，该文件记录了上一次操作 Flow 的所有命令。



如，在界面有如下操作：

1. 打开工程 demo.al

2. 设置参数 Optimize RTL rtl_sim_model ON

3. 运行 HDL2Bit Flow

4. 导出 tcl 脚本 demo.tcl

```
demo.tcl*
1  import_device al3_10.db -package LQFP144
2  open_project C:/Users/wenyan.zeng/Desktop/work/al3/demo/demo.al
3  elaborate -top demo
4  set_param rtl rtl_sim_model on
5  optimize_rtl
6  write_verilog "C:/Users/wenyan.zeng/Desktop/work/al3/demo/demo_rtl_sim.v"
7  report_area -file C:/Users/wenyan.zeng/Desktop/work/al3/demo/demo_rtl.area
8  read_sdc "src/demo.sdc"
9  read_adc "src/demo.adc"
10 export_db "C:/Users/wenyan.zeng/Desktop/work/al3/demo/demo_rtl.db"
11 set_param gate map_sim_model on
12 set_param gate gate_sim_model on
13 map_macro
14 map
15 write_verilog "C:/Users/wenyan.zeng/Desktop/work/al3/demo/demo_map_sim.v"
16 pack
17 write_verilog "C:/Users/wenyan.zeng/Desktop/work/al3/demo/demo_gate_sim.v"
18 report_area -file C:/Users/wenyan.zeng/Desktop/work/al3/demo/demo_gate.area
19 export_db "C:/Users/wenyan.zeng/Desktop/work/al3/demo/demo_gate.db"
20 set_param place effort high
21 start_timer
22 place
23 report_area -io_info -file C:/Users/wenyan.zeng/Desktop/work/al3/demo/demo_phy.area
24 set_param route opt_timing high
25 set_param route effort high
26 set_param route fix_hold on
27 route
28 export_db "C:/Users/wenyan.zeng/Desktop/work/al3/demo/demo_pr.db"
29 start_timer
30 report_timing -mode FINAL -net_info -ep_num 7 -path_num 8 -file "C:/Users/wenyan.zeng/Desktop/work/al3/demo/demo_phy.timing" -sdf C:/Users/wenyan.z
31 write_verilog -sdf "C:/Users/wenyan.zeng/Desktop/work/al3/demo/demo.sdf" "C:/Users/wenyan.zeng/Desktop/work/al3/demo/demo_phy_sim.v"
32 bitgen -bit "C:/Users/wenyan.zeng/Desktop/work/al3/demo/demo.bit" -version 0X0000 -svf "C:/Users/wenyan.zeng/Desktop/work/al3/demo/demo.svf" -svf_c
```

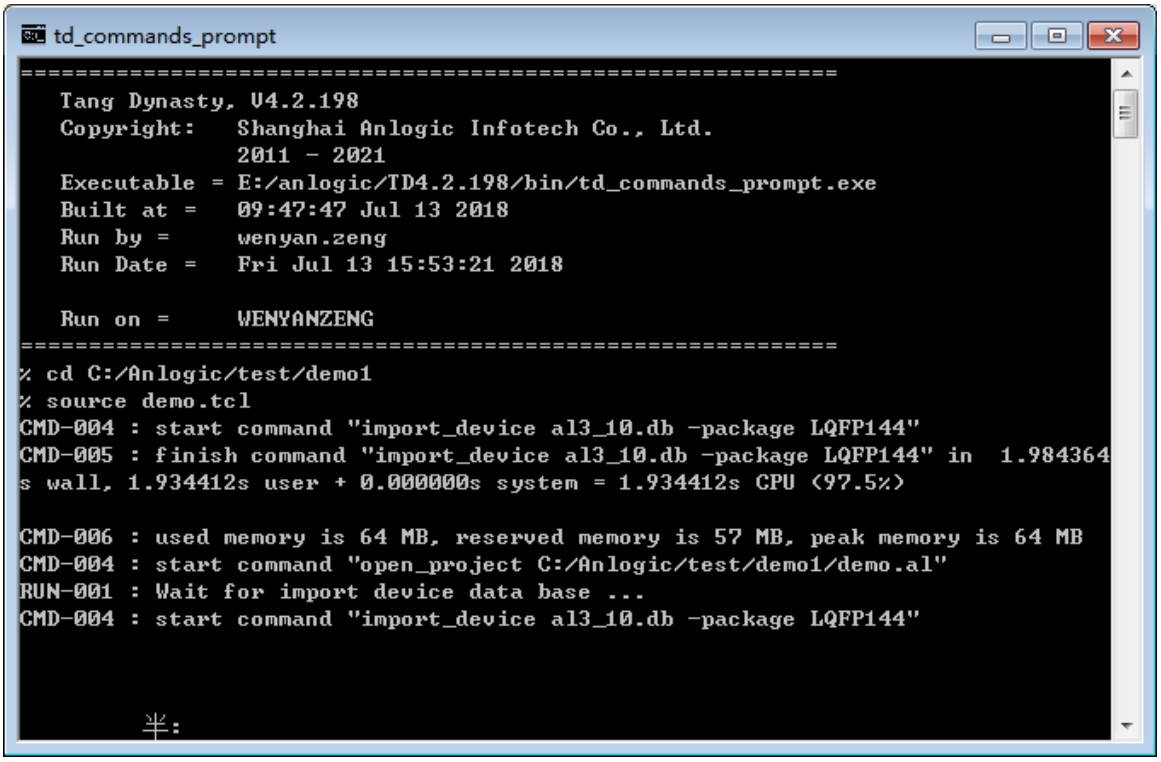
执行 tcl 脚本的步骤为:

1. 在 Windows 下启动 TD CMD 窗口:

Start → All Programs → td_commands_prompt

2. 进入工程所在的目录

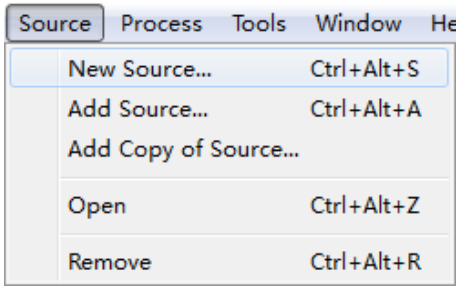
3. 执行命令：source demo.tcl。



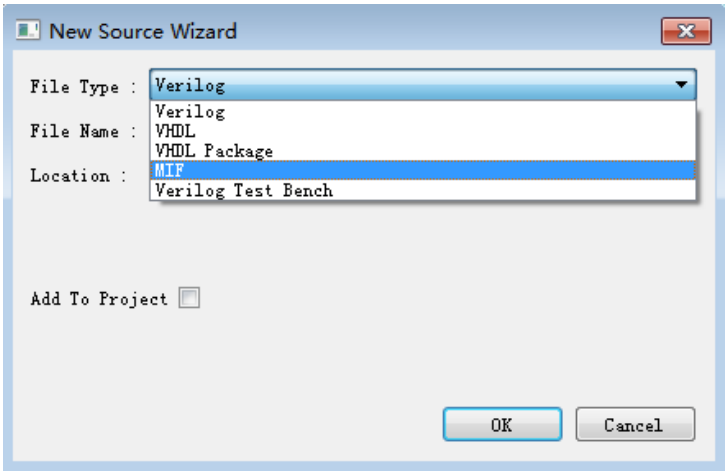
2.5 源文件管理

2.5.1. 新建文件

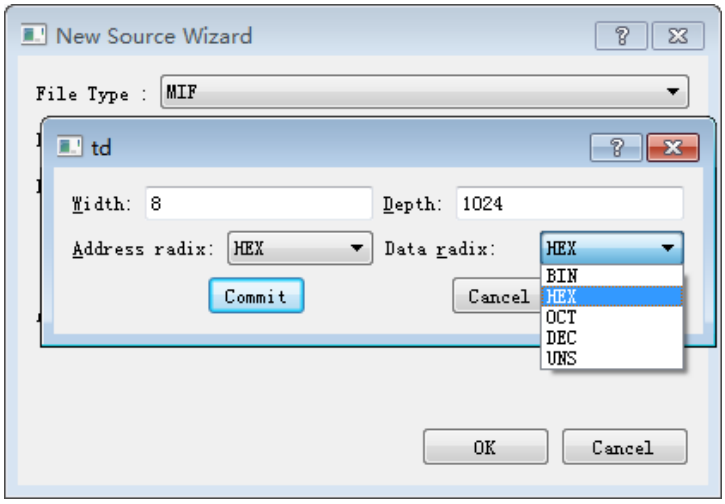
1. **Source → New Source**



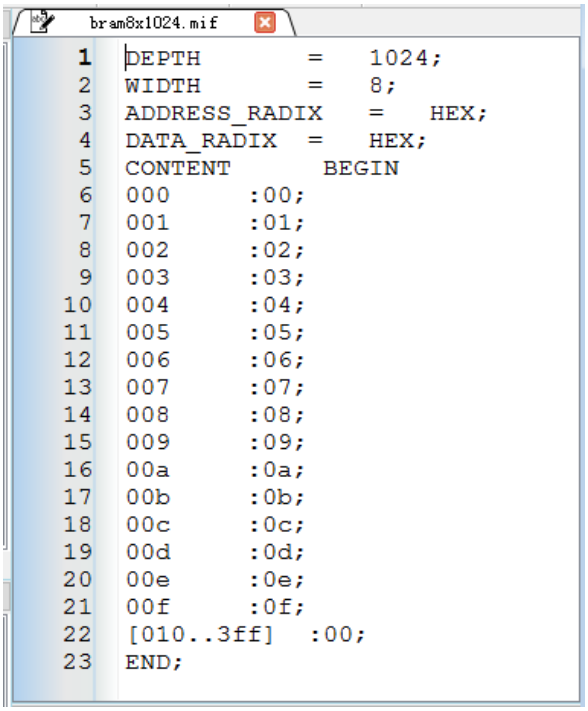
2. 选择生成文件的类型：**Verilog**, **VHDL**, **MIF**，输入文件名称，选择文件路径，并选择是否添加到工程。



3. 当选择的类型为 **MIF** 时，将会出现如下的配置界面：



输入 MIF 文件的宽度和深度，选择数据和地址的基数，生成的 MIF 文件如下所示：

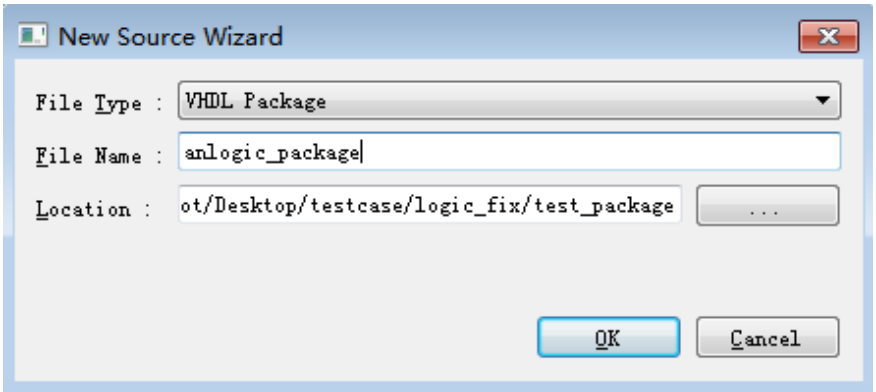


2.5.2 创建 VHDL Package

TD 软件支持用户使用自定义的 VHDL 库文件，在 VHDL 库中可以存储常用的 entity 和 package。具体操作如下：

1. Source → New Source

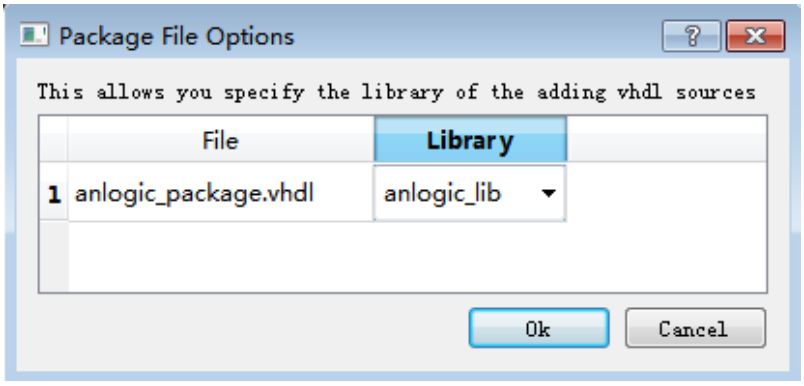
选择 VHDL Package，并填写 File Name，选择文件所在文件夹。



点击 OK 后，会弹出如下图所示的对话框，定义该 package 所归属的 Library 名

称。若工程中定义了多个 Library，可通过下拉菜单进行选择。

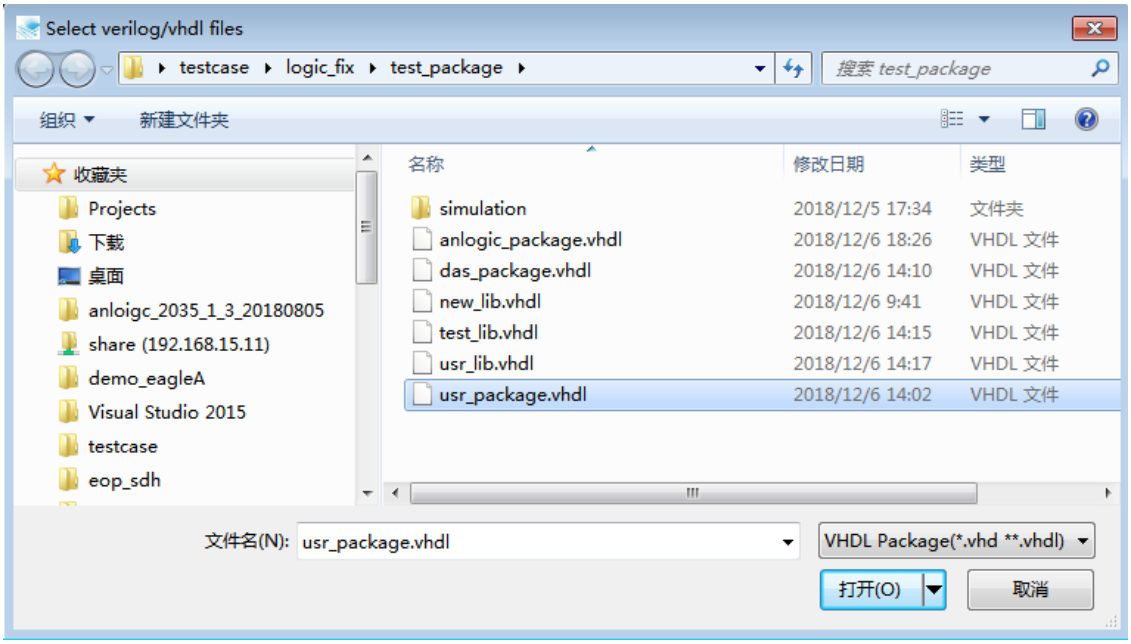
继续点击 OK，将会创建一个新的 anlogic_package.vhdl，并将该文件默认添加到工程中。



在 anlogic_package.vhdl 中定义好相关的 entity 后，即可使用该 Library。

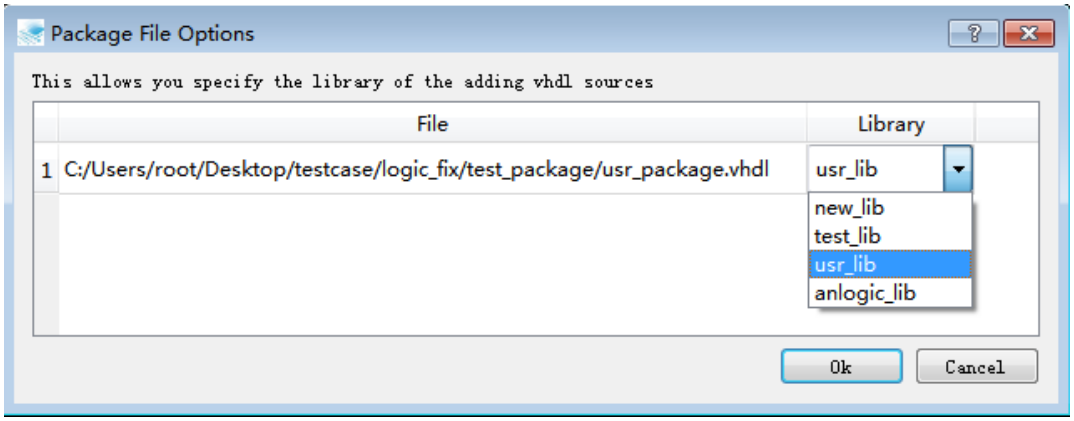
```
21  library IEEE;
22  use IEEE.std_logic_1164.all;
23  library anlogic_lib;
24  use anlogic_lib.anlogic_package.ALL;
25
```

2. 在 **Hierarchy** 中，单击右键，选择 **Add Sources**，选择文件类型为 VHDL Package，

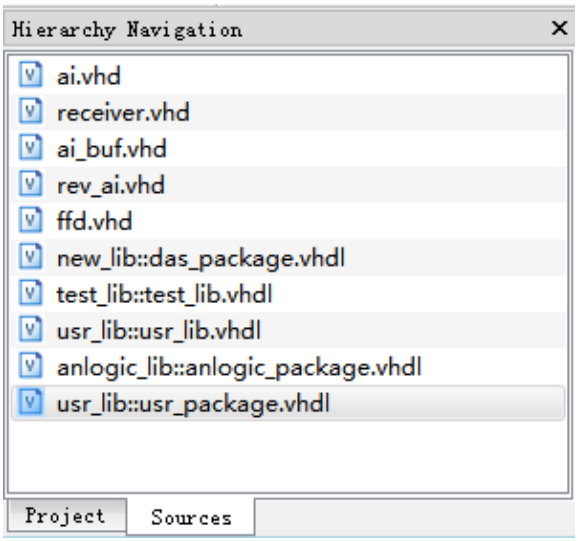


选择一个已存在的 vhdl 文件并打开，为该文件创建一个新的或者选择一个已存在

的 Library。点击 OK 后，默认将 anlogic_package.vhdl 文件添加到工程中，即可使用该 Library。



添加完成后，可在 Hierarchy Navigation 的 Sources 一栏中查看每一个 Library 下所包含的文件。



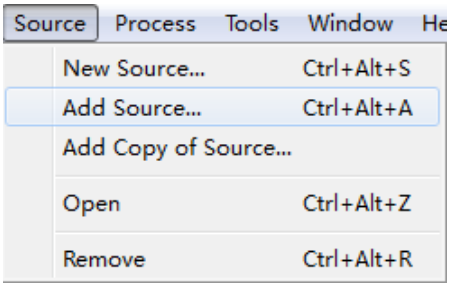
若要移除某个 package 文件，可在 Hierarchy Navigation 的 Sources 一栏中，选中该文件，右键单击，选择 Remove Souce。

注意：一旦该 package 文件被移除，若要再次使用，需要重新为其指定 Library。

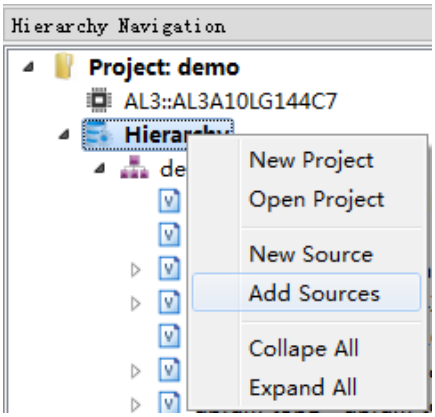
2.5.3 添加和移除文件

添加文件有两种方式：

1. **Source → Add Source**

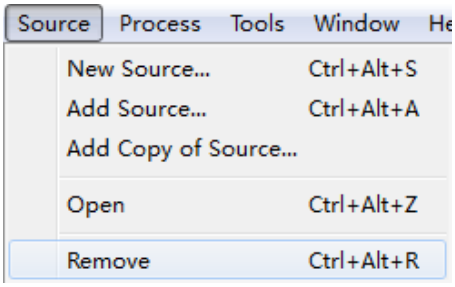


2. 在 **Hierarchy** 中，单击右键，选择 **Add Sources**

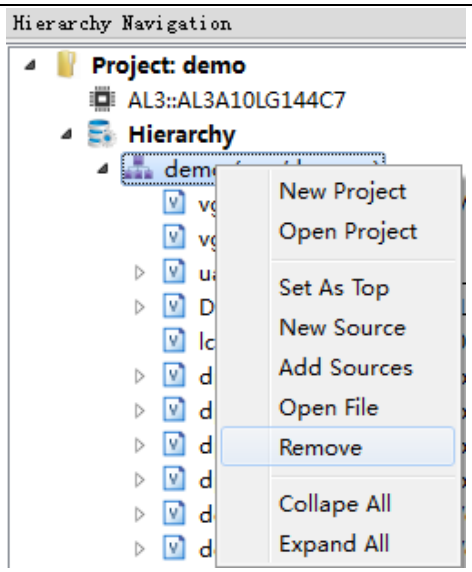


移除文件同样有两种方式：

1. **Source → Remove**



2. 在 **Hierarchy** 中，选择某个文件并单击右键，选择 **Remove**



2.5.4 编辑文件

TD Editor 对编辑文件有很多方便的功能，具体操作可通过菜单栏中的 **Edit** 选项进行查看。

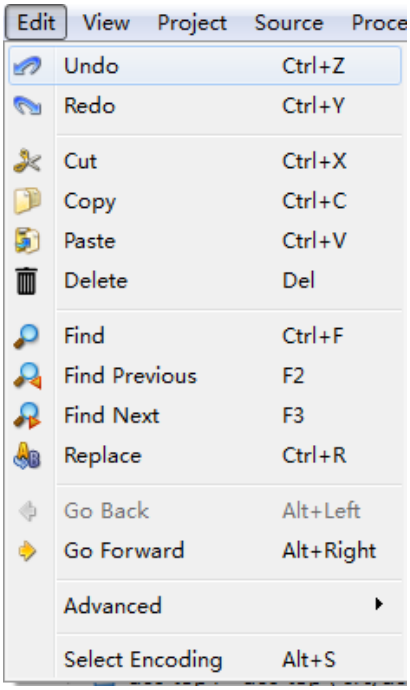
Undo，**Redo** 可在编辑时进行撤销和重做；

Cut，**Copy**，**Paste**，**Delete** 与常规的剪切，复制，粘贴，删除功能一致；

Find 查找功能，**Find Previous** 查找上一个，**Find Next** 查找下一个，**Replace** 替换功能；

Go Back 跳回当前行的首端，**Go Forward** 跳转到当前行的末端；

Select Encoding 对字符进行编码。

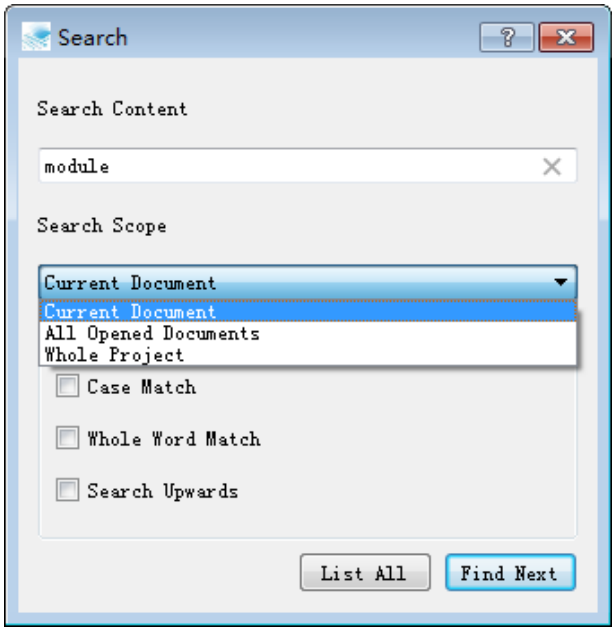


下面主要介绍查找替换功能和 **Advanced** 中涉及到的功能：

1. 查找功能

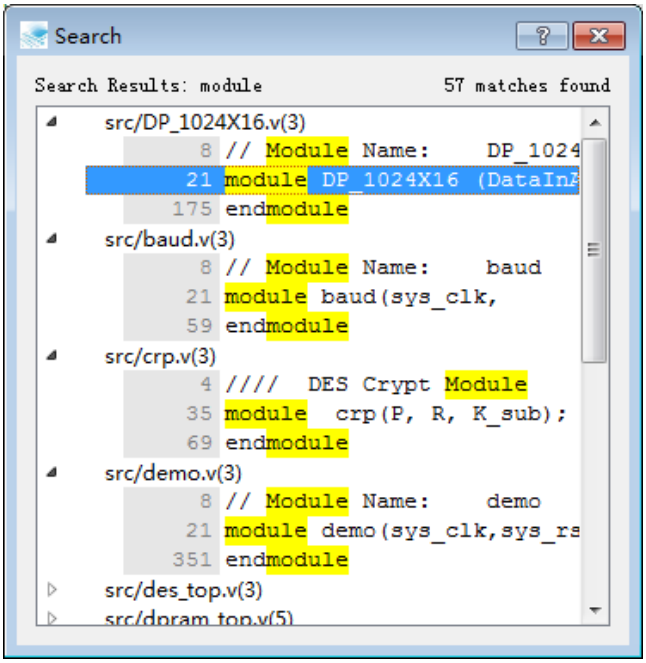
通过 **Edit → Find**，或者快捷方式 **Ctrl + F** 进入功能，将会出现如下选择框：

输入要查找的字符，选择搜索的范围：当前文档、所有打开的文档或整个工程，也可根据需求选择匹配的方式：大小写匹配、整词匹配或向上向下所搜。



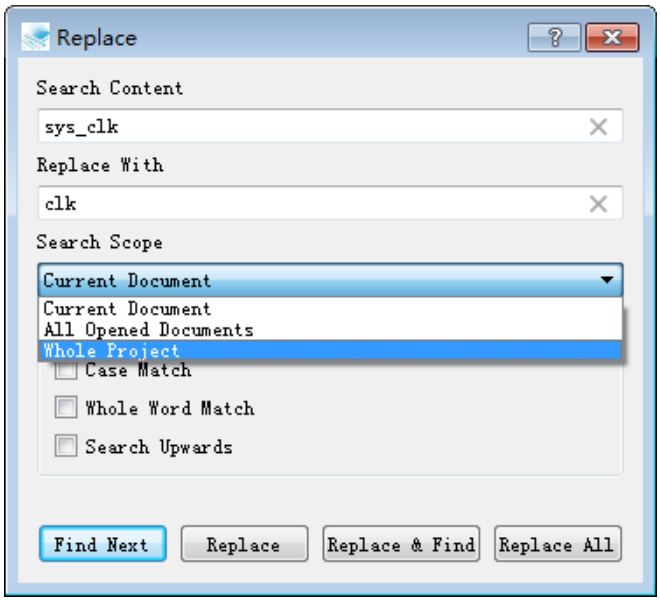
当点击 List All 时，将会列出在搜索范围内所查找到的所有相关字符，并且可

通过双击跳转至该字符所在源文件的位置。



2. 替换功能

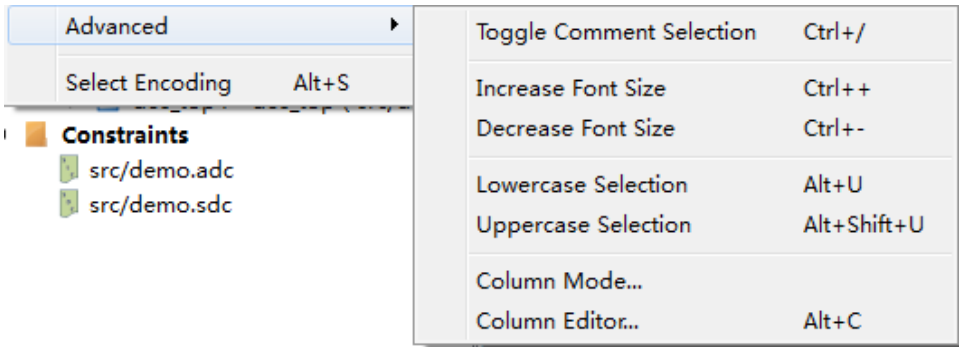
通过 **Edit → Find**，或者快捷方式 **Ctrl + R** 进入功能，将会出现如下选择框：



输入想要查找的字符，并输入替换的内容，同样可以选择搜索的范围和匹配方式，如选择搜索范围为“**Whole Project**”，并点击“**Replace All**”，则会将整个工程中的所有 sys_clk 都替换为 clk。

3. **Advanced** 功能

展开 **Edit → Advanced**，可以看到有如下功能：



Toggle Comment Select 对选中的代码进行注释，如果选中的为已经注释的代码，则会解除注释；

Increase Font Size 放大字体；

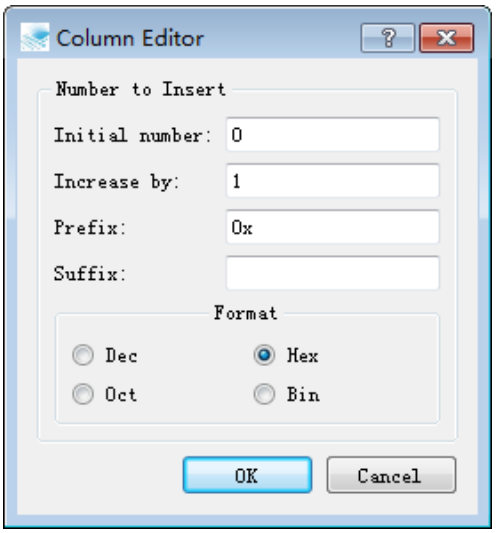
Decrease Font Size 缩小字体；

Lowercase Selection 转换选中的字符为小写字符；

Uppercase Selection 转换选中的字符为大写字符；

Column Mode…列操作模式；

Column Editor…列编辑器，如下所示，可在列操作模式下，进行递增，并可选择输入数据的前缀或后缀。



3 IP 生成器

IP 生成器是一个创建 IP 核的图形交互设计界面。用户可以在 IP 生成器中对所选 IP 进行配置，并自动生成相应的 IP 模块。目前支持的 IP 模块有 **COMMON**、**PLL**、**DSP**、**RAM**、**FIFO**、**DRAM**、**SDRAM**、**MCU**、**ADC**。(ELF 系列的器件仅支持 DRAM 模块。)

3.1 COMMON 模块

Common 模块中包含了一些常用的单元：BUFG、IDELAY、IDDR、ODDR。

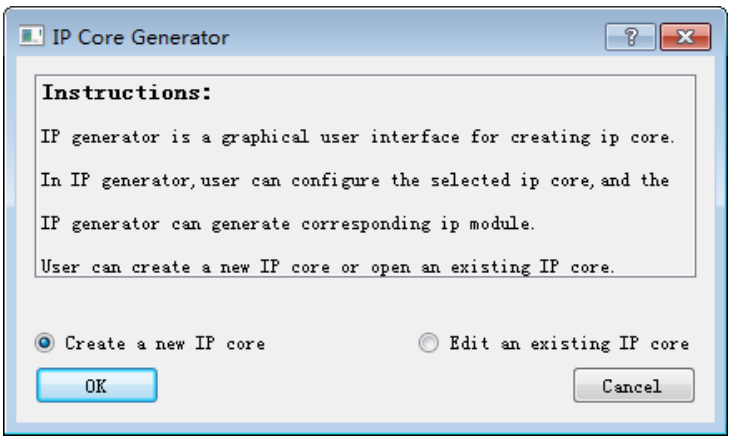
3.1.1 BUFG 模块

全局时钟模块，可减少全局时钟信号的延时与偏移。

注：BUFG 模块的使用条件有所限制，在 GCLK IO 与 PLL 的输出端口后不能添加，而在大多数情况下 TD 软件将自动适时的为时钟信号添加 BUFG 模块。建议只有在软件没有添加的情况下才手动例化该模块。

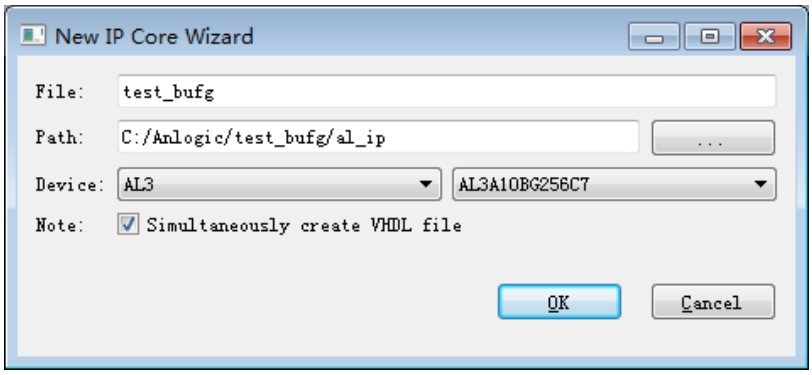
1. 创建 BUFG 模块

选择 **Tools** → **IP Generator**，选择“**Create a new IP core**”

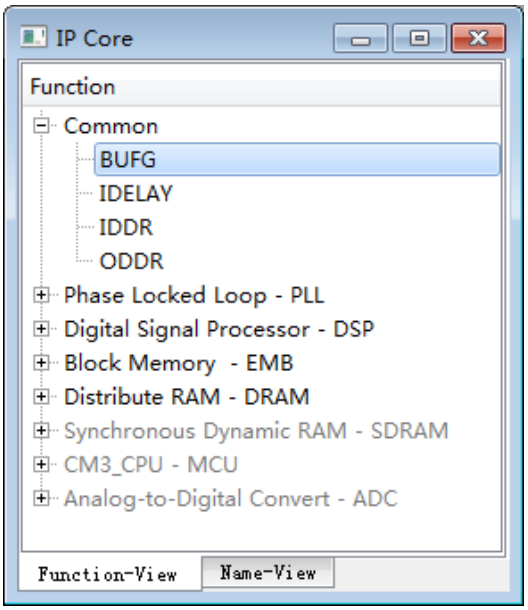


输入模块名称并选择存储路径。此处，若是在有工程的基础上创建 BUFG 模块，存储路径和器件名称将与工程保持一致。若在没有工程的基础上创建 BUFG 模块，用户需

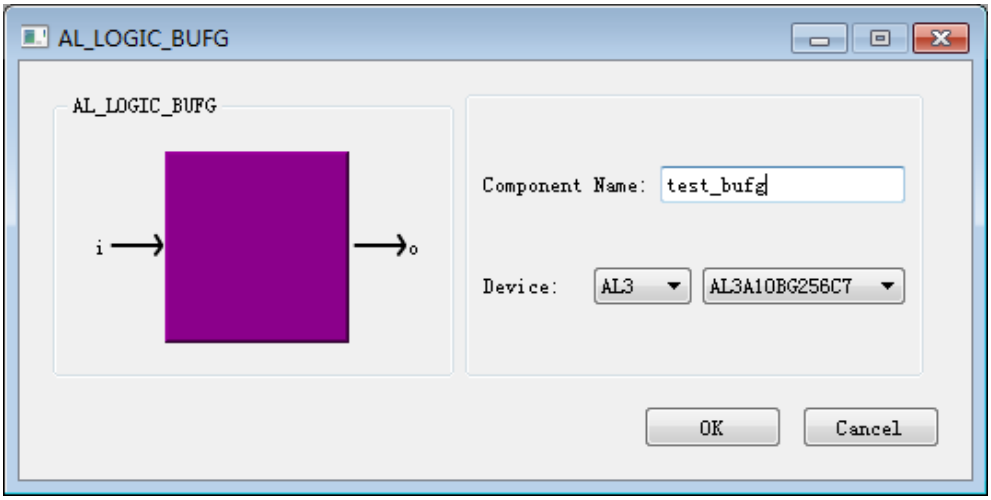
手动设置保存路径和器件名称。若勾选“**Simultaneously create VHDL file**”, TD 将会生成相应的 VHDL 文件。



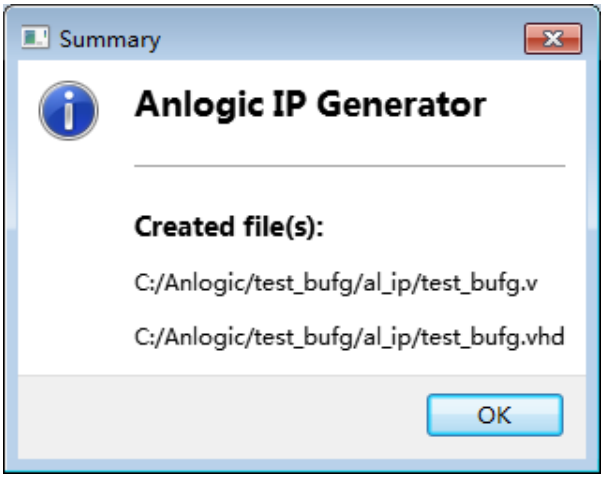
在 Function 窗口中展开 Common 模块，双击 **BUFG** 打开配置界面



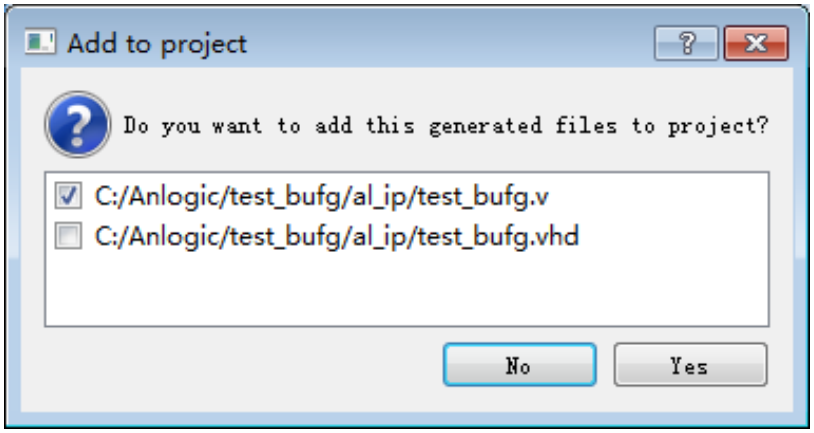
输入模块名称，选择相应的器件，默认为工程器件



点击“OK”完成设置，生成文件如下：



继续点击“OK”，并选择是否添加文件至工程。



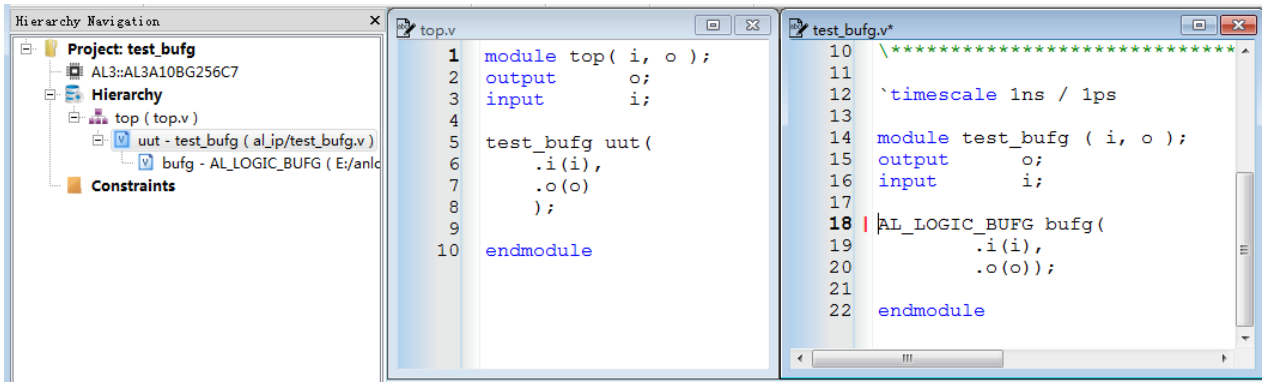
2. 例化 BUFG 模块

以新建工程为例介绍例化 BUFG 模块的过程。用户在已有工程的基础上进行例化的过程一致。

新建工程，并为工程添加顶层模块;

在工程中添加上一步生成的 test_bufg.v;

在顶层模块中调用 test_bufg 模块，并修改 inst 名称和端口名称，点击保存按钮，即完成了 BUFG 模块的例化。点击 **File → Save** 保存文件。



~~IDELAY 模块描述是否需要增加?~~

~~IDELAY，IO 延时单元~~

~~* 特殊 IP 的使用：以 AL3S10 为例，可使用该单元调节 RGMII 信号的输入延时。~~

~~AL_LOGIC_IDELAY U0_AL_LOGIC_IDELAY(.i(PHY1_RXDV),.o(rxdv_int));~~

~~defparam U0_AL_LOGIC_IDELAY.INDEL = 1;~~

~~使用后有个初始延时，为 0.2ns，参数用于设置延时长度，每增加 1，增加延时 0.1ns。比如，参数设置为 1 时，延时为 0.3ns。~~

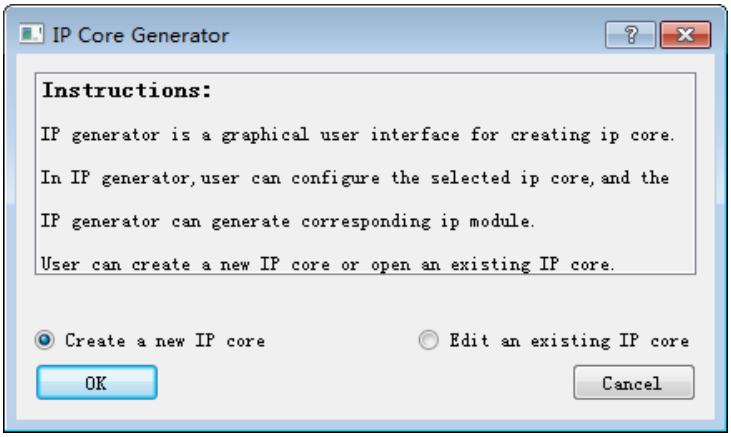
IDELAY 模块不用描述，这个就应该从界面上拿掉的。

3.1.2 IDDR 模块

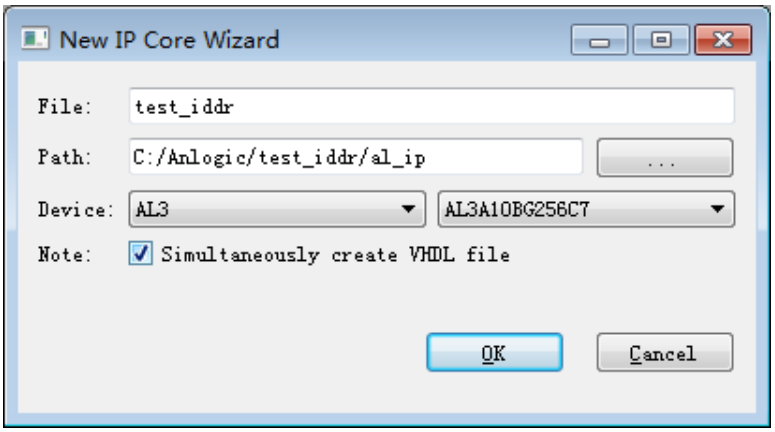
输入双沿采样模块，是一个专用的输入寄存器，可用于对输入信号的双沿采样。

1. 创建 IDDR 模块

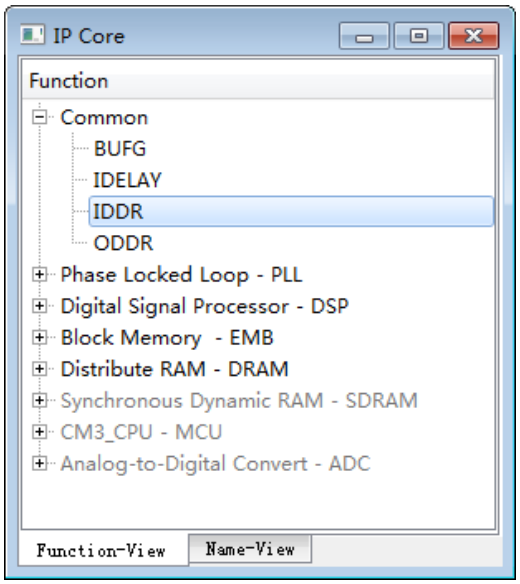
选择 **Tools** → **IP Generator**，选择“**Create a new IP core**”



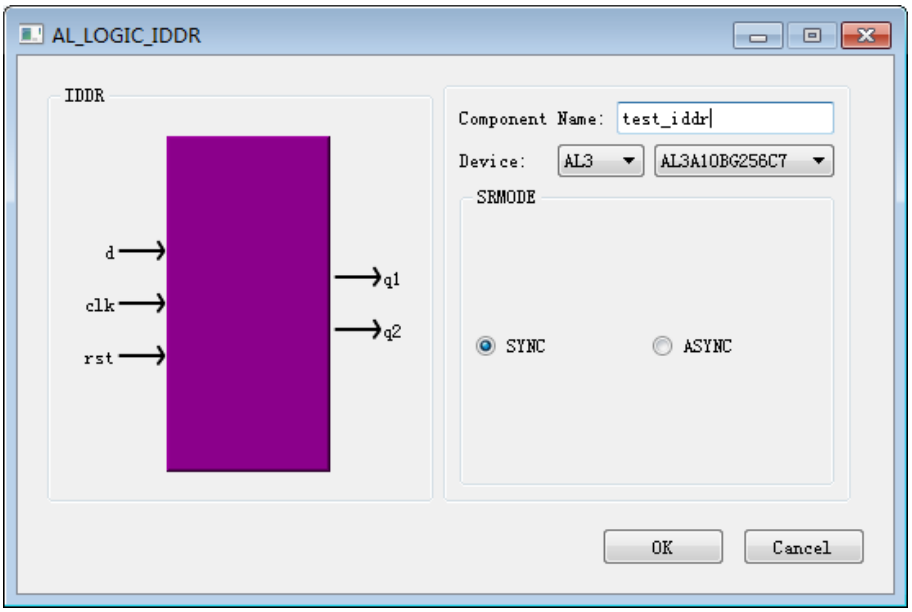
输入模块名称并选择存储路径。此处，若是在有工程的基础上创建 IDDR 模块，存储路径和器件名称将与工程保持一致。若在没有工程的基础上创建 IDDR 模块，用户需手动设置保存路径和器件名称。若勾选“**Simultaneously create VHDL file**”, TD 将会生成相应的 VHDL 文件。



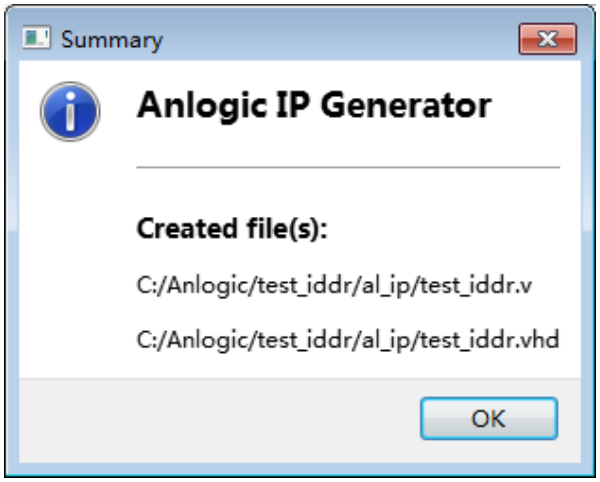
在 Function 窗口中展开 Common 模块，双击 **IDDR** 打开配置界面



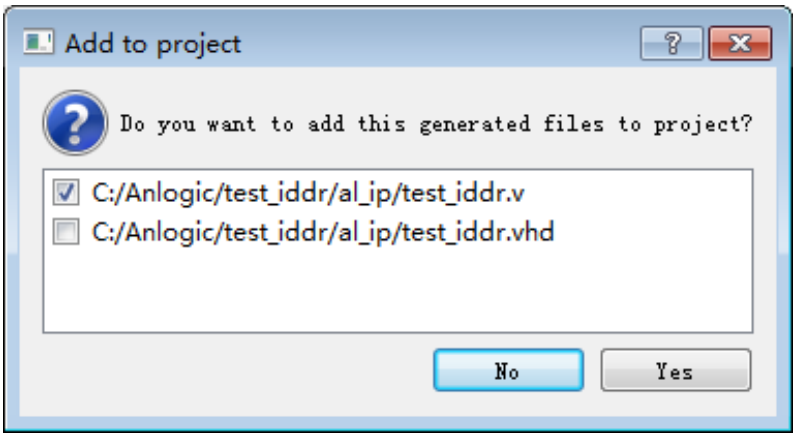
输入模块名称，选择相应的器件，默认为工程器件



点击“OK”完成设置，生成文件如下：



继续点击“OK”，并选择是否添加文件至工程。



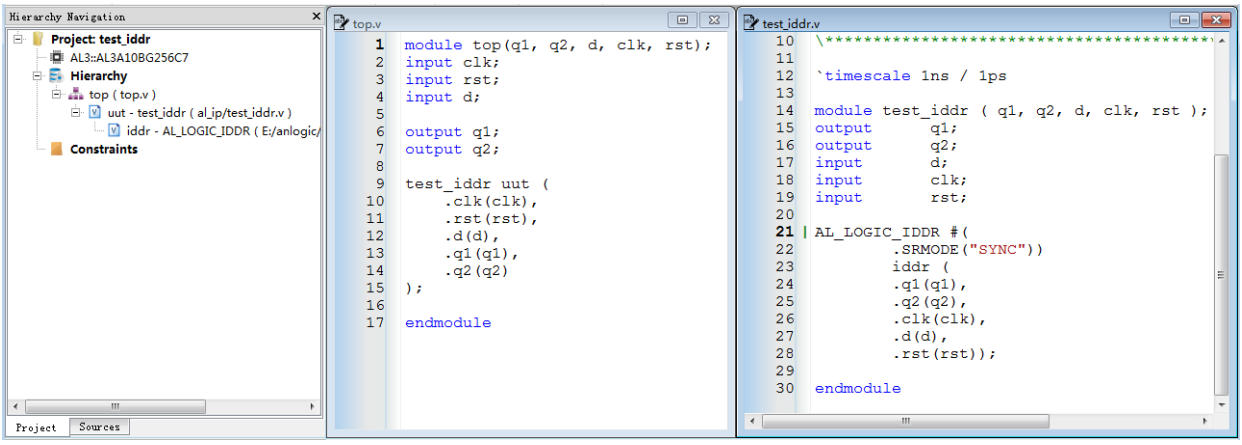
2. 例化 IDDR 模块

以新建工程为例介绍例化 IDDR 模块的过程。用户也可以在已有工程的基础上进行例化，例化过程一致。

新建工程，并为工程添加顶层模块;

在工程中添加上一步生成的 test_iddr.v;

在顶层模块中调用 test_iddr 模块，并修改 inst 名称和端口名称，点击保存按钮，即完成了 IDDR 模块的例化。点击 **File → Save** 保存文件。



*对于 AL3S10 等器件，IDDR 可用于对 RGMII 输入信号的双边沿采样

AL_LOGIC_IDDR IDDR_0

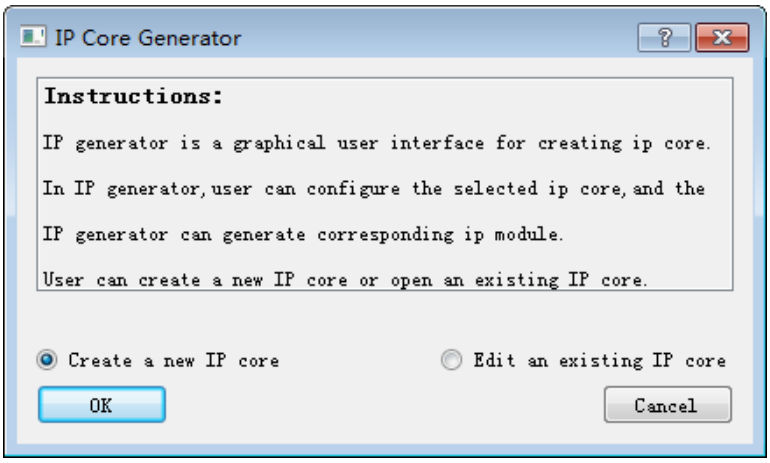
(.q1(rxd_r2g_tmp[3]), .q2(rxd_r2g_tmp[7]), .clk(rxc), .d(rxd[3]), .rst(~rst_n));

3.1.3 ODDR 模块

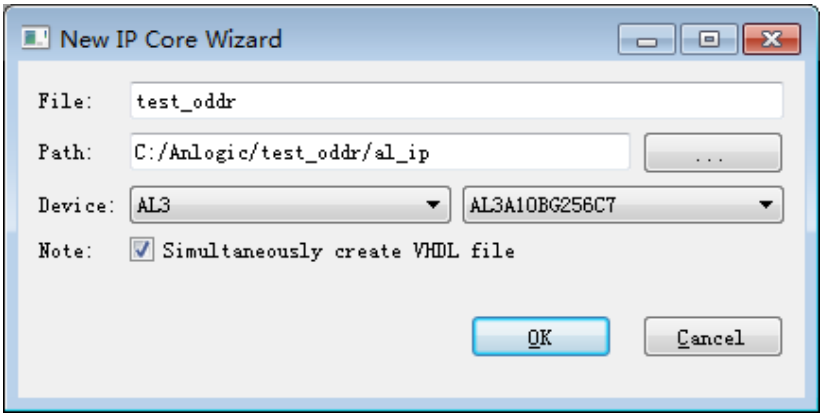
输出双沿驱动模块，可用于对输出信号的双沿驱动。

1. 创建 ODDR 模块

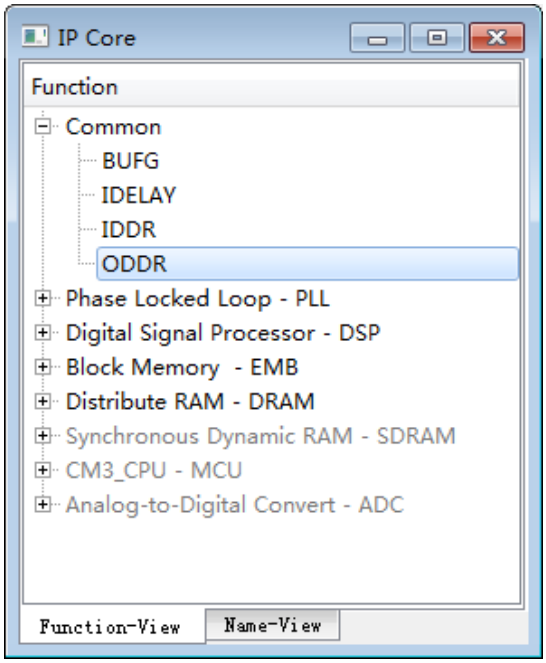
选择 **Tools** → **IP Generator**，选择“**Create a new IP core**”



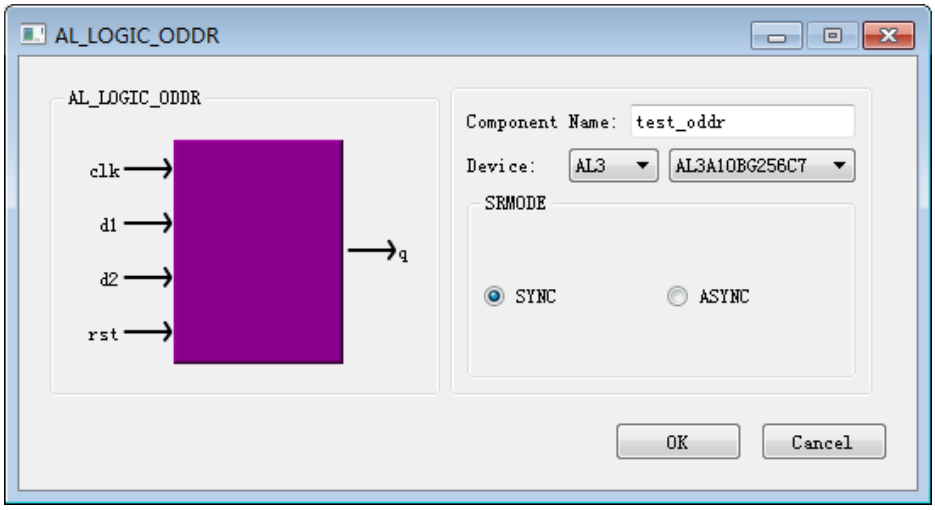
输入模块名称并选择存储路径。此处，若是在有工程的基础上创建 ODDR 模块，存储路径和器件名称将与工程保持一致。若在没有工程的基础上创建 ODDR 模块，用户需手动设置保存路径和器件名称。若勾选“**Simultaneously create VHDL file**”, TD 将会生成相应的 VHDL 文件。



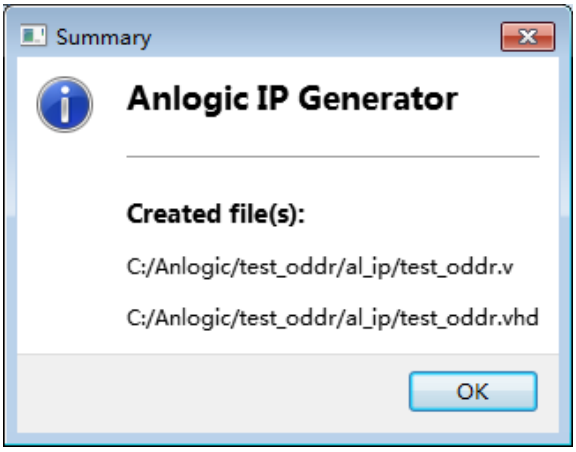
在 Function 窗口中展开 Common 模块，双击 **ODDR** 打开配置界面



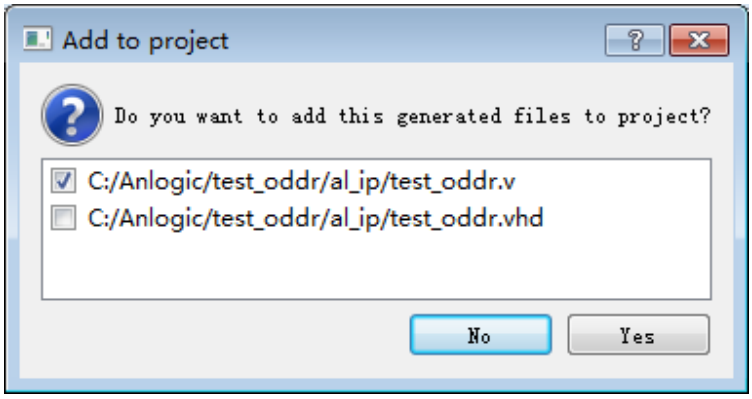
输入模块名称，选择相应的器件，默认为工程器件



点击“OK”完成设置，生成文件如下：



继续点击“OK”，并选择是否添加文件至工程。



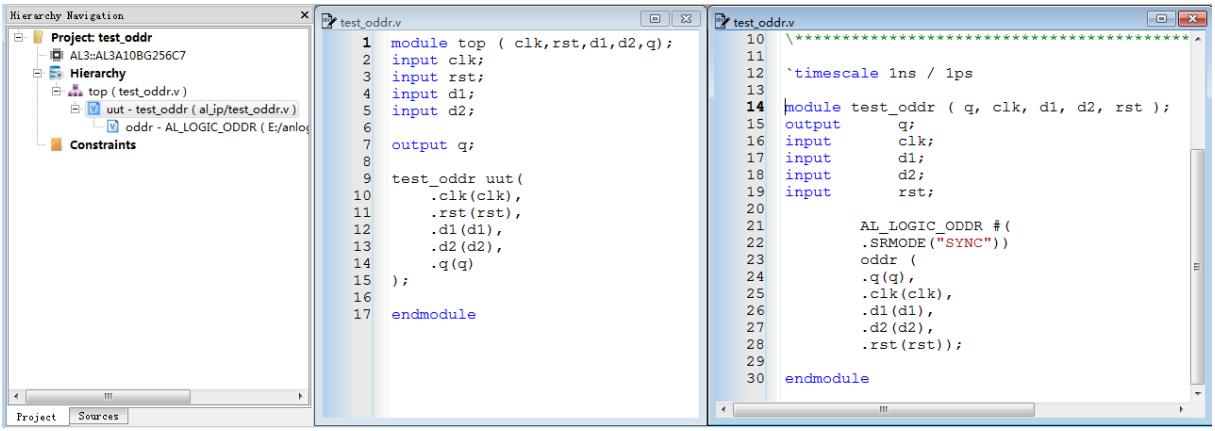
2. 例化 ODDR 模块

以新建工程为例介绍例化 ODDR 模块的过程。用户也可以在已有工程的基础上进行例化，例化过程一致。

新建工程，并为工程添加顶层模块;

在工程中添加上一步生成的 test_oddr.v;

在顶层模块中调用 test_oddr 模块，并修改 inst 名称和端口名称，点击保存按钮，即完成了 ODDR 模块的例化。点击 **File → Save** 保存文件。



*对于 AL3S10 等器件，ODDR 用于对 RGMII 输出信号的双边沿驱动

AL_LOGIC_ODDR ODDR_0

(.q(txd[0]), .clk(txc_tmp), .d1(txd_tmp[4]),　.d2(txd_tmp[0]), .rst(RST_OUT0));

3.2 PLL 模块

本手册以 EAGLE 系列介绍 PLL 模块。EAGLE 系列 FPGA 最多内嵌有 4 个多功能锁相环(PLL0~PLL3),可实现高性能时钟管理功能。每个 PLL 都能实现时钟分频/倍频、输入和反馈时钟对准、多相位时钟输出功能。

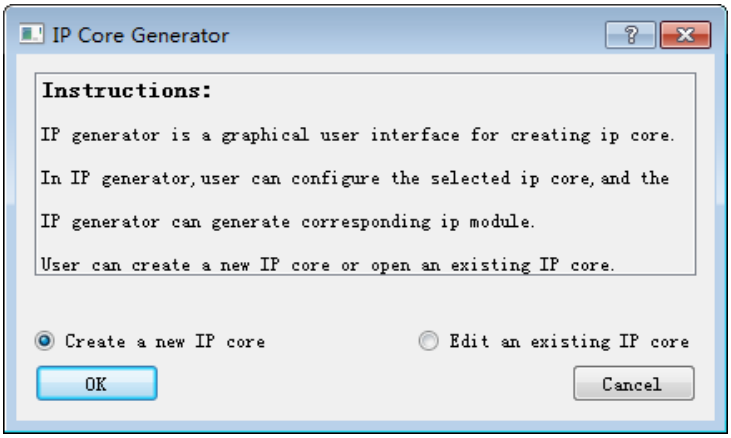
PLL 参考时钟输入有：时钟网络输出、互连输出和内部振荡器输出。

PLL 反馈时钟输入有：时钟网络输出、内部寄存器时钟节点、互连输出、PLL 内部反馈时钟以及相移时钟 C0~C4。

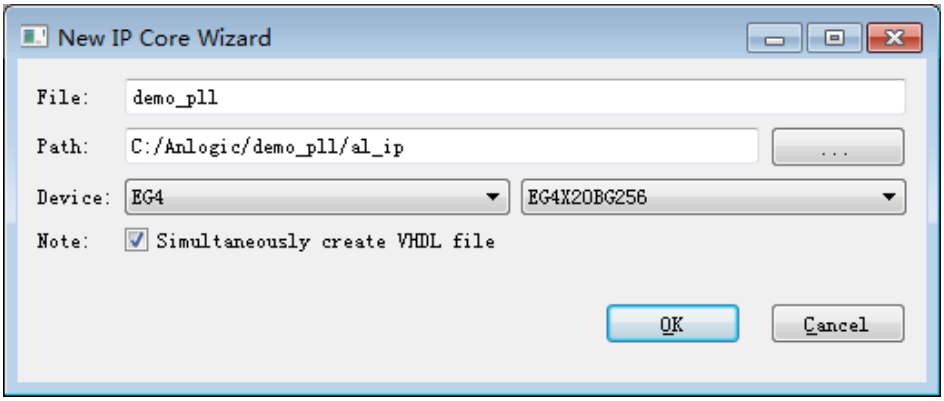
PLL 有输出驱动芯片的专用时钟输出管脚。

3.2.1 创建 PLL 模块

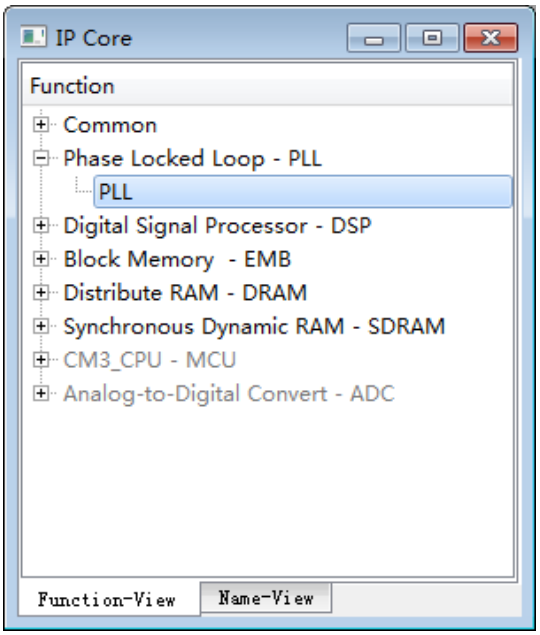
1. 选择 **Tools** → **IP Generator**，选择“**Create a new IP core**”



2. 输入模块名称并选择存储路径。此处，若是在有工程的基础上创建 PLL 模块，存储路径和器件名称将与工程保持一致。若在没有工程的基础上创建 PLL 模块，用户需手动设置保存路径和器件名称。若勾选“**Simultaneously create VHDL file**”, TD 将会生成相应的 VHDL 文件。

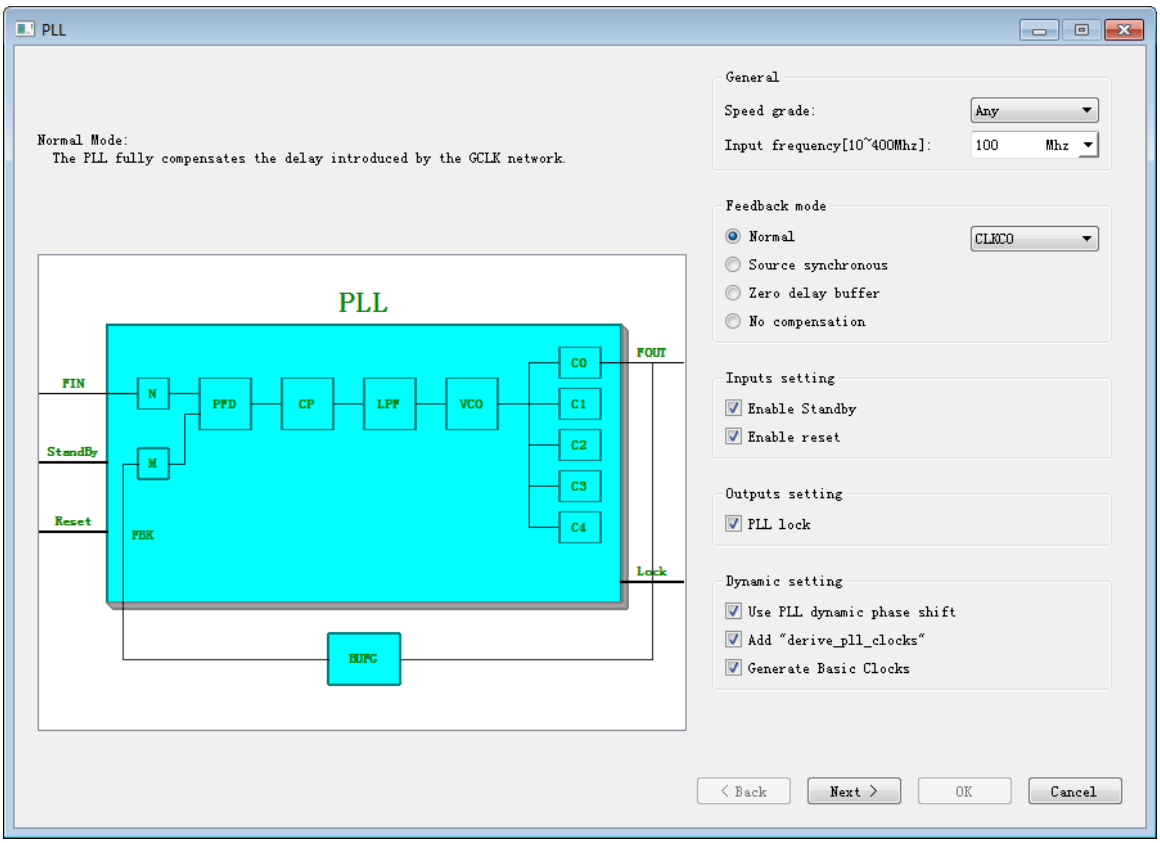


3. 在 Function 窗口中展开 **Phase Locked Loop**，双击 **PLL** 打开配置界面。



4. 设置 PLL 的相关参数

1) PLL 模式设置



PLL 支持 4 种反馈模式，每种模式都支持时钟分频/倍频和相移。

a) 普通模式（**Normal**）

普通模式中，PLL 会补偿 GCLK 网络延迟，保证内部寄存器输入时钟相位和时钟管脚相位一致。

b) 源同步模式（**Source-Synchronous**）

源同步模式通过动态相移功能，调节时钟相位保证数据端口到 IOB 输入寄存器的延迟和时钟输入端口到 IOB 寄存器的延迟相等（数据和时钟输入端口模式相同情况下）。

c) 无补偿模式（**No Compensation**）

在无补偿模式，PLL 不对时钟网络延迟进行补偿，PLL 采用内部自反馈，这会

提高 PLL 的抖动特性。

d) 零延迟缓冲模式（**Zero Delay Buffer**）

零延迟缓冲模式，时钟输出管脚相位和 PLL 参考时钟输入管脚相位对齐。

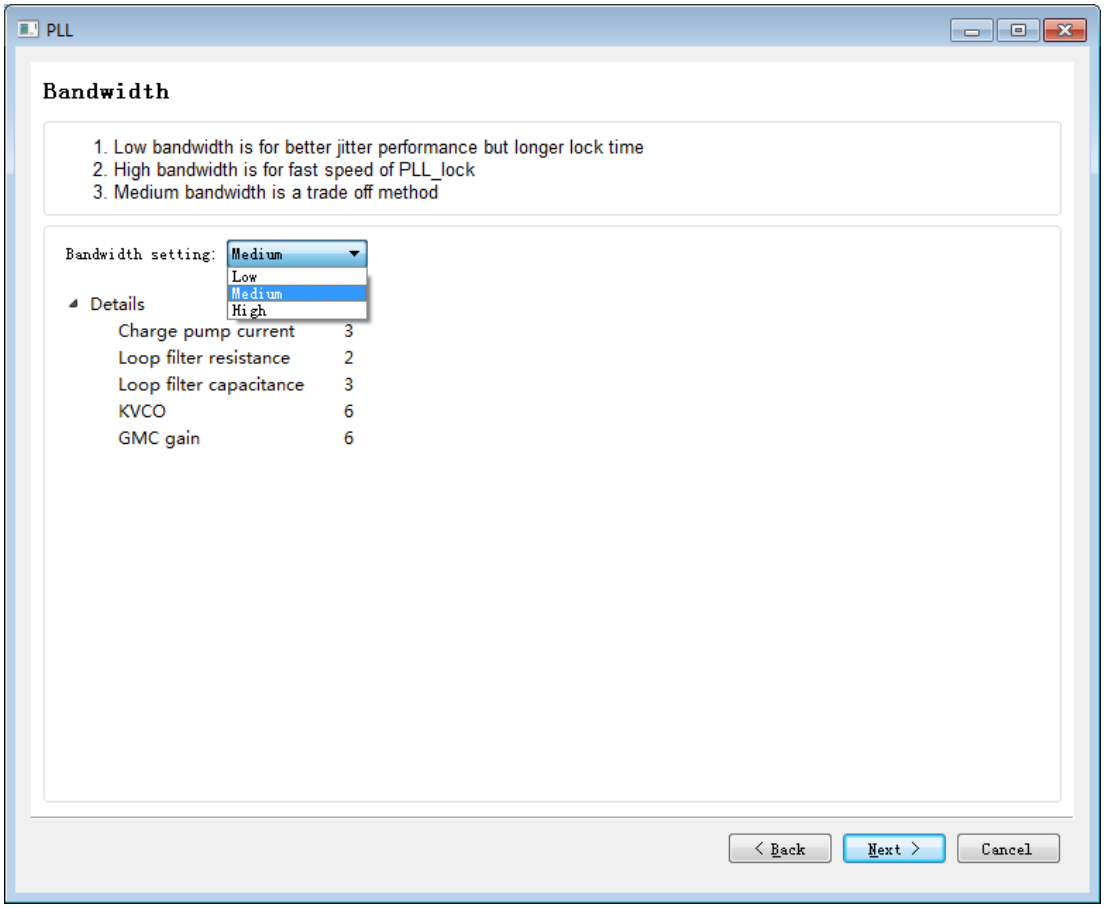
PLL 参数特性如下表所示：

Parameter	**Feature**
输入时钟频率范围	10~400 MHz
输出时钟频率范围	4~400 MHz
VCO 频率范围	300~1200 MHz
输出端口数	5 （各端口相位独立可选）
参考时钟分频系数（M）	1~128
反馈时钟分频系数（N）	1~128
输出时钟分频系数（C0~C4）	1~128
相移分辨率	45°
输出端口可选相位偏移量（度）	0, 45, 90, 135, 180, 225, 170, 315
用户动态相移控制	支持（+/-每单位 45 度相移）
锁定状态输出	Lock
专用时钟输出管脚	支持

当选择“**Add derive_pll_clocks**”时，在编译工程时会自动在所有用到的 PLL clkc[x]端口生成时钟约束，生成时钟的频率、相位都将严格按照 PLL 内部的参数设定。而选择“**Generate Basic Clocks**”将会在对应的 PLL refclk 上定义 FIN 频率的基准时钟，否则将自动搜索 refclk pin 以及所连 net 上定义的时钟，没找到则报错。

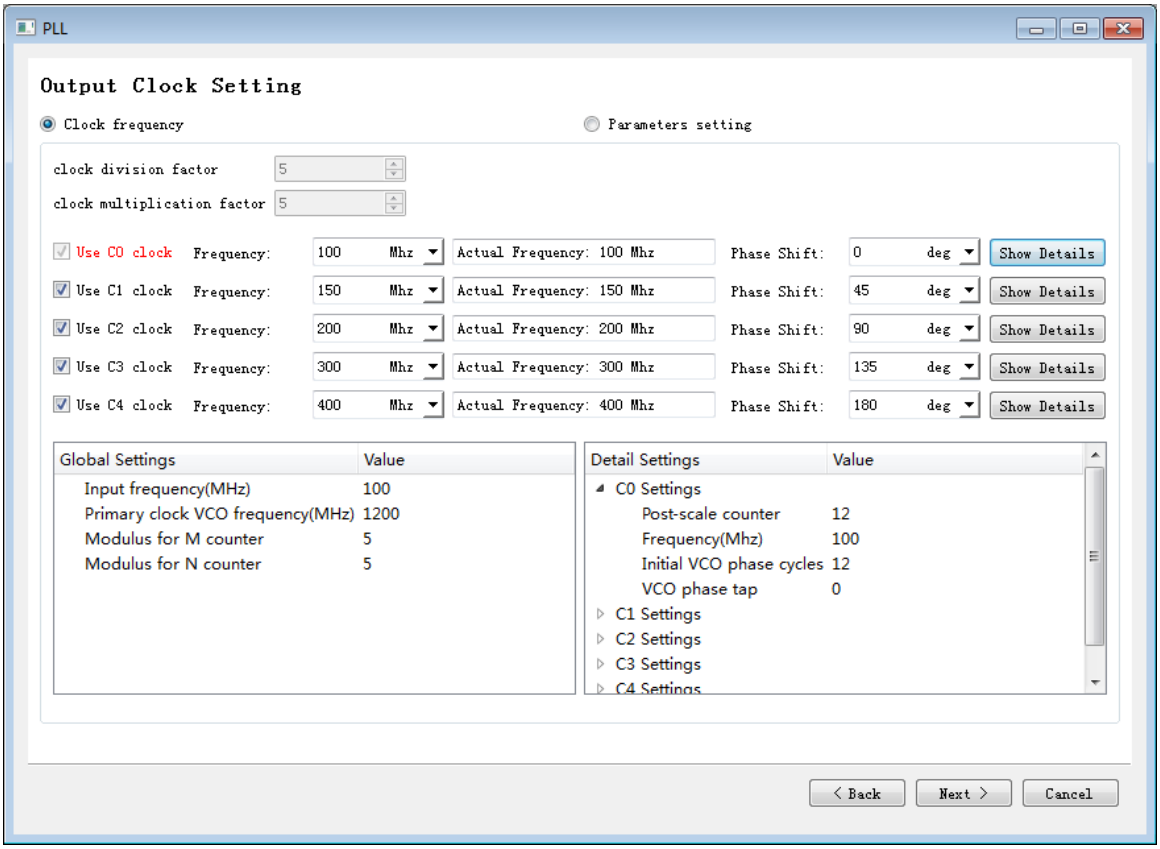
2）Bandwidth 的设置

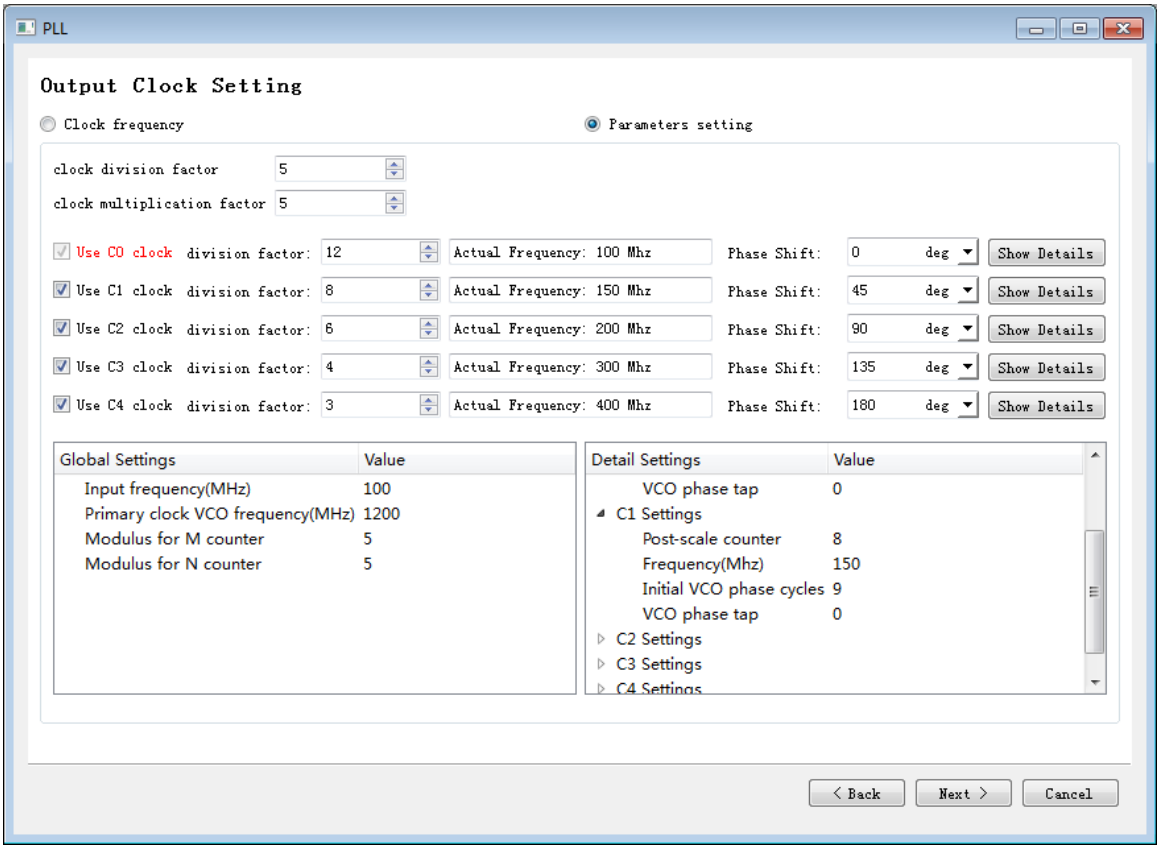
可分别设置 Bandwidth 的值为 **Low**、**Medium**、**High**，默认值为 **Medium**。点击“**Show Details**”可查看该带宽下，PLL 各性能参数的值。



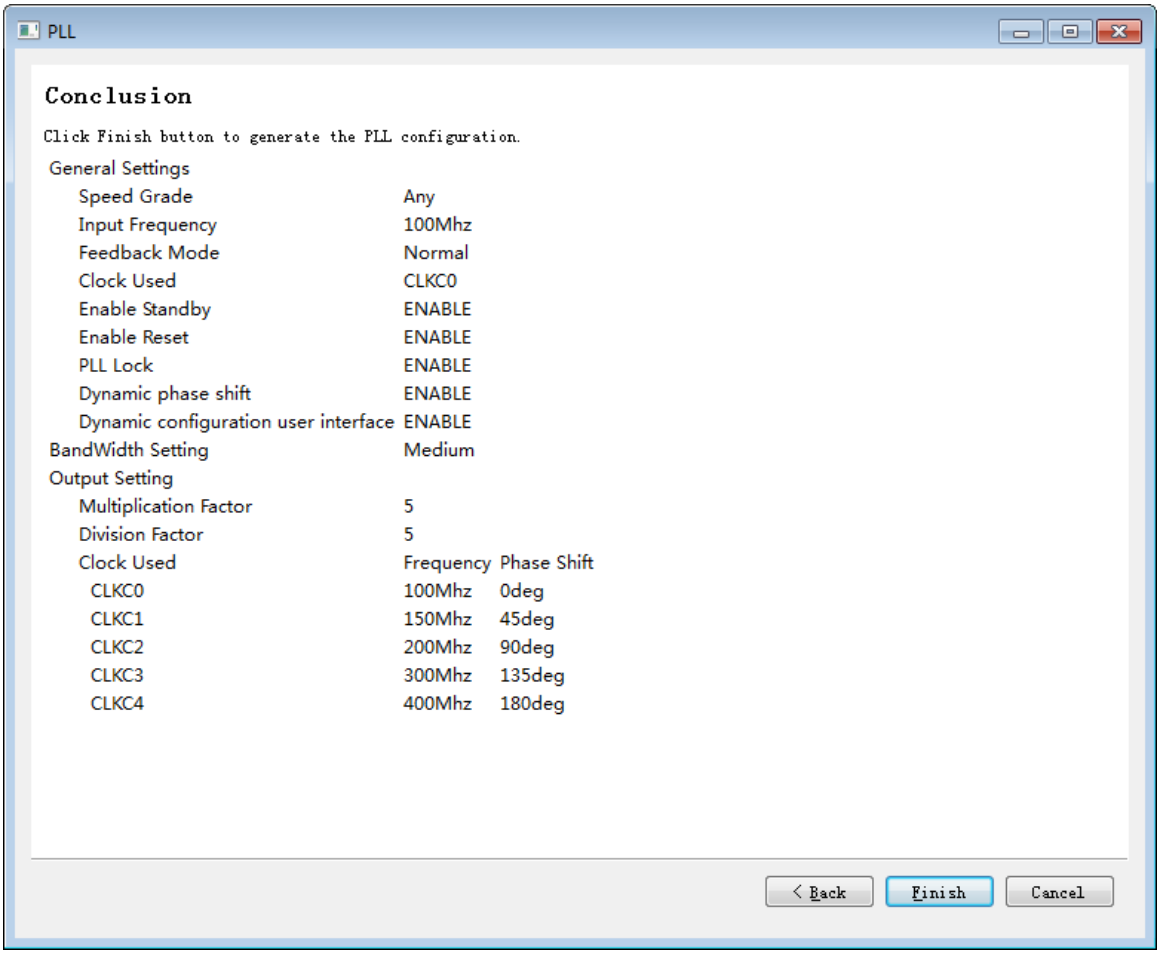
3）输出时钟的设置

每个 PLL 皆有 5 个输出时钟 C0~C4，可根据需求选择输出时钟的数量并配置输出时钟的频率及相位偏移量。设置输出频率时，可在 **Clock frequency** 界面直接设置输出频率，也可根据输入频率，在 **Parameters setting** 界面设置分频系数。点击“**Show Details**”可在右下角查看该输出的各项性能参数值。

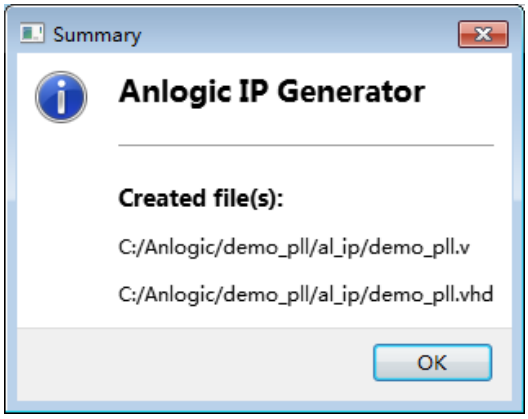

PLL
Output Clock Setting
Clock frequency
Parameters setting
clock division factor 5
clock multiplication factor 5
Use C0 clock Frequency: 100 Mhz Actual Frequency: 100 Mhz Phase Shift: 0 deg Show Details
Use C1 clock Frequency: 150 Mhz Actual Frequency: 150 Mhz Phase Shift: 45 deg Show Details
Use C2 clock Frequency: 200 Mhz Actual Frequency: 200 Mhz Phase Shift: 90 deg Show Details
Use C3 clock Frequency: 300 Mhz Actual Frequency: 300 Mhz Phase Shift: 135 deg Show Details
Use C4 clock Frequency: 400 Mhz Actual Frequency: 400 Mhz Phase Shift: 180 deg Show Details
Global Settings Value
Input frequency(MHz) 100
Primary clock VCO frequency(MHz) 1200
Modulus for M counter 5
Modulus for N counter 5
Detail Settings Value
C0 Settings
Post-scale counter 12
Frequency(Mhz) 100
Initial VCO phase cycles 12
VCO phase tap 0
C1 Settings
C2 Settings
C3 Settings
C4 Settings
< Back
Next >
Cancel


PLL
Output Clock Setting
Clock frequency
Parameters setting
clock division factor 5
clock multiplication factor 5
Use C0 clock division factor: 12 Actual Frequency: 100 Mhz Phase Shift: 0 deg Show Details
Use C1 clock division factor: 8 Actual Frequency: 150 Mhz Phase Shift: 45 deg Show Details
Use C2 clock division factor: 6 Actual Frequency: 200 Mhz Phase Shift: 90 deg Show Details
Use C3 clock division factor: 4 Actual Frequency: 300 Mhz Phase Shift: 135 deg Show Details
Use C4 clock division factor: 3 Actual Frequency: 400 Mhz Phase Shift: 180 deg Show Details
Global Settings Value
Input frequency(MHz) 100
Primary clock VCO frequency(MHz) 1200
Modulus for M counter 5
Modulus for N counter 5
Detail Settings Value
VCO phase tap 0
C1 Settings
Post-scale counter 8
Frequency(Mhz) 150
Initial VCO phase cycles 9
VCO phase tap 0
C2 Settings
C3 Settings
C4 Settings
< Back
Next >
Cancel

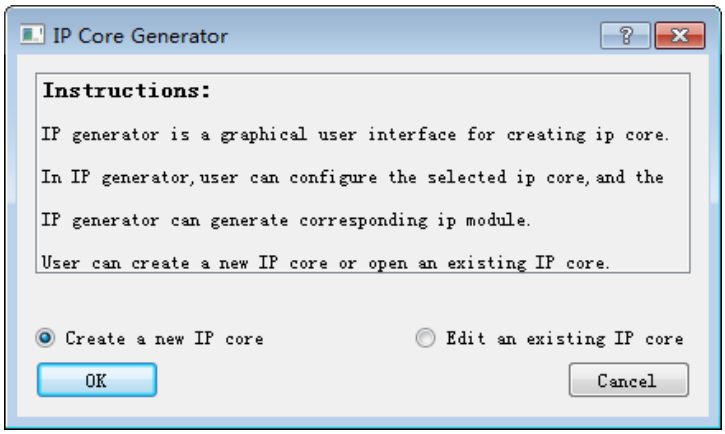
4） 最后确认各项参数是否正确，点击“**Finish**”完成 PLL 的配置。

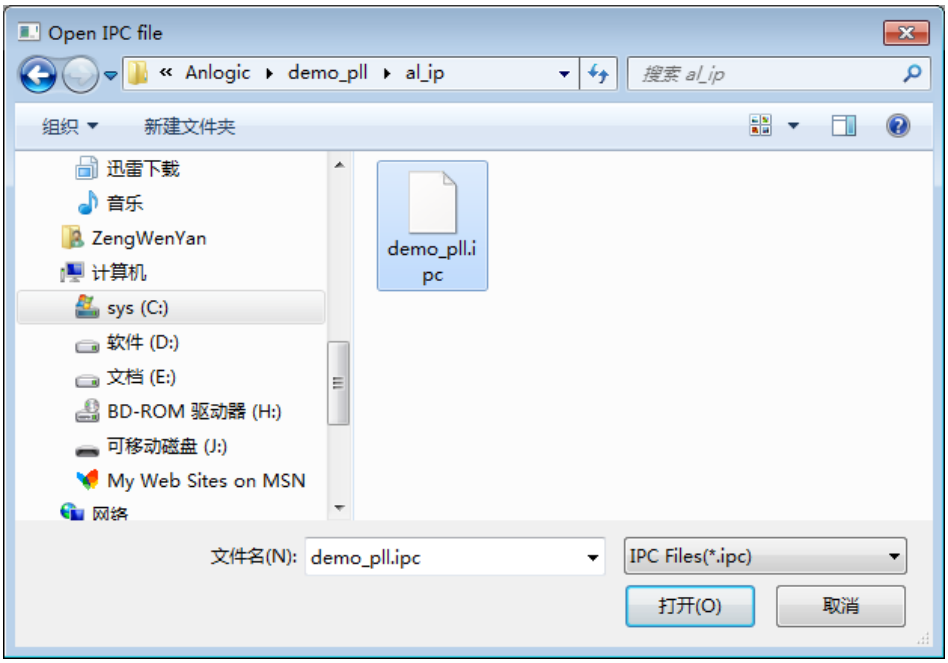


5. TD 将给出生成文件的路径，点击“**OK**”后，可根据提示选择是否将生成的文件添加至工程中。



6. 可通过选择 **Tools → IP Generator**，选择“**Edit an existing IP core**”来打开一个已存在的 IP。


IP Core Generator
Instructions:
IP generator is a graphical user interface for creating ip core.
In IP generator, user can configure the selected ip core, and the
IP generator can generate corresponding ip module.
User can create a new IP core or open an existing IP core.
Create a new IP core
Edit an existing IP core
OK
Cancel


Open IPC file
« Anlogic ▸ demo_pll ▸ al_ip
搜索 al_ip
组织
新建文件夹
迅雷下载
音乐
ZengWenYan
计算机
sys (C:)
软件 (D:)
文档 (E:)
BD-ROM 驱动器 (H:)
可移动磁盘 (J:)
My Web Sites on MSN
网络
demo_pll.ipc
文件名(N): demo_pll.ipc
IPC Files(*.ipc)
打开(O)
取消

3.2.2 例化 PLL 模块

以新建工程为例介绍例化 PLL 模块的过程。用户也可以在已有工程的基础上进行例化，例化过程一致。

1. 新建工程，并为工程添加顶层模块。
2. 在工程中添加上一步生成的 demo_pll.v
3. 在顶层模块中调用 demo_pll 模块，并修改 inst 名称和端口名称，点击保存按钮，即完成了 PLL 模块的例化。点击 **File → Save** 保存文件

top.v*

```
module top (reset,
            stdby,
            extlock,
            clk0_out,
            clk1_out,
            clk2_out,
            clk3_out);

    input reset;
    input stdby;
    output extlock;
    output clk0_out;
    output clk1_out;
    output clk2_out;
    output clk3_out;

    wire osc_clk;

    EG_PHY_OSC EG_PHY_OSC(
      .osc_dis(1'b0),
      .osc_clk(osc_clk)
    );

    demo_pll uut (
          .refclk(osc_clk),
          .reset(reset),
          .stdby(stdby),
          .extlock(extlock),
          .clk0_out(clk0_out),
          .clk1_out(clk1_out),
          .clk2_out(clk2_out),
          .clk3_out(clk3_out));

endmodule
```

demo_pll.v*

```
module demo_pll(refclk,reset,stdby,extlock,
            clk0_out,clk1_out,clk2_out,clk3_out);
    input refclk;
    input reset;
    input stdby;
    output extlock;
    output clk0_out;
    output clk1_out;
    output clk2_out;
    output clk3_out;

    wire clk0_buf;

    EG_LOGIC_BUFG bufg_feedback( .i(clk0_buf), .o(clk0_out) );

    EG_PHY_PLL #(.DYNCFG("DISABLE"),
          .FIN("100.000"),
          .FEEDBK_PATH("CLKC0_EXT"),
          .STDBY_ENABLE("ENABLE"),
          .PLLRST_ENA("ENABLE"),
          .SYNC_ENABLE("ENABLE"),
          .GMC_GAIN(6),
          .ICP_CURRENT(3),
          .KVCO(6),
          .LPF_CAPACITOR(3),
          .LPF_RESISTOR(2),
          .REFCLK_DIV(5),
          .FBCLK_DIV(5),
          .CLKC0_ENABLE("ENABLE"),
          .CLKC0_DIV(12),
          .CLKC0_CPHASE(12),
          .CLKC0_FPHASE(0),
          .CLKC1_ENABLE("ENABLE"),
          .CLKC1_DIV(6),
          .CLKC1_CPHASE(6),
          .CLKC1_FPHASE(6),
          .CLKC2_ENABLE("ENABLE"),
          .CLKC2_DIV(4),
          .CLKC2_CPHASE(5),
          .CLKC2_FPHASE(0),
          .CLKC3_ENABLE("ENABLE"),
          .CLKC3_DIV(3),
          .CLKC3_CPHASE(4),
          .CLKC3_FPHASE(1) )
    pll_inst (.refclk(refclk),
          .reset(reset),
          .stdby(stdby),
          .extlock(extlock),
          .psclk(1'b0),
          .psdown(1'b0),
          .psstep(1'b0),
          .psclksel(3'b000),
          .dclk(1'b0),
          .dcs(1'b0),
          .dwe(1'b0),
          .di(8'b00000000),
          .daddr(6'b000000),
          .fbclk(clk0_out),
          .clkc({open, clk3_out, clk2_out, clk1_out, clk0_buf}));

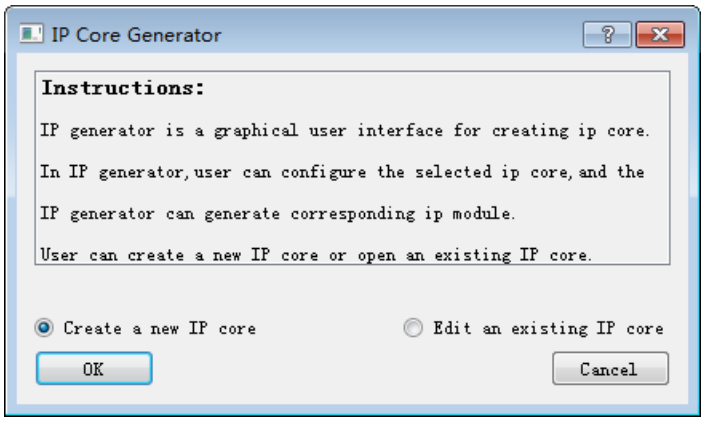
endmodule
```

3.3 DSP 模块

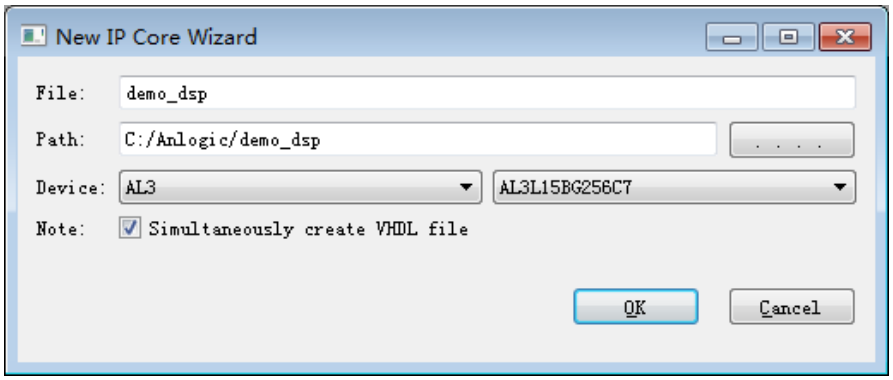
在 AL3 系列器件中，嵌入式乘法器可以配置成一个带输入输出寄存器的 18×18 乘法器，或者配置成两个 9×9 乘法器。

3.3.1 创建 DSP 模块

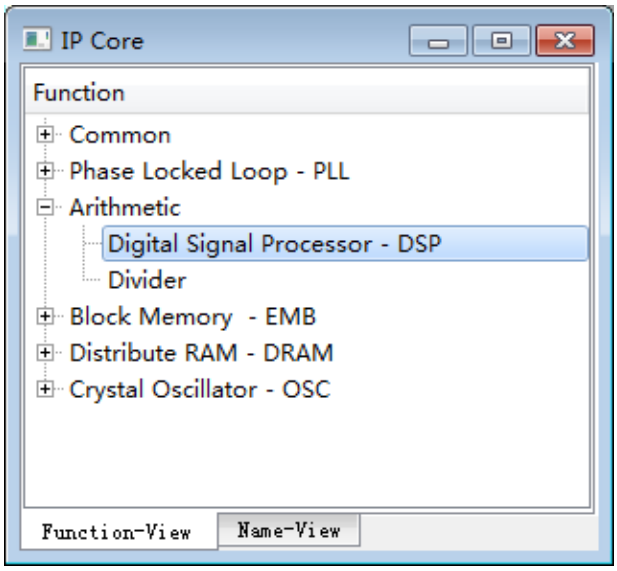
1. 选择 **Tools → IP Generator**，选择“**Create a new IP core**”



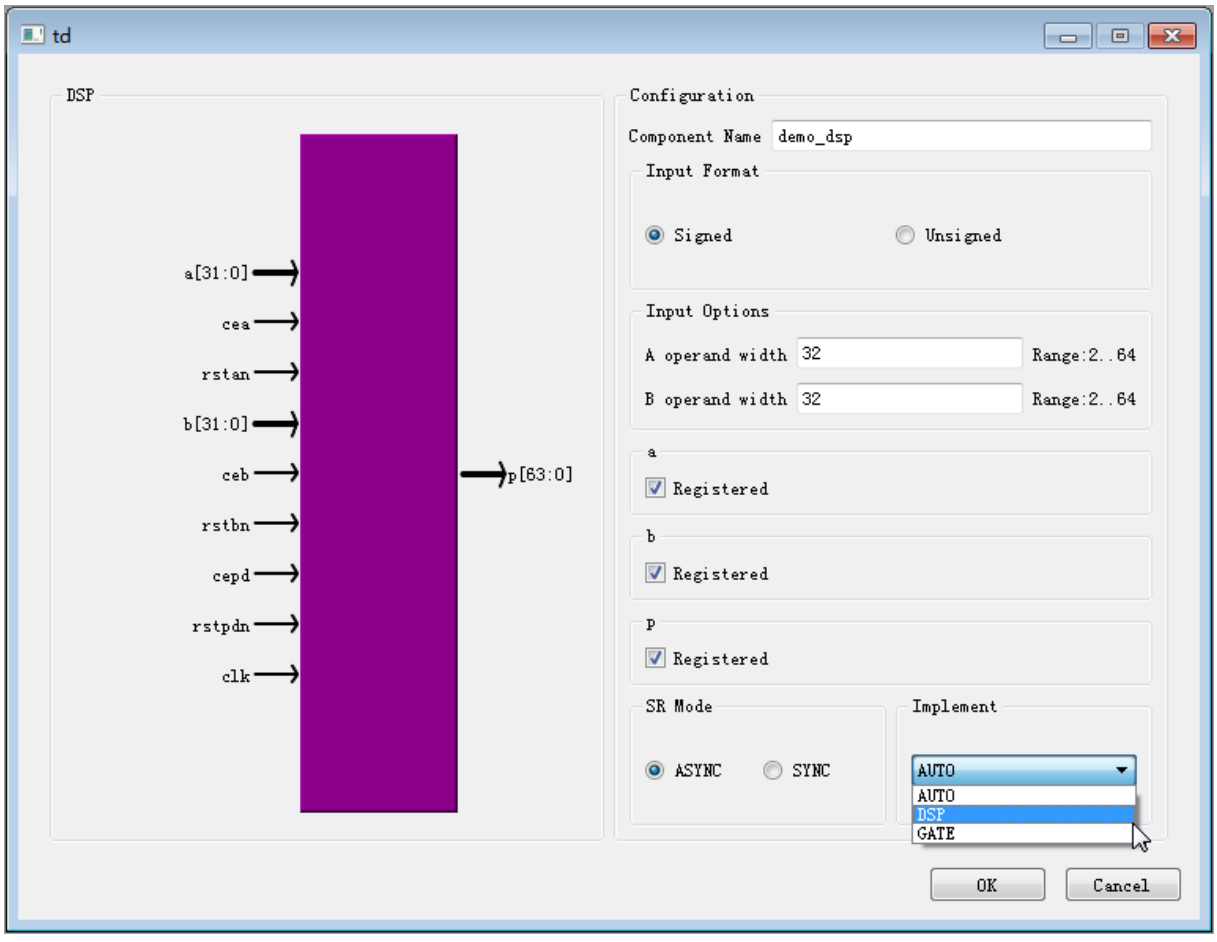
2. 输入模块名称并选择存储路径。此处，若是在有工程的基础上创建 DSP 模块，存储路径和器件名称将与工程保持一致。若在没有工程的基础上创建 DSP 模块，用户需手动设置保存路径和器件名称。若勾选“**Simultaneously create VHDL file**”, TD 将会生成相应的 VHDL 文件。



3. 在 Function 窗口中展开 **Arithmetic→Digital Signal Processor**，双击 **DSP** 打开配置界面



4. 填写 “**Component Name**”并设置相应参数



IP Generator 中用户可自定义乘法运算的实现方式，并提供三个参数供用户选择。其中，**DSP** 表示强制使用硬件 DSP 来实现乘法运算，若 DSP 不够则报错，硬件 DSP 实现的速度要快于使用逻辑门； **GATE** 表示只使用逻辑门来实现乘法运算；**AUTO** 表示优先使用 DSP，若 DSP 不够，则使用逻辑门实现乘法运算。默认参数为：**AUTO**。

AL3 系列器件的嵌入式乘法器均由以下几个单元组成：输入寄存器、乘法器核和输出寄存器。

- 输入寄存器

  根据乘法器的操作模式，可以将每个乘法器输入信号连接到输入寄存器，或直接以 9bit 或 18bit 的形式连接到内部乘法器。可以分别设置乘法器的每个输入是否使用输入寄存器。

  下面的控制信号可用于嵌入式乘法器中的每一个输入寄存器：

  - 时钟 (clk)
  - 时钟使能 (cea / ceb)
  - 同步/异步清零 (rstan / rstbn : n 表示低电平有效)。

  同一个嵌入式乘法器中的所有输入与输出寄存器均由同一时钟信号驱动，时钟使能信号以及异步清零信号驱动可以独立配置。

- 乘法器核

  嵌入式乘法器模块的乘法器既支持 9×9 或 18×18 乘法器，也可实现这些配置位宽之间的其它乘法器。根据乘法器的数据宽度或者操作模式，单一嵌入式乘法器能够同时执行一个或者两个乘法运算。

  乘法器的两个操作数可通过 signed / unsigned 选项来声明为有符号 / 无符号数，以此来确定乘法器的类型。

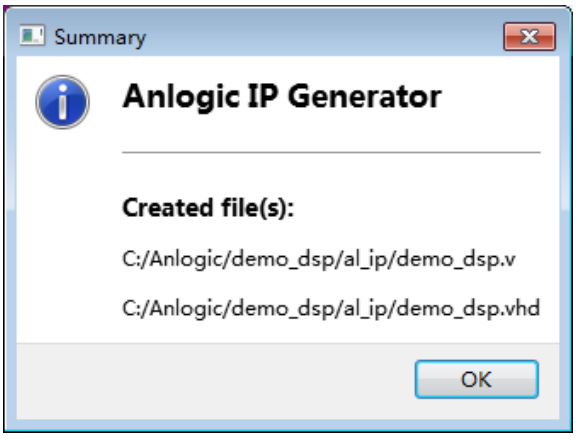
➢ 输出寄存器

根据乘法器的操作模式，可以用 18bit 或 36bit 的形式来使用输出寄存器对嵌入式乘法器的输出进行寄存。下面的控制信号可用于嵌入式乘法器中的每一个输出寄存器：

- 时钟 (clk)
- 时钟使能 (cepd)
- 同步/异步清零 (rstpdn : n 表示低电平有效)

同一个嵌入式乘法器中的所有输入与输出寄存器均由同一时钟信号驱动，时钟使能信号以及异步清零信号驱动可以独立配置。

5. 点击“OK”完成 DSP 的设置，TD 将给出生成文件的路径。



可通过选择 Tools → IP Generator，选择“Edit an existing IP core”来打开一个已存在的 IP。

3.3.2 例化 DSP 模块

本手册以新建工程为例介绍例化 DSP 模块的过程。用户也可以在已有工程的基础上进行例化，例化过程一致。

1. 新建工程，并为工程添加顶层模块。
2. 在工程中添加上一步生成的 demo_dsp.v
3. 在顶层模块中调用 demo_dsp 模块，并修改 inst 名称和端口名称，点击保存按钮，即完成了 DSP 模块的例化。

```
module top ( p, a, b, cea, ceb, cepd,
             clk, rstan, rstbn, rstpdn );

    output [86:0] p;

    input  [63:0] a;
    input  [22:0] b;
    input  cea;
    input  ceb;
    input  cepd;
    input  clk;
    input  rstan;
    input  rstbn;
    input  rstpdn;

    demo_dsp uut(
        .a(a),
        .b(b),
        .p(p),
        .cea(cea),
        .ceb(ceb),
        .cepd(cepd),
        .clk(clk),
        .rstan(rstan),
        .rstbn(rstbn),
        .rstpdn(rstpdn));


endmodule
```

```
module demo_dsp ( p, a, b, cea, ceb, cepd,
                  clk, rstan, rstbn, rstpdn );

    output [86:0] p;

    input  [63:0] a;
    input  [22:0] b;
    input  cea;
    input  ceb;
    input  cepd;
    input  clk;
    input  rstan;
    input  rstbn;
    input  rstpdn;

    AL_LOGIC_MULT #( .INPUT_WIDTH_A(64),
                     .INPUT_WIDTH_B(23),
                     .OUTPUT_WIDTH(87),
                     .INPUTFORMAT("SIGNED"),
                     .INPUTREGA("ENABLE"),
                     .INPUTREGB("ENABLE"),
                     .OUTPUTREG("ENABLE"),
                     .IMPLEMENT("AUTO"),
                     .SRMODE("ASYNC"))
                inst(
                     .a(a),
                     .b(b),
                     .p(p),
                     .cea(cea),
                     .ceb(ceb),
                     .cepd(cepd),
                     .clk(clk),
                     .rstan(rstan),
                     .rstbn(rstbn),
                     .rstpdn(rstpdn));


endmodule
```

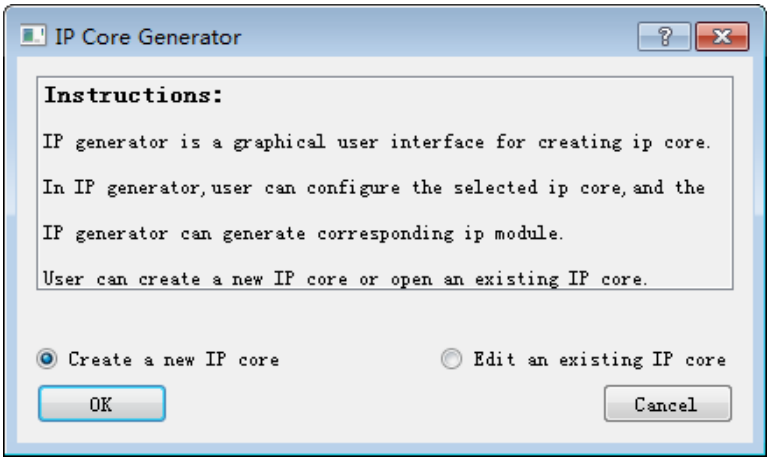
```
Console
Utilization Statistics
#le                          1329    out of  8640 15.38%
  #lut                       1329    out of  8640 15.38%
  #reg                          0    out of  8640  0.00%
#dsp                            3    out of     3   100%
#bram                           0    out of    48     0%
 #bram9k                        0
 #fifo9k                        0
Console  Error  Warning
```

3.4 Divider 模块

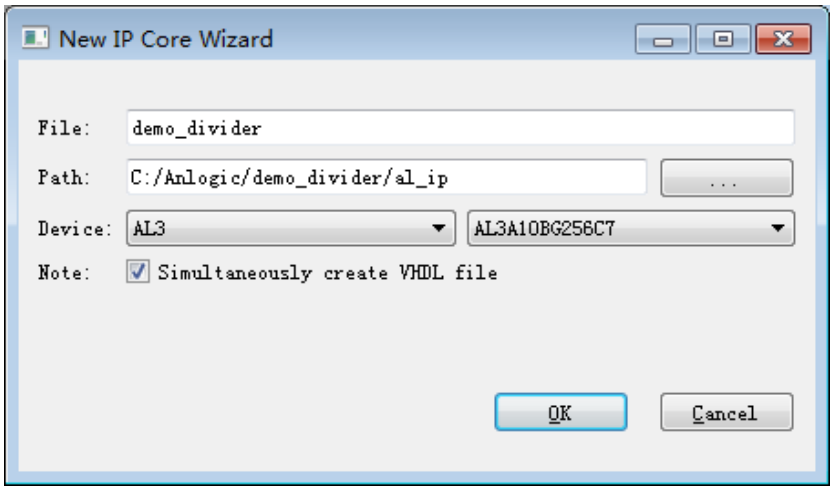
TD 软件实现了基于时钟驱动的除法器。

3.4.1 创建 Divider 模块

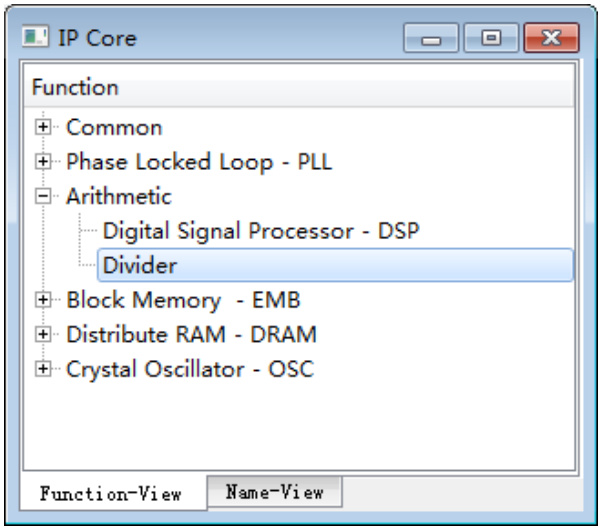
1. 选择 **Tools → IP Generator**，选择“**Create a new IP core**”。



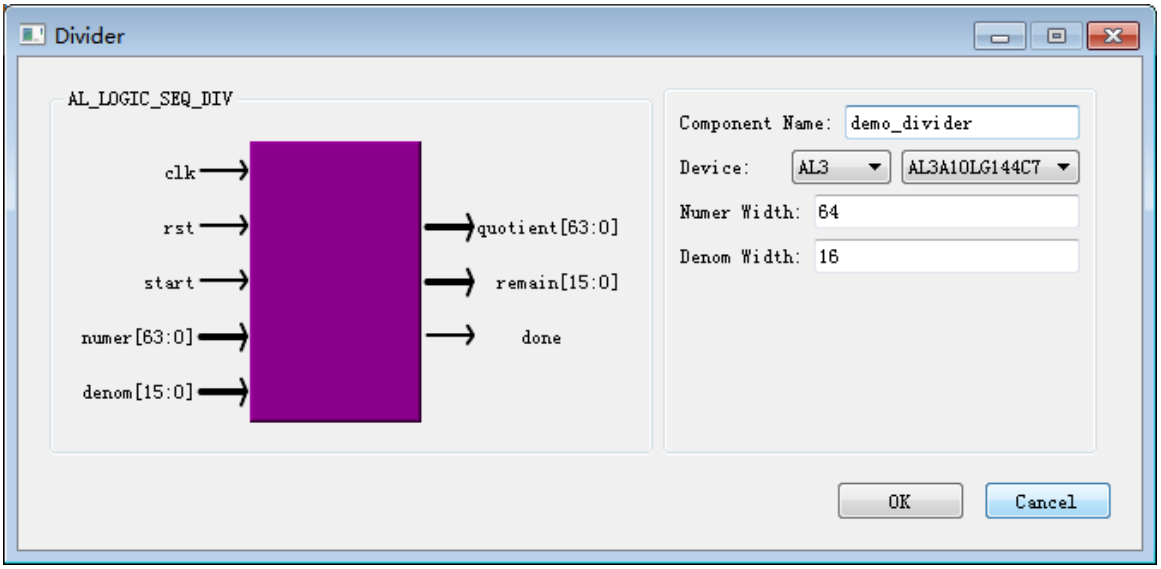
2. 输入模块名称并选择存储路径。此处，若是在有工程的基础上创建 Divider 模块，存储路径和器件名称将与工程保持一致。若在没有工程的基础上创建 Divider 模块，用户需手动设置保存路径和器件名称。若勾选“**Simultaneously create VHDL file**”, TD 将会生成相应的 VHDL 文件。



3. 在 Function 窗口中展开 **Arithmetic→Divider**，双击 **Divider** 打开配置界面。



4. 填写 **Component Name** 及相应的操作数。



其中，

1) Numer Width 为被除数的位宽，同时也作为商的位宽 ；

2) Denom Width 为除数的位宽，同时也作为余数的位宽 ；

3) clk 为驱动计算使用的时钟 ；

4) rst 为复位信号。当 rst 为 1 时，内部寄存器和输出（quotient，remain）会置 0，而 done 会置 1；

5) start 为计算的启动信号。当 start 设置为 1 时，会将输入数据缓存到内部寄存

器中；而当 start 由 1 变成 0 后，计算过程才真正开始；

6) numer 为被除数。虽然 start 为 0 时，改变 numer 的值，不会影响计算（原来的值已经在 start 为 1 的 clock 上升沿缓存到内部寄存器）。但是，为了防止波形显示时产生误解，请在 done 为 1（或者等待计算过程所需要的 clock 周期数）后，才会其提供下一个计算的输入；

7) denom 为除数。注意事项同 numer；

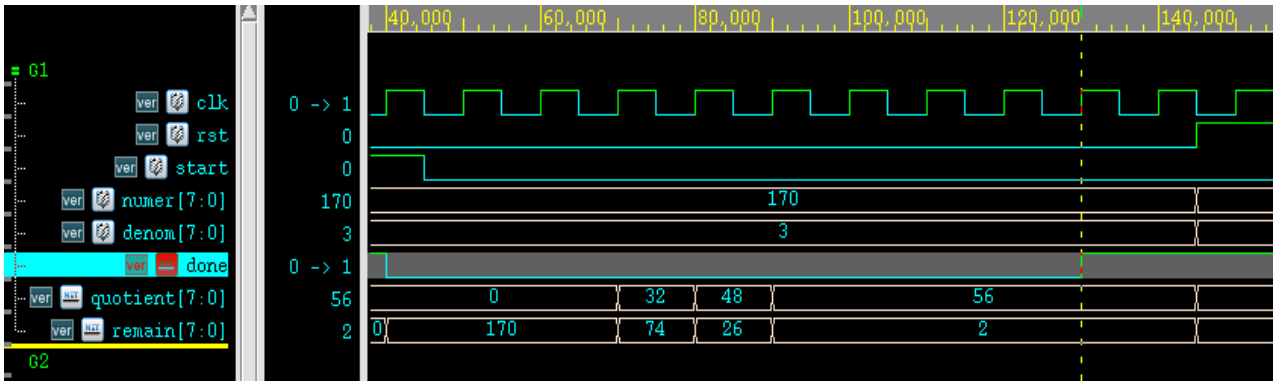
8) quotient 为商。只有当 done 为 1 时，quotient 的值才是计算出的最终结果；

9) remain 为余数。只有当 done 为 1 时，remain 的值才是计算出的最终结果；

10) done 为计算完成的信号。当 done 设置为 1 时，表示计算已完成。为保证输出结果的正确性，需要在 done 信号为 1，才使用输出值 quotient 和 remain；同时也需要在 done 信号为 1 后，才提供下一组输入值。

仿真波形如下所示：

numer = 170，denom = 3，当 start 由 1 变为 0 时，开始计算，直到 done = 1 时，得出 quotient = 56，remain = 2。



5. 点击“OK”完成 Divider 的设置，TD 将给出生成文件的路径。

可通过选择 Tools → IP Generator，选择“Edit an existing IP core”来打开一个已存在的 IP。

3.4.2 例化 Divider 模块

本手册以新建工程为例介绍例化 Divider 模块的过程。用户也可以在已有工程的基础上进行例化，例化过程一致。

1. 新建工程，并为工程添加顶层模块。
2. 在工程中添加上一步生成的 demo_divider.v
3. 在顶层模块中调用 demo_divider 模块，并修改 inst 名称和端口名称，点击保存按钮，即完成了 Divider 模块的例化。

top.v

```
1  module top (clk, rst, start, numer, denom,
2                  quotient, remain, done );
3
4  input clk;
5  input rst;
6  input start;
7  input [63:0] numer;
8  input [15:0] denom;
9
10 output [63:0] quotient;
11 output [15:0] remain;
12 output  done;
13
14 demo_divider divider (
15         .clk(clk),
16         .rst(rst),
17         .start(start),
18         .numer(numer),
19         .denom(denom),
20         .quotient(quotient),
21         .remain(remain),
22         .done(done));
23
24 endmodule
```

demo_divider.v*

```
11
12 `timescale 1ns / 1ps
13
14 module demo_divider ( clk, rst, start, numer,
15                       denom, quotient, remain, done );
16     parameter NUMER_WIDTH = 64;
17     parameter DENOM_WIDTH = 16;
18     input   clk;
19     input   rst;
20     input   start;
21     input   [NUMER_WIDTH-1:0]   numer;
22     input   [DENOM_WIDTH-1:0]   denom;
23     output  [NUMER_WIDTH-1:0]   quotient;
24     output  [DENOM_WIDTH-1:0]   remain;
25     output  done;
26
27     AL_LOGIC_SEQ_DIV #(
28         .NUMER_WIDTH(NUMER_WIDTH),
29         .DENOM_WIDTH(DENOM_WIDTH)
30     )
31     seq_div (
32         .clk(clk),
33         .rst(rst),
34         .start(start),
35         .numer(numer),
36         .denom(denom),
37         .quotient(quotient),
38         .remain(remain),
39         .done(done));
40 endmodule
```

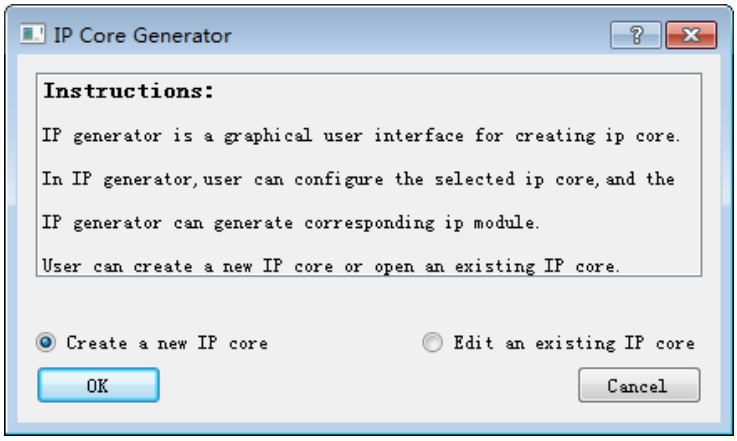
3.5 BRAM 模块

AL3 系列器件支持嵌入式存储器模块（Embedded Memory Block）。AL3-10 中包括两类 EMB：EMB9K 和 EMB32K。

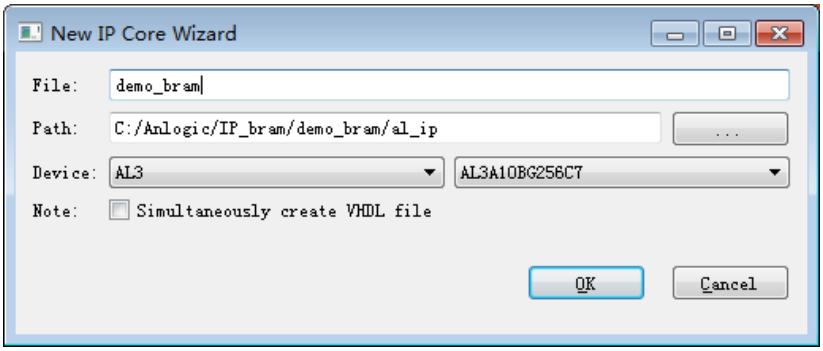
EMB9K 每块容量 9Kbits，多个 EMB9K 模块排成一列，按列分布在可编程功能块(Programmable Function Block, PFB)的阵列中。EMB32K 每块容量 32Kbits，分布在 IO 空隙中。

3.5.1 创建 BRAM 模块

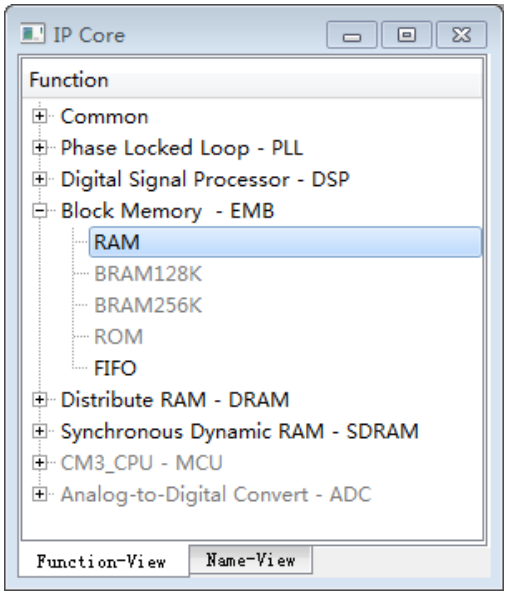
3. 选择 **Tools → IP Generator**，选择“**Create a new IP core**”



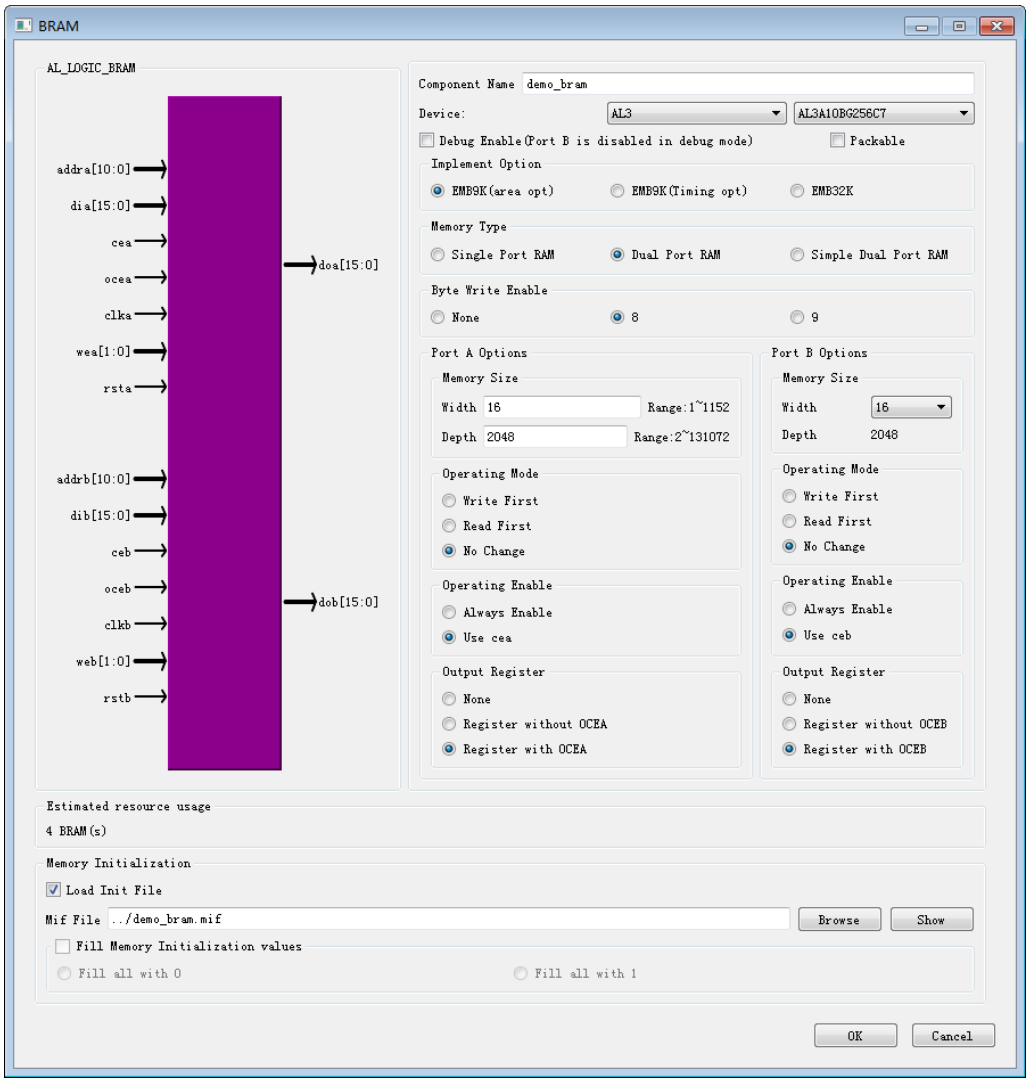
4. 输入模块名称并选择存储路径。此处，若是在有工程的基础上创建 BRAM 模块，存储路径和器件名称将与工程保持一致。若在没有工程的基础上创建 BRAM 模块，用户需手动设置保存路径和器件名称。若勾选“**Simultaneously create VHDL file**”, TD 将会生成相应的 VHDL 文件。



5. 在 Function 窗口中展开 **Block Memory**，双击 **RAM** 打开配置界面



6. 填写“**Component Name**”并设置相应参数



本手册以 EMB9K 为例介绍 AL3 系列器件 BRAM 模块的使用。

EMB9K 可实现：

- 单口 RAM (Single Port RAM)
- 双口 RAM (Dual Port RAM)
- 简单双口 RAM（Simple Dual RAM，也称为伪双口）

EMB9K 模块支持的功能特色有：

- 9216 （9K）bits / 每块
- A/B 口时钟独立
- 可单独配置 A/B 口数据位宽，真双口从 x1 到 x9，支持 x18 简单双口（一写一读）
- 9 或 18 位写操作时带有字节使能（Byte Enable）控制
- 输出锁存器可选择（支持 1 级流水线）
- 支持 RAM 模式下数据初始化（通过初始化文件在配置过程中对 EMB9K 进行数据初始化）
- 支持多种写操作模式。可选择只写（No Change），先读后写（Read First），先写后读（Write First）三种模式
- 支持 Byte Enable 功能。

若勾选“**Debug Enable**”前面的复选框，TD 会默认 EMB 的模式为 Single Port RAM，在这种情况下，端口 B 将被占用，端口 A 的数据可进行回读，方便用户通过 BramEditor 进行 Debug。其中，EMB9k 以面积优化为主，EMB9k(fast)以时序优化为主。

Byte Enable 是指 BRAM 的输入数据 port 位宽为多个 byte 时，在读数据时用一组 byte enable 信号来分别控制每个 byte 写入与否。在界面上可选择 **Byte Write Enable** 的值

为 None 或 8 或 9。当 byte-write 为 None 时，表示不启动 byte enable 功能；当 byte-write 为 8 时，A 口与 B 口（若有 B 口）的数据宽度必须为 8 的整数倍，倍数的值被用作 wea 与 web 的宽度；当 byte-write 为 9 时，A 口与 B 口（若有 B 口）的数据宽度必须为 9 的倍数，倍数被用作 wea 与 web 的宽度。当启动 byte enable 功能时，不建议使用 BRAM32K，原因是当 BRAM 的深度比较小时，会浪费很多内存。

7. 添加初始化文件

TD 的初始化文件支持用户用第三方 mif(memory initialization file)格式描述，或者用 verilog 存储空间初始化 dat 格式来描述。

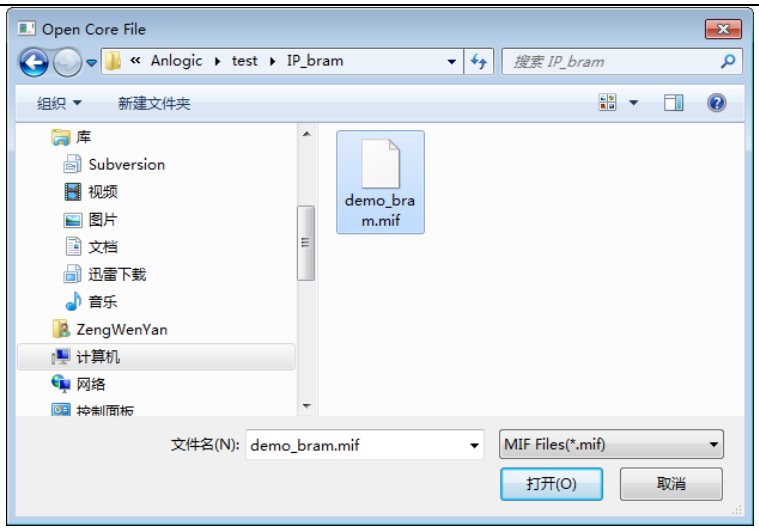
- mif 格式描述如下：

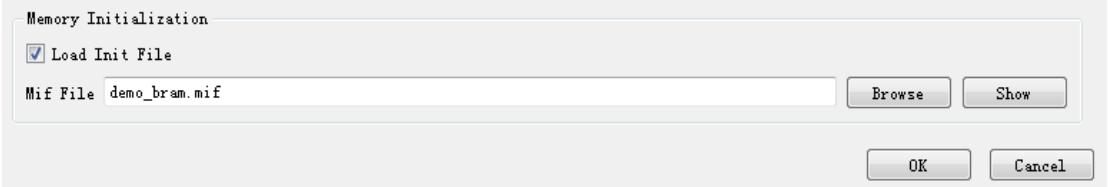
  mif 格式的初始化文件包含每一个初始化地址和数据，并且必须定义内存数据的深度和宽度。用户可以将数据和地址格式定义为二进制 BIN 、 十六进制 HEX 、 八进制 OCT 、 无符号十进制 UNS 等。数据的值必须和数据格式相匹配。

- dat 格式描述如下：

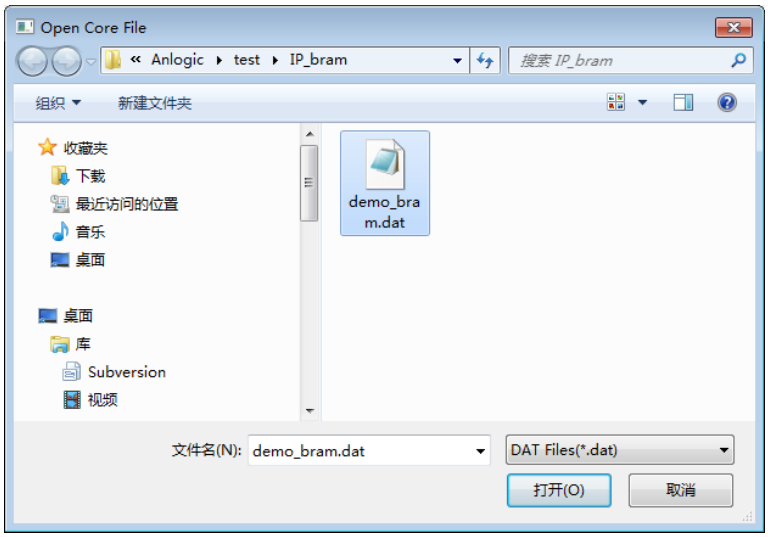
  内存数据可以存储在一个以十六进制为地址的文件中，其中，地址以 “@”表示。起始地址由用户自己定义，根据内存数据的深度可以相应的确定结束地址。为了能够使数据和地址清晰对应，通常会给文件添加可识别的地址标志。若初始化文件很大时，也可直接省略地址。

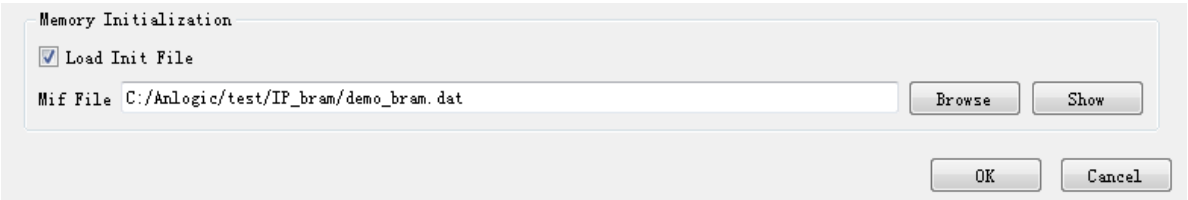
为 BRAM 模块添加初始化文件时，可勾选“**Load Init File**”前面的复选框，并选择需要添加的文件，当在右下角的下拉框中选择.mif 格式时，在文件夹中只提供.mif 文件供用户选择，如下图所示。



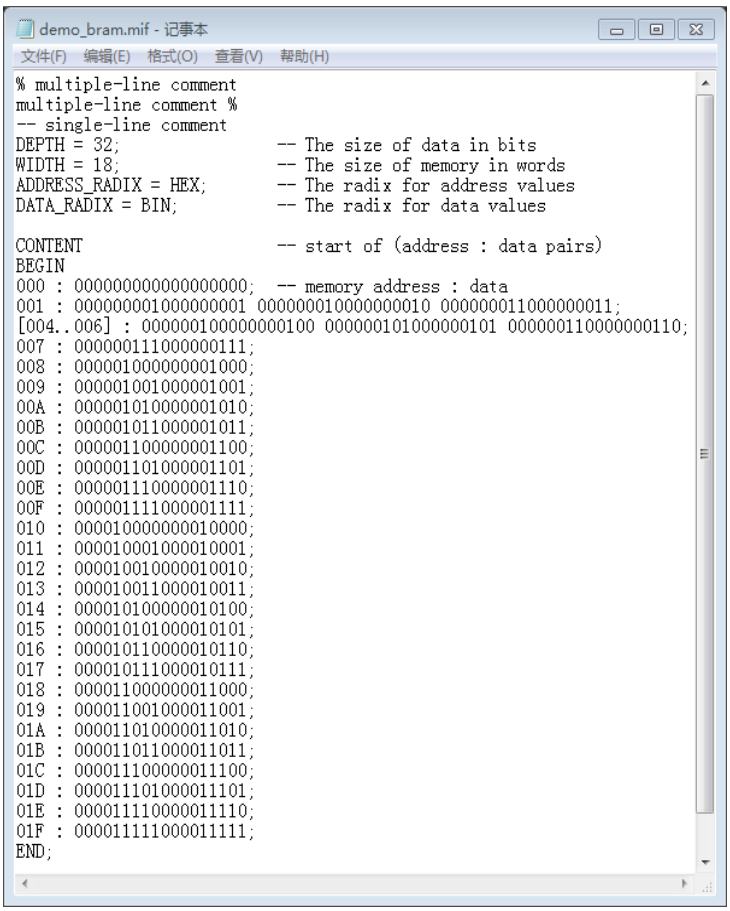


若选择.dat 格式时，在文件夹中则只提供.dat 文件供用户选择，如下图所示。

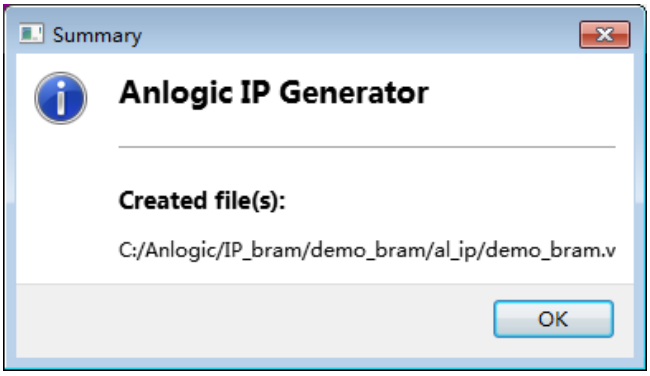




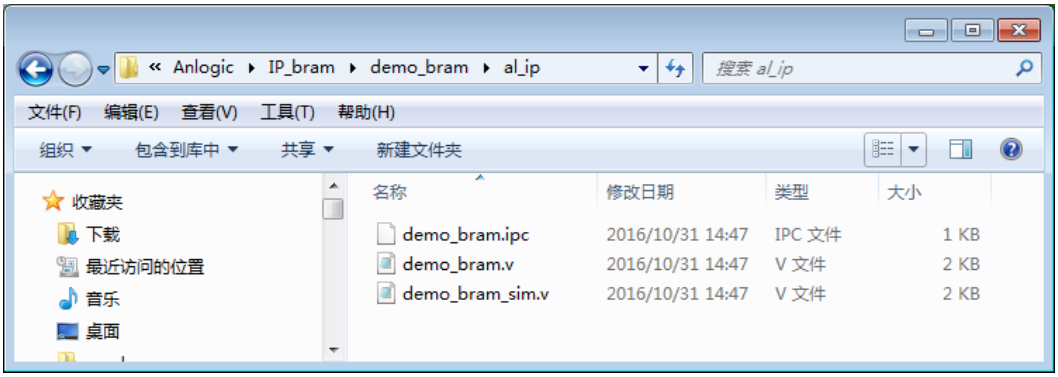
点击按钮“**Show**”可以看到添加文件的内容，此处给出的是 mif 文件和 dat 文件的格式范例。



点击“OK”完成 BRAM 的创建，TD 给出生成文件的路径如下：

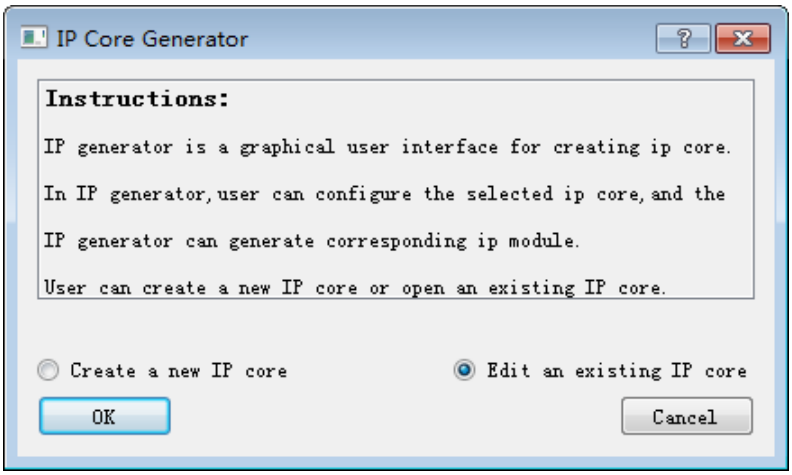


而在工程目录下可以看到如下文件：



其中，demo_bram.ipc 为 IP Generator 生成的工程文件，可通过如下方式打开：

选择 **Tools → IP Generator**，选择“**Edit an exist IP core**”。



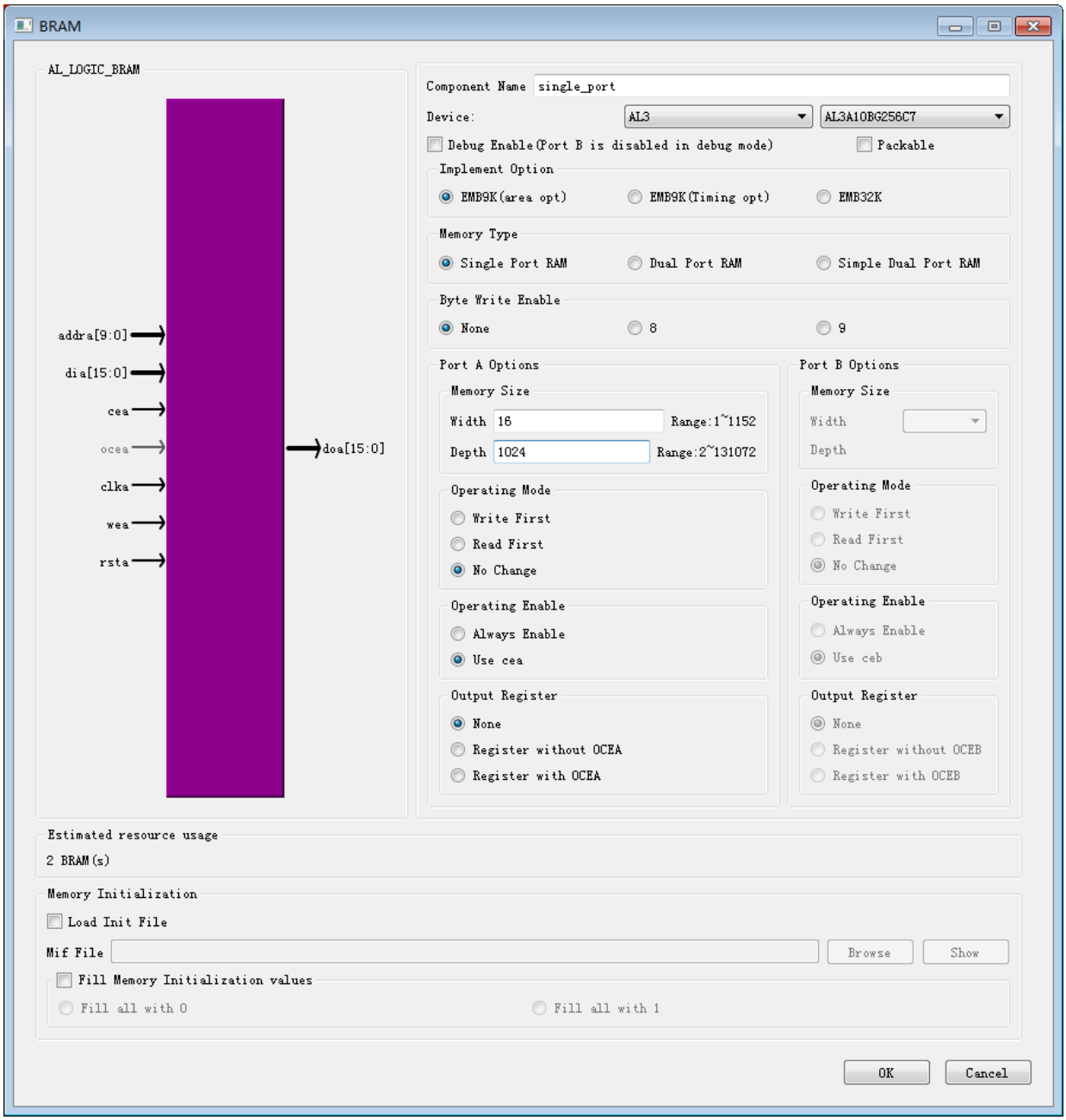
demo_bram.v 为 BRAM 模块创建供用户进行例化的文件。

demo_bram_sim.v 描述了 BRAM 的划分方式，供用户仿真和 Debug，但不可将其添加为工程的源文件，否则将与 demo_bram.v 中的模块产生冲突。

下面分别介绍 BRAM 三种不同模式的界面设置及生成文件的范例：

1. 单口模式(Single Port RAM)

单口模式支持对非同时发生的对同一地址的读或写操作。界面设置如下：



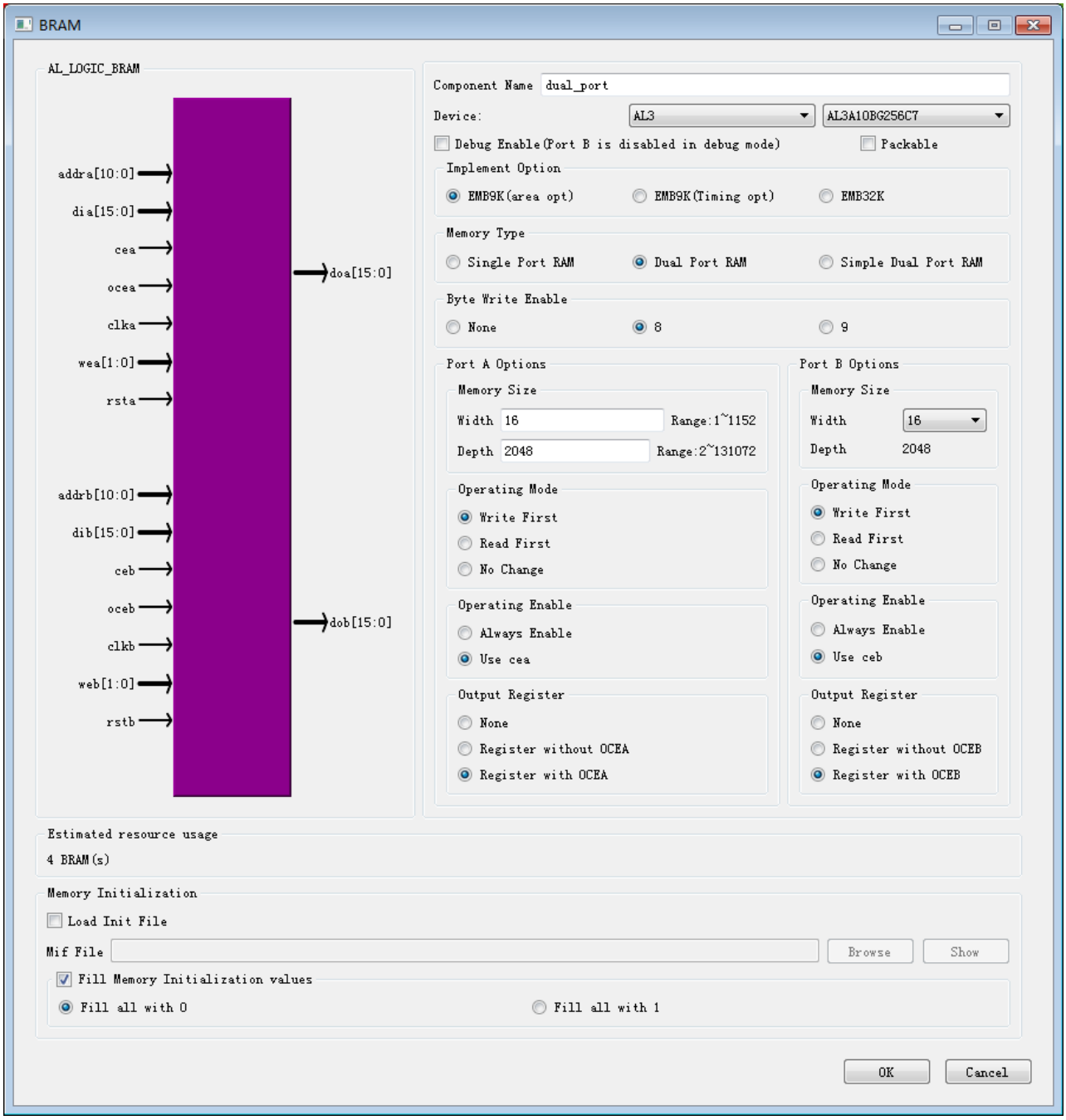
单口模式下生成的文件如下所示：

```
10  \*************************************************************/
11
12  `timescale 1ns / 1ps
13
14  module single_port ( doa, dia, addra, cea, clka, wea, rsta );
15
16      output [15:0] doa;
17
18      input  [15:0] dia;
19      input  [9:0] addra;
20      input  wea;
21      input  cea;
22      input  clka;
23      input  rsta;
24
25      AL_LOGIC_BRAM #( .DATA_WIDTH_A(16),
26                       .ADDR_WIDTH_A(10),
27                       .DATA_DEPTH_A(1024),
28                       .DATA_WIDTH_B(16),
29                       .ADDR_WIDTH_B(10),
30                       .DATA_DEPTH_B(1024),
31                       .MODE("SP"),
32                       .REGMODE_A("NOREG"),
33                       .WRITEMODE_A("NORMAL"),
34                       .RESETMODE("SYNC"),
35                       .IMPLEMENT("9K"),
36                       .DEBUGGABLE("NO"),
37                       .PACKABLE("NO"),
38                       .INIT_FILE("NONE"),
39                       .FILL_ALL("NONE"))
40                  inst(
41                       .dia(dia),
42                       .dib({16{1'b0}}),
43                       .addra(addra),
44                       .addrb({10{1'b0}}),
45                       .cea(cea),
46                       .ceb(1'b0),
47                       .ocea(1'b0),
48                       .oceb(1'b0),
49                       .clka(clka),
50                       .clkb(1'b0),
51                       .wea(wea),
52                       .web(1'b0),
53                       .bea(1'b0),
54                       .beb(1'b0),
55                       .rsta(rsta),
56                       .rstb(1'b0),
57                       .doa(doa),
58                       .dob());
59
60  endmodule
```

2. 双口模式(Dual Port RAM)

双口模式支持 A 口 / B 口的所有独立读写操作组合：两读，两写，一读和一写。

界面设置如下：



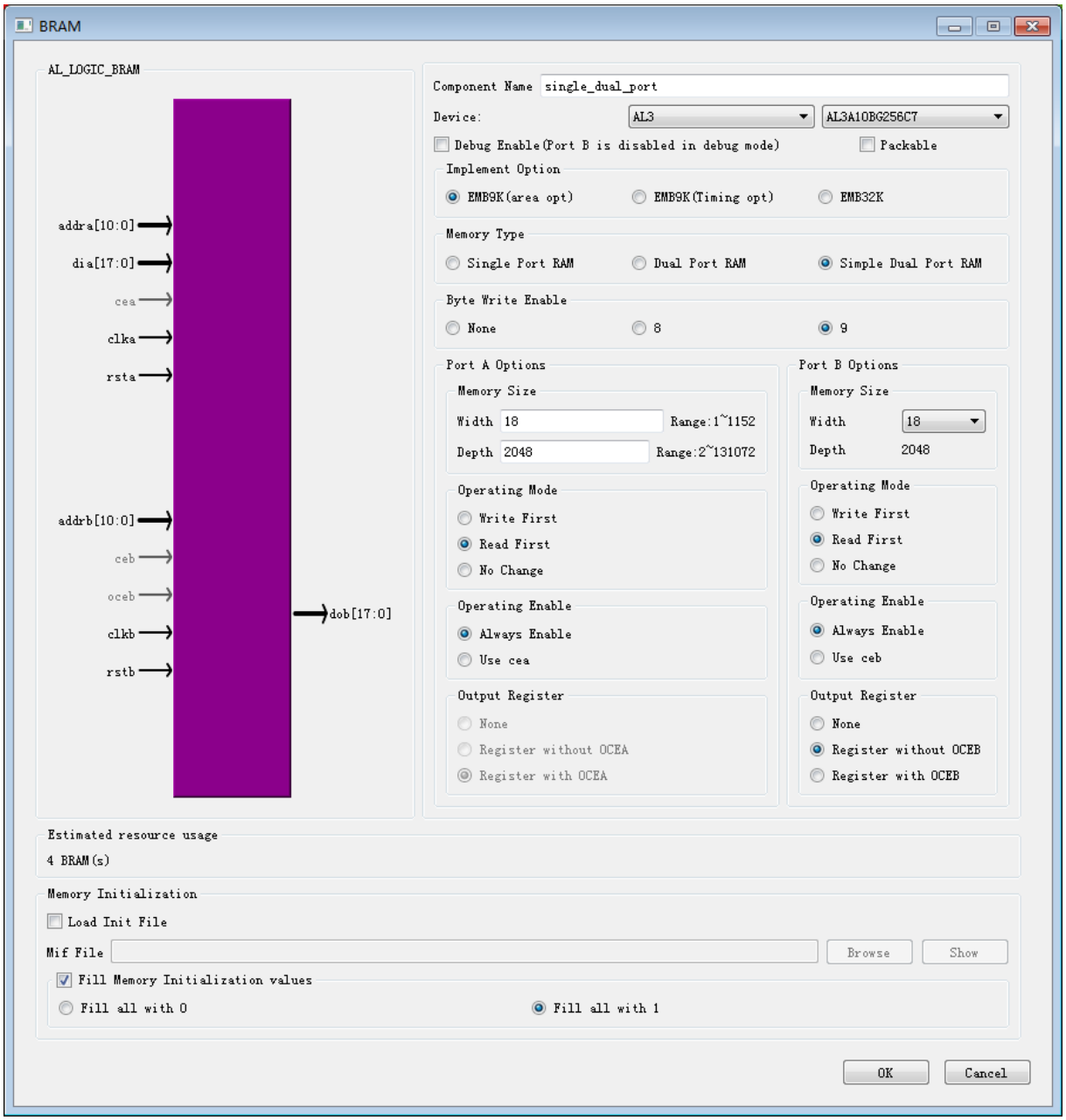
双口模式下生成的文件如下所示：

```
13
14  module dual_port (
15      doa, dia, addra, cea, clka, wea, rsta, ocea,
16      dob, dib, addrb, ceb, clkb, web, rstb, oceb
17  );
18      output [15:0] doa;
19      output [15:0] dob;
20      input  [15:0] dia;
21      input  [15:0] dib;
22      input  [10:0] addra;
23      input  [10:0] addrb;
24      input  [1:0] wea;
25      input  [1:0] web;
26      input  cea;
27      input  ceb;
28      input  clka;
29      input  clkb;
30      input  rsta;
31      input  rstb;
32      input  ocea;
33      input  oceb;
34
35      AL_LOGIC_BRAM #( .DATA_WIDTH_A(16),
36                       .DATA_WIDTH_B(16),
37                       .ADDR_WIDTH_A(11),
38                       .ADDR_WIDTH_B(11),
39                       .DATA_DEPTH_A(2048),
40                       .DATA_DEPTH_B(2048),
41                       .BYTE_ENABLE(8),
42                       .BYTE_A(2),
43                       .BYTE_B(2),
44                       .MODE("DP"),
45                       .REGMODE_A("OUTREG"),
46                       .REGMODE_B("OUTREG"),
47                       .WRITEMODE_A("WRITETHROUGH"),
48                       .WRITEMODE_B("WRITETHROUGH"),
49                       .RESETMODE("SYNC"),
50                       .IMPLEMENT("9K"),
51                       .INIT_FILE("NONE"),
52                       .FILL_ALL("0"))
53                 inst(
54                       .dia(dia),
55                       .dib(dib),
56                       .addra(addra),
57                       .addrb(addrb),
58                       .cea(cea),
59                       .ceb(ceb),
60                       .ocea(ocea),
61                       .oceb(oceb),
62                       .clka(clka),
63                       .clkb(clkb),
64                       .wea(1'b0),
```

3. 简单双口模式(Simple Dual Port RAM)

将一块 EMB9K 配置成 18 位写入或 18 位读出时，该 EMB9K 不支持双口模式，但支持单口和简单双口模式。当 EMB9K 为 18 位模式时，A 端口控制信号作为写入控制信号，B 端口控制信号作为读出控制信号。当 EMB9K 配置为 18 位写入时，dib[8:0]作为高 9 位数据输入，dia[8:0]作为低 9 位数据输入；当 EMB9K 配置为 18 位读出时，dob[8:0]作为高 9 位数据输出，doa[8:0]作为低 9 位数据输出。

简单双口模式的界面设置如下:



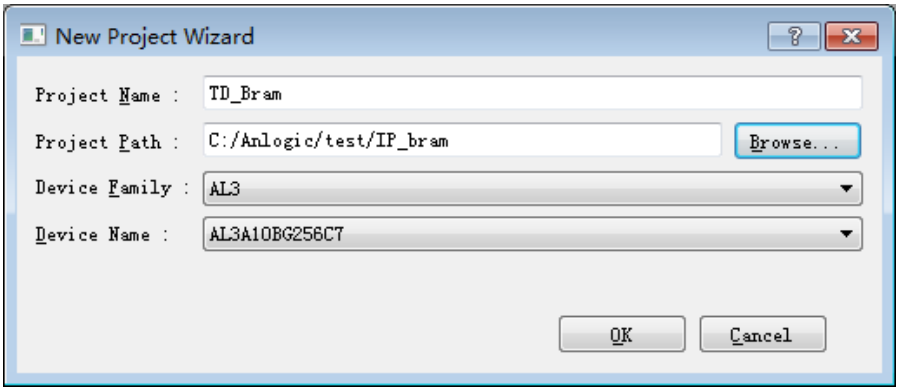
简单双口模式下生成的文件如下所示：

```
13
14  module single_dual_port (
15      dia, addra, clka, rsta, wea,
16      dob, addrb, clkb, rstb
17  );
18      output [17:0] dob;
19      input  [17:0] dia;
20      input  [10:0] addra;
21      input  [10:0] addrb;
22      input  [1:0] wea;
23      input  clka;
24      input  clkb;
25      input  rsta;
26      input  rstb;
27
28      AL_LOGIC_BRAM #( .DATA_WIDTH_A(18),
29                       .DATA_WIDTH_B(18),
30                       .ADDR_WIDTH_A(11),
31                       .ADDR_WIDTH_B(11),
32                       .DATA_DEPTH_A(2048),
33                       .DATA_DEPTH_B(2048),
34                       .BYTE_ENABLE(9),
35                       .BYTE_A(2),
36                       .BYTE_B(2),
37                       .MODE("PDPW"),
38                       .REGMODE_A("OUTREG"),
39                       .REGMODE_B("OUTREG"),
40                       .WRITEMODE_A("READBEFOREWRITE"),
41                       .WRITEMODE_B("READBEFOREWRITE"),
42                       .RESETMODE("SYNC"),
43                       .IMPLEMENT("9K"),
44                       .INIT_FILE("NONE"),
45                       .FILL_ALL("1"))
46                 inst(
47                       .dia(dia),
48                       .dib({18{1'b0}}),
49                       .addra(addra),
50                       .addrb(addrb),
51                       .cea(1'b1),
52                       .ceb(1'b1),
53                       .ocea(1'b0),
54                       .oceb(1'b1),
55                       .clka(clka),
56                       .clkb(clkb),
57                       .wea(1'b0),
58                       .bea(wea),
59                       .rsta(rsta),
60                       .rstb(rstb),
61                       .doa(),
62                       .dob(dob));
63  endmodule
```

3.5.2 例化 BRAM 模块

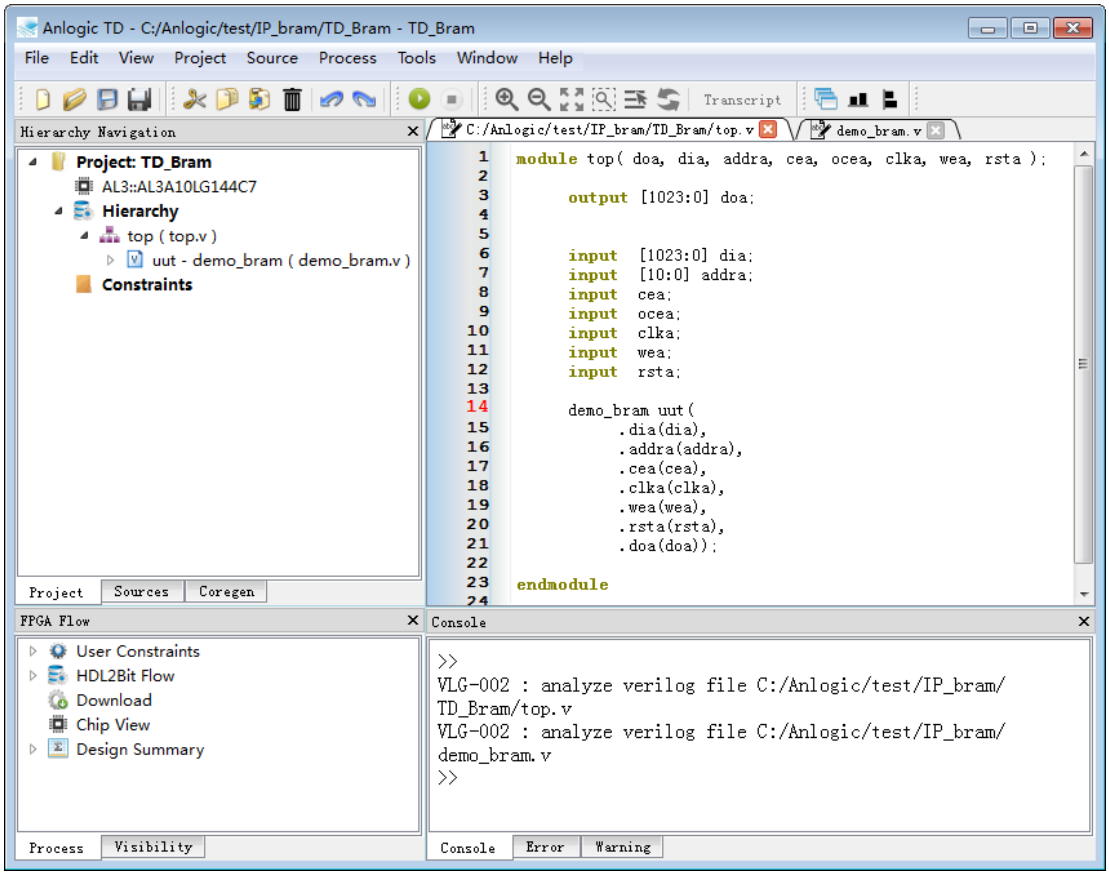
本手册以新建工程为例介绍例化 BRAM 模块的过程。用户也可以在已有工程的基础上进行例化，例化过程一致。

1. 新建工程，并为工程添加顶层模块。



2. 在工程中添加上一步生成的 demo_bram.v

3. 在顶层模块中调用 demo_bram 模块，并修改 inst 名称和端口名称，点击保存按钮，即完成了 BRAM 模块的例化。

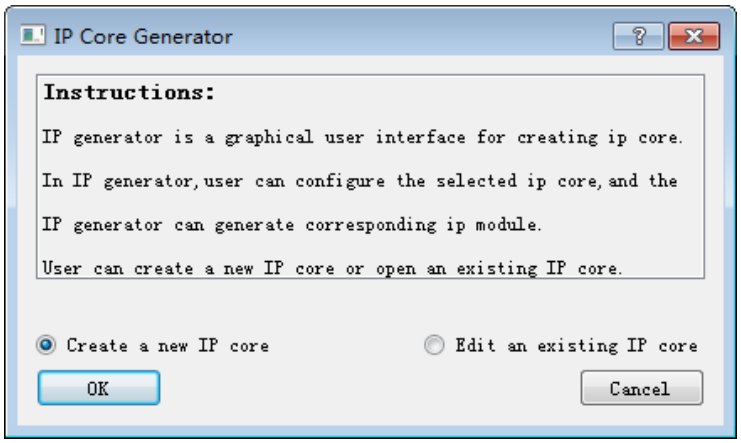


3.6 FIFO 模块

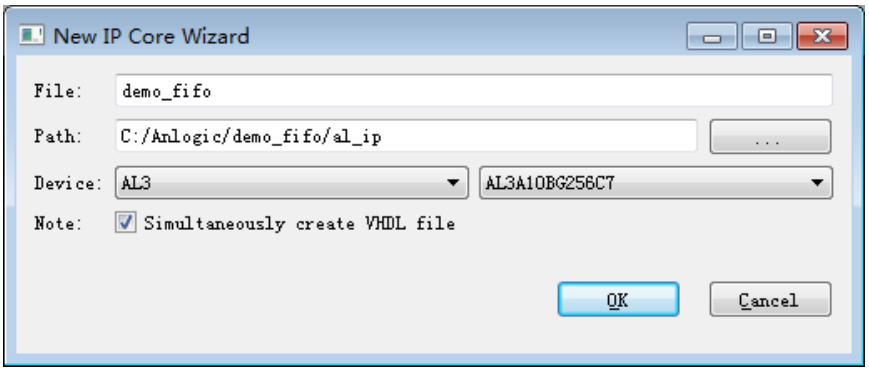
EMB9k 内部硬件支持同步/异步 FIFO 模式。FIFO 模式下 EMB9K 位宽设置和简单双口 RAM 设置相同，最高可支持 18bit 位宽。

3.6.1 创建 FIFO 模块

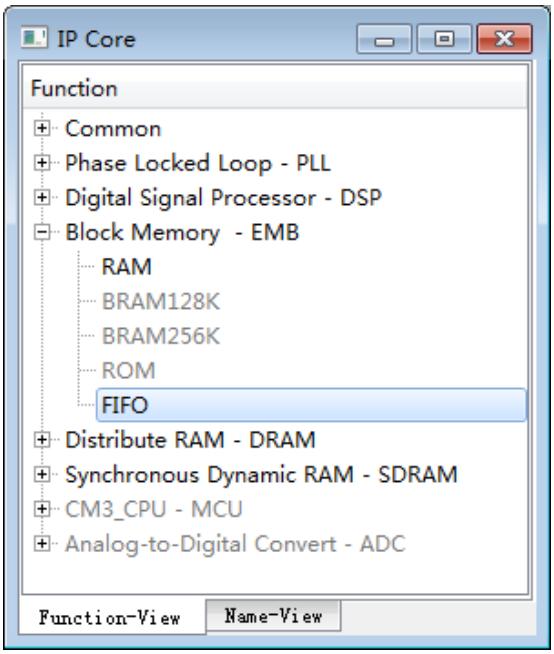
1. 选择 **Tools → IP Generator**，选择“**Create a new IP core**”



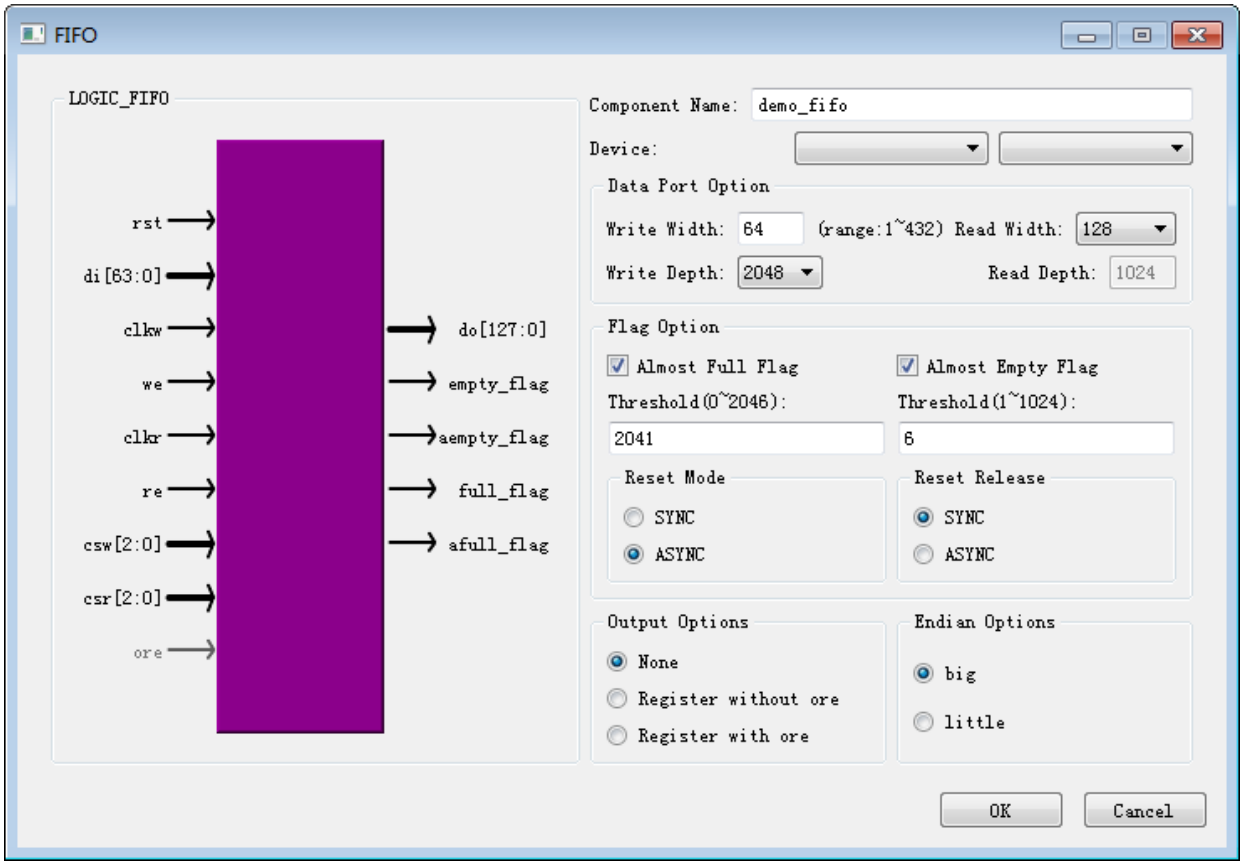
2. 输入模块名称并选择存储路径。此处，若是在有工程的基础上创建 FIFO 模块，存储路径和器件名称将与工程保持一致。若在没有工程的基础上创建 FIFO 模块，用户需手动设置保存路径和器件名称。若勾选“**Simultaneously create VHDL file**”, TD 将会生成相应的 VHDL 文件。



3. 在 Function 窗口中展开 **Block Memory**，双击 **FIFO** 打开配置界面



4. 填写“**Component Name**”并设置相应参数



Endian Options 为输出数据的大小端模式，具体体现为:

Endian Options : big 模式下:

当写入的数据为:　AA,BB,CC,DD,　则读出为 BB_AA,DD_CC

当写入的数据为:　BB_AA,DD_CC, 则读出为 AA,BB,CC,DD

Endian Options: little 模式下:

当写入的数据为:　AA,BB,CC,DD, 则读出为 AA_BB, CC_DD

当写入的数据为:　AA_BB,CC_DD, 则读出为 AA,BB,CC,DD

FIFO 模式下的空满标志说明:

FIFO 标志名	方向	设置范围	说明
Empty_Flag (EF)	输出	0	FIFO 读空标志，和 clkr 同步
Aempty_Flag (AE)	输出	1 to FF-1	FIFO 几乎空标志，和 clkr 同步， 相对读空提前量由 AE_POINTER 参数决定
Full_Flag (FF)	输入	1 to Max	FIFO 满标志，和 clkr 同步， FIFO 满容量由 FULL_POINTER 参数决定
Afull_Flag (AF)	输入	1 to FF-1	FIFO 几乎满标志，和 clkw 同步， FIFO 几乎满容量由 AF_POINTER 参数决定

- 空满标志的设置

FIFO 模式下，用户可以通过软件设置 FIFO 空满标志属性：空标志(Empty_Flag)，几乎空标志(Almost_Empty)，满标志(Full_Flag)，几乎满标志(Almost_Full)。当内部计数器计数到标志值时，相应的空满指针 EF/AE/FF/AF 所对应的端口会输出高电平。

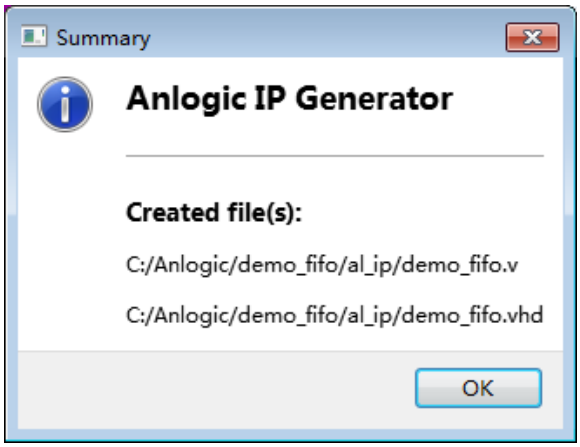
- 空满标志指针的说明

FIFO 模式下空指针(EF)固定为 0，AE/AF/FF 指针为 14 位二进制数"0b0X_XXXX_XXXX_XXXX"，最高位[13]始终为 0，有效位[12:0]，代表 8K 深度 1bit 宽度。指针最低 4 位[3:0]是否有效取决于读出/写入的位宽。

- 利用 csw/csr 反向配置实现简单 FIFO

csw 为 FIFO 写端口的 3 位片选信号，可反向；csr 为 FIFO 读端口的 3 位片选信号，可反向。 当 FIFO 写满或读空时为了避免指针溢出，可以通过互联资源将满信号反向后接入 csw 端，空信号反向后接入 csr 端。反向逻辑可以利用 csw/csr 内部的反向与逻辑实现。

5. 点击“OK”完成 FIFO 的设置，TD 将给出生成文件的路径。

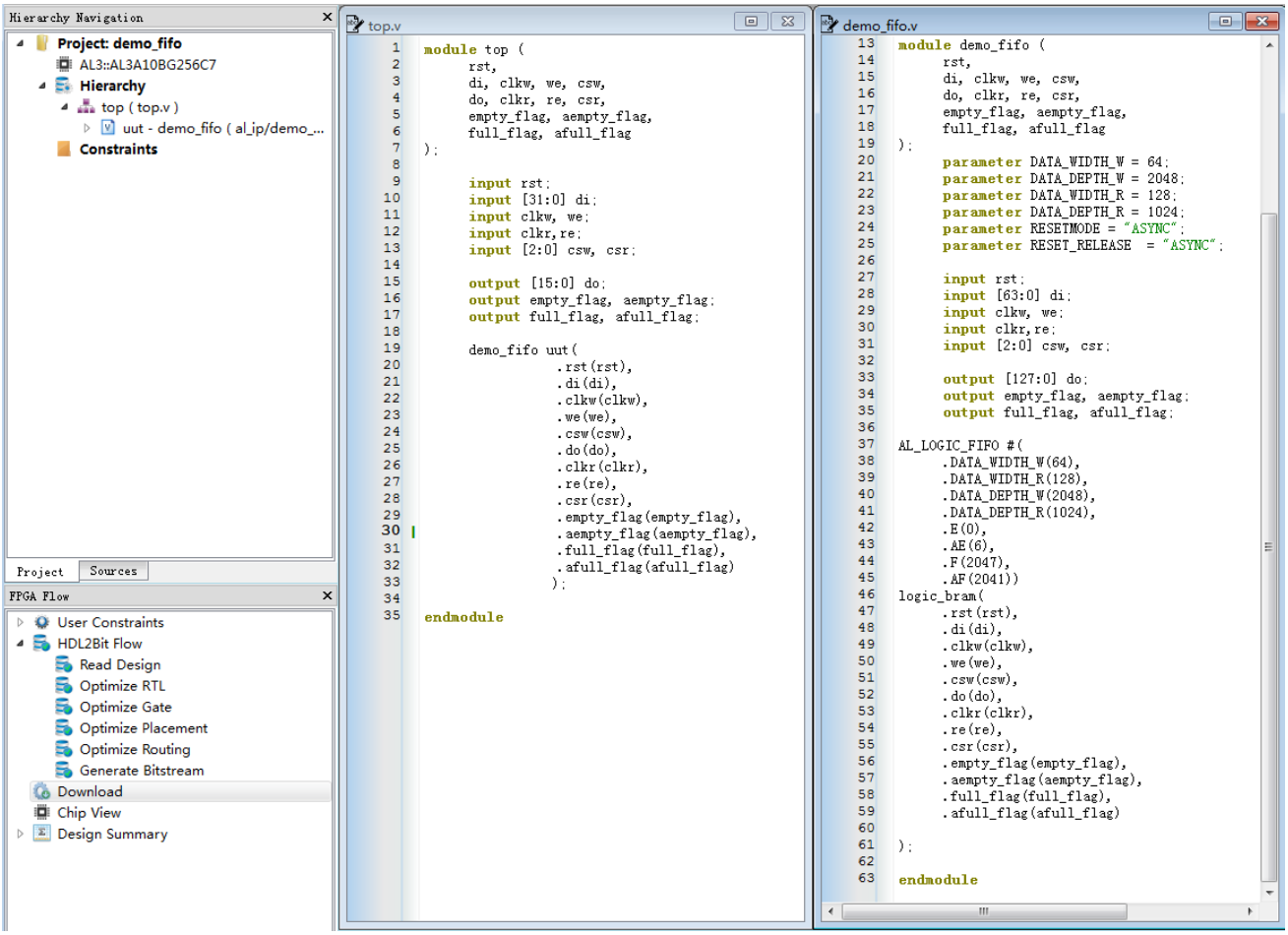


同样可通过“**Edit an existing IP core**”方式来打开并编辑已存在的 demo_fifo.ipc。

3.6.2 例化 FIFO 模块

该手册以新建工程为例介绍例化 FIFO 模块的过程。用户也可以在已有工程的基础上进行例化，例化过程一致。

1. 新建工程，并为工程添加顶层模块。
2. 在工程中添加上一步生成的 demo_fifo.v
3. 在顶层模块中调用 demo_fifo 模块，并修改 inst 名称和端口名称，点击保存按钮，即完成了 FIFO 模块的例化。

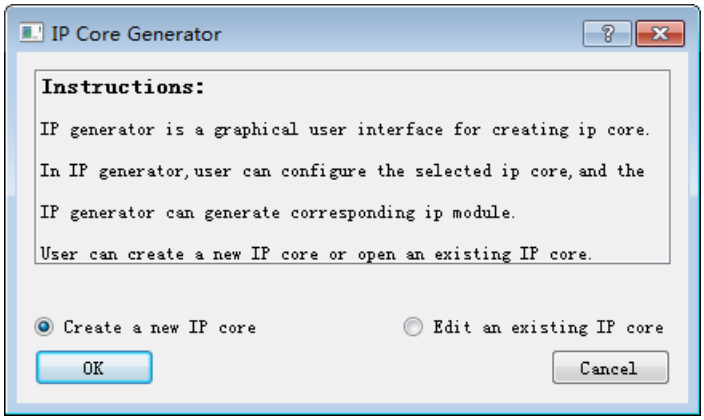


3.7 RAMFIFO

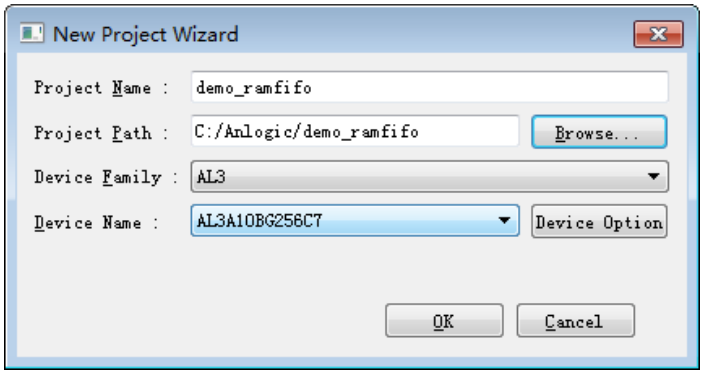
RAMFIFO 模块是一种基于片上 RAM 存储资源的异步 FIFO 模块。模块支持异步时钟读写、异步复位功能、空/满信号指示功能、读写数据个数指示功能、SHOWAHEAD 功能，具有良好的时序特性和合理的资源消耗。

3.7.1 创建 RAMFIFO 模块

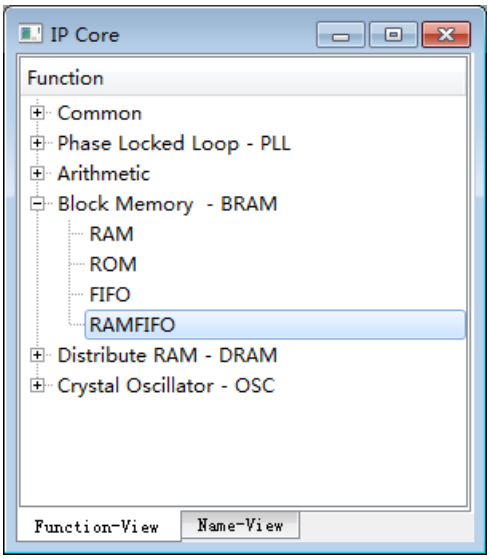
1. 选择 Tools → IP Generator，选择“Create a new IP core”



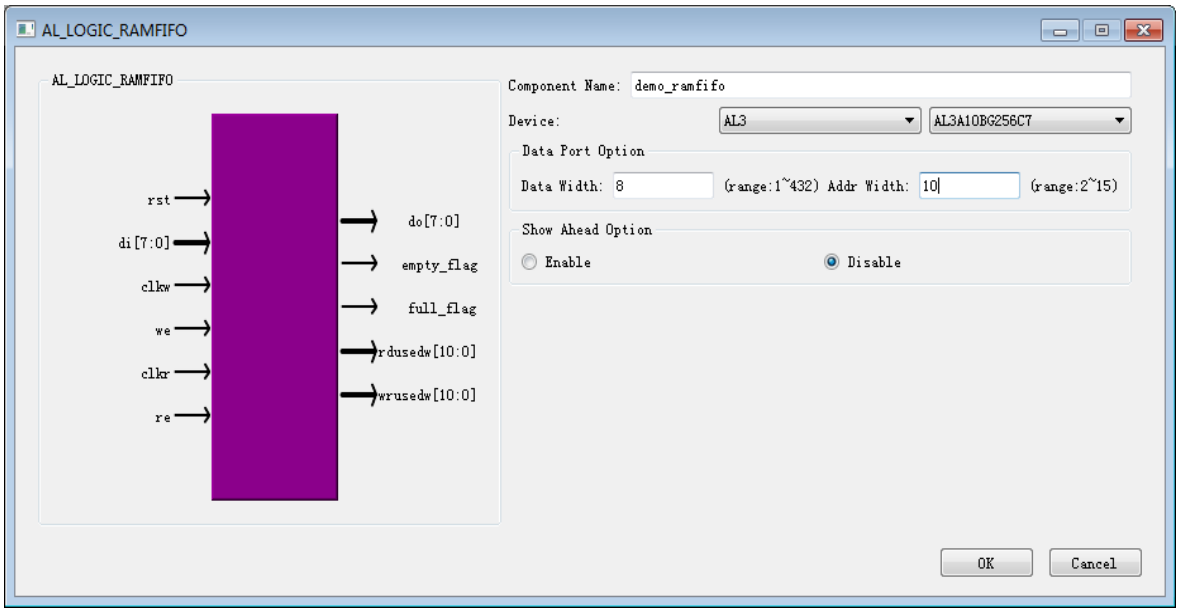
2. 输入模块名称并选择存储路径。此处，若是在有工程的基础上创建 RAMFIFO 模块，存储路径和器件名称将与工程保持一致。若在没有工程的基础上创建 RAMFIFO 模块，用户需手动设置保存路径和器件名称。若勾选 “**Simultaneously create VHDL file**”, TD 将会生成相应的 VHDL 文件。



3. 在 Function 窗口中展开 **Block Memory -- BRAM**，双击 **RAMFIFO** 打开配置界面



4. 填写“**Component Name**”并设置相应参数



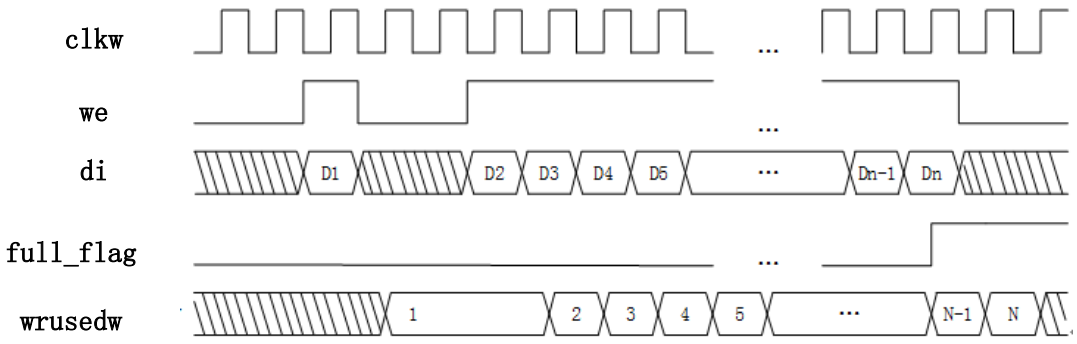
参数定义如下表所示：

参数	说明
DATA_WIDTH	读写 FIFO 的数据位宽
ADDR_WIDTH	读写 FIFO 的地址位宽，FIFO 的深度为 2 的 ADDR_WIDTH 次方
SHOWAHEAD	SHOWAHEAD 模式使能，1 表示 SHOWAHEAD 模式，0 表示普通模式

端口定义如下表所示：

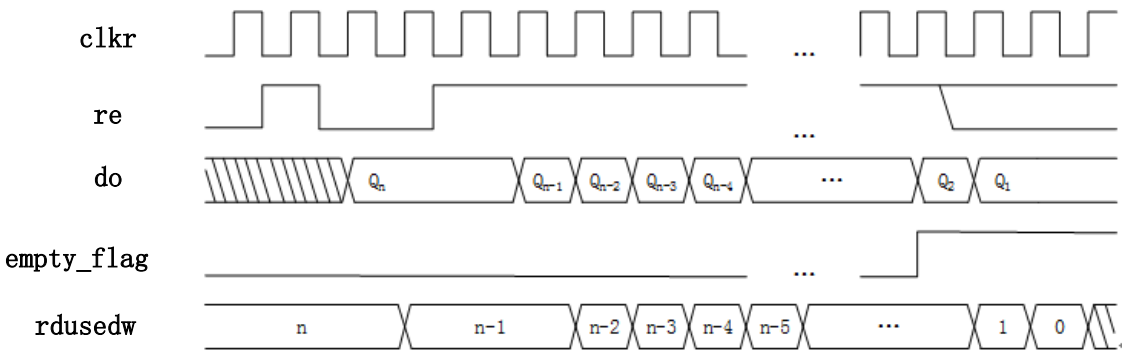
端口	功能	说明
rst	异步复位信号	高电平有效，用于复位整个模块
di	数据输入	位宽由参数 DATA_WIDTH 决定
clkw	写时钟信号	为写端提供同步时钟
we	写使能信号	高电平，且 FIFO 不为满时将数据写入 FIFO
clkr	读时钟信号	为读端提供同步时钟
re	读使能信号	高电平，且 FIFO 不为空时将数据从 FIFO 读出
do	数据输出	位宽由参数 DATA_WIDTH 决定
empty_flag	空指示信号	高电平时表示 FIFO 已空，将忽略读使能信号
full_flag	满指示信号	高电平时表示 FIFO 已满，将忽略写使能信号
rdusedw	可读数据个数	位宽由参数 ADDR_WIDTH 决定
wrusedw	已写数据个数	位宽由参数 ADDR_WIDTH 决定

写时序图：



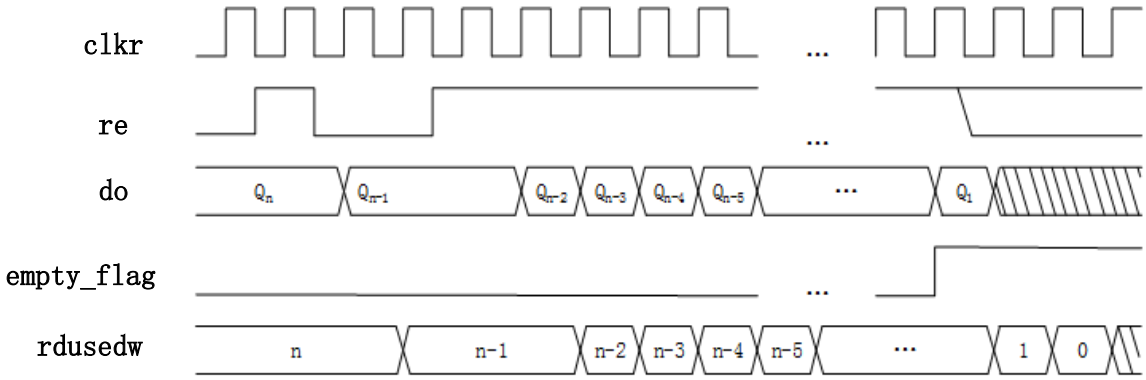
在时钟上升沿时，如果 we 为高且 full_flag 为低，则将此时的 di 写入 FIFO，完成一次写操作，同时写数据个数指示信号 wrusedw 自加 1（wrusedw 如时序图所示，延迟一个时钟周期输出），若干时钟延时后 rdusedw 自加 1（具体延时与读写时钟周期有关）。

普通模式读时序图：



在时钟上升沿时，如果 re 为高且 empty_flag 为低，则在下一个时钟周期读出 FIFO 数据到 do，完成一次读操作，同时读数据个数指示信号 rdusedw 自减 1（rdusedw 如时序图所示，延迟一个时钟周期输出），若干时钟延时后 wrusedw 自减 1（具体延时和读写时钟周期有关）。

SHOWAHEAD 模式读时序图：



在时钟上升沿时，如果 re 为高且 empty_flag 为低，则在当前时钟周期读出 FIFO 数据到 do，完成一次读操作，同时读数据个数指示信号 rdusedw 自减 1（rdusedw 如时序图所示，延迟一个时钟周期输出），若干时钟延时后 wrusedw 自减 1（具体延时和读写时钟周期有关）。

SHOWAHEAD 模式读时序与普通模式读时序唯一的区别在于输出数据在 re 到来之前就已经在 do 准备好，re 信号作为应答信号，通知 FIFO 当前数据已经读出，可以准备下一个数据。

5. 点击“OK”完成 RAMFIFO 的设置，TD 将给出生成文件的路径。

可通过选择 Tools → IP Generator，选择“Edit an existing IP core”来打开一个已存在的 demo_ramfifo.ipc。

3.7.2 例化 RAMFIFO 模块

本手册以新建工程为例介绍例化 RAMFIFO 模块的过程。用户也可以在已有工程的基础上进行例化，例化过程一致。

1. 新建工程，并为工程添加顶层模块。
2. 在工程中添加上一步生成的 demo_ramfifo.v
3. 在顶层模块中调用 demo_ramfifo 模块，并修改 inst 名称和端口名称，点击保存按钮，即完成了 RAMFIFO 模块的例化。

top.v

```
module top (
    rst,
    di, clkr, re, clkw, we,
    do,
    empty_flag,full_flag,rdusedw,wrusedw
);

input rst;
input [7:0] di;
input clkw, we;
input clkr,re;

output [7:0] do;
output empty_flag;
output full_flag;
output [10:0] rdusedw;
output [10:0] wrusedw;

demo_ramfifo ramfifo(
    .rst(rst),
    .di(di),
    .clkw(clkw),
    .we(we),
    .do(do),
    .clkr(clkr),
    .re(re),
    .empty_flag(empty_flag),
    .full_flag(full_flag),
    .rdusedw(rdusedw),
    .wrusedw(wrusedw)
    );

endmodule
```

demo_ramfifo.v

```
`timescale 1ns / 1ps

module demo_ramfifo (
    rst,
    di, clkr, re, clkw, we,
    do,
    empty_flag,full_flag,rdusedw,wrusedw

);

    input rst;
    input [7:0] di;
    input clkw, we;
    input clkr,re;

    output [7:0] do;
    output empty_flag;
    output full_flag;
    output [10:0] rdusedw;
    output [10:0] wrusedw;

AL_LOGIC_RAMFIFO #(
    .DATA_WIDTH(8),
    .ADDR_WIDTH(10),
    .SHOWAHEAD(0)
)
logic_ramfifo(
    .rst(rst),
    .di(di),
    .clkw(clkw),
    .we(we),
    .do(do),
    .clkr(clkr),
    .re(re),
    .empty_flag(empty_flag),
    .full_flag(full_flag),
    .rdusedw(rdusedw),
    .wrusedw(wrusedw)
);

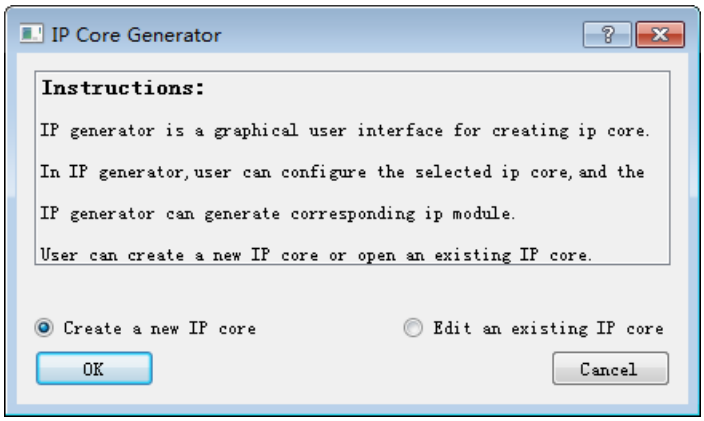
endmodule
```

3.8 DRAM 模块

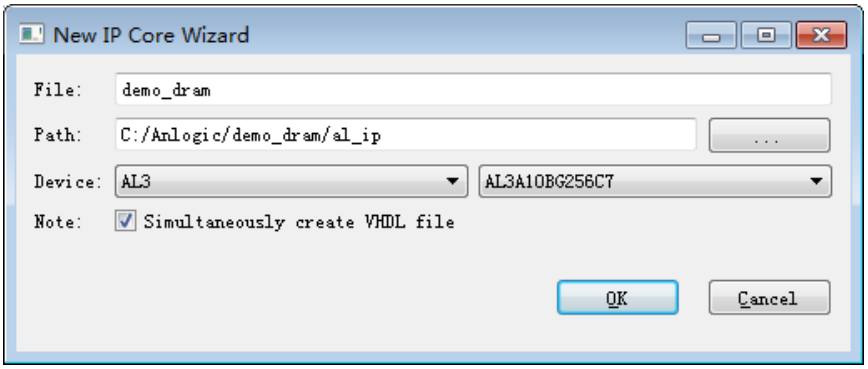
在 AL3 系列器件中，每个 PLB 包含 2 个 MSLICE（4 个 LUT），可实现 16x4 的 RAM 块，每个 DRAM 支持简单双口的 RAM。

3.8.1 创建 DRAM 模块

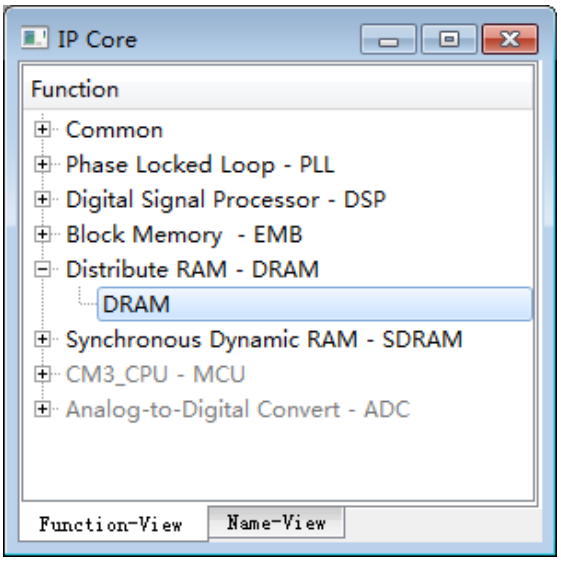
1. 选择 **Tools → IP Generator**，选择“**Create a new IP core**”



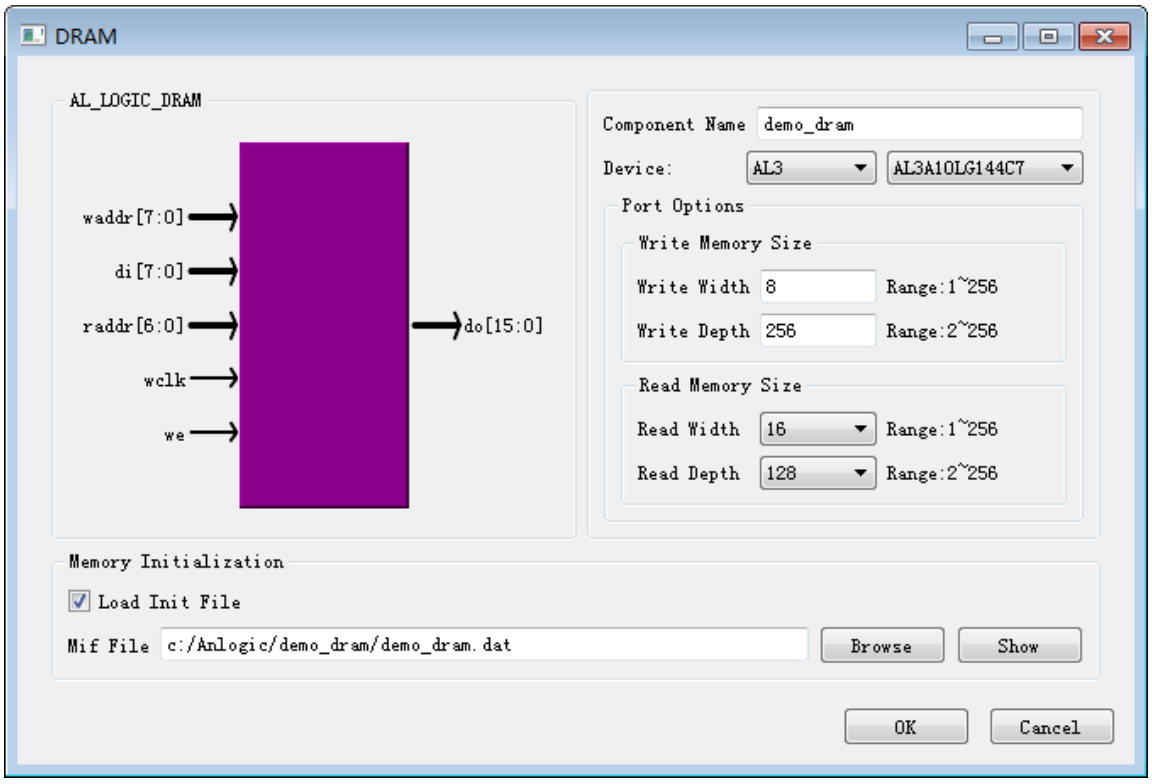
2. 输入模块名称并选择存储路径。此处，若是在有工程的基础上创建 DRAM 模块，存储路径和器件名称将与工程保持一致。若在没有工程的基础上创建 DRAM 模块，用户需手动设置保存路径和器件名称。若勾选“**Simultaneously create VHDL file**”, TD 将会生成相应的 VHDL 文件。



3. 在 Function 窗口中展开 **Distribute RAM**，双击 **DRAM** 打开配置界面



4. 填写“**Component Name**”并设置相应参数



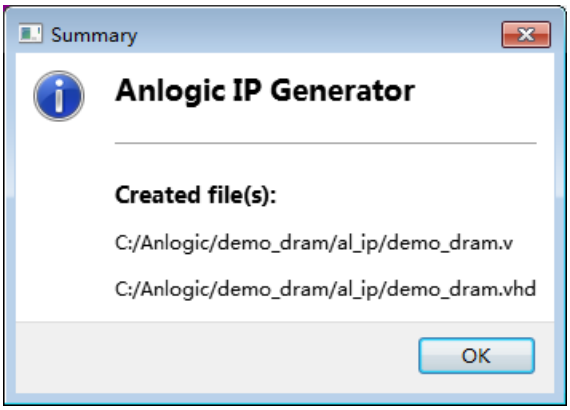
此处, Init File 的格式和 BRAM 中相同。

端口列表和说明如下：

其中，写端口由 WCLK 同步，读端口异步工作。

端口	功能	说明
WCLK	写操作时钟	上升沿有效（可编程反向）
WE	写使能	内嵌 WCLK 上升沿同步锁存器
WADDR[3:0]	写地址	内嵌 WCLK 上升沿同步锁存器
DI[3:0]	写数据	内嵌 WCLK 上升沿同步锁存器
RADDR[3:0]	读地址	异步
DO[3:0]	读数据	异步

5. 点击“OK”完成 DRAM 的设置，TD 将给出生成文件的路径。

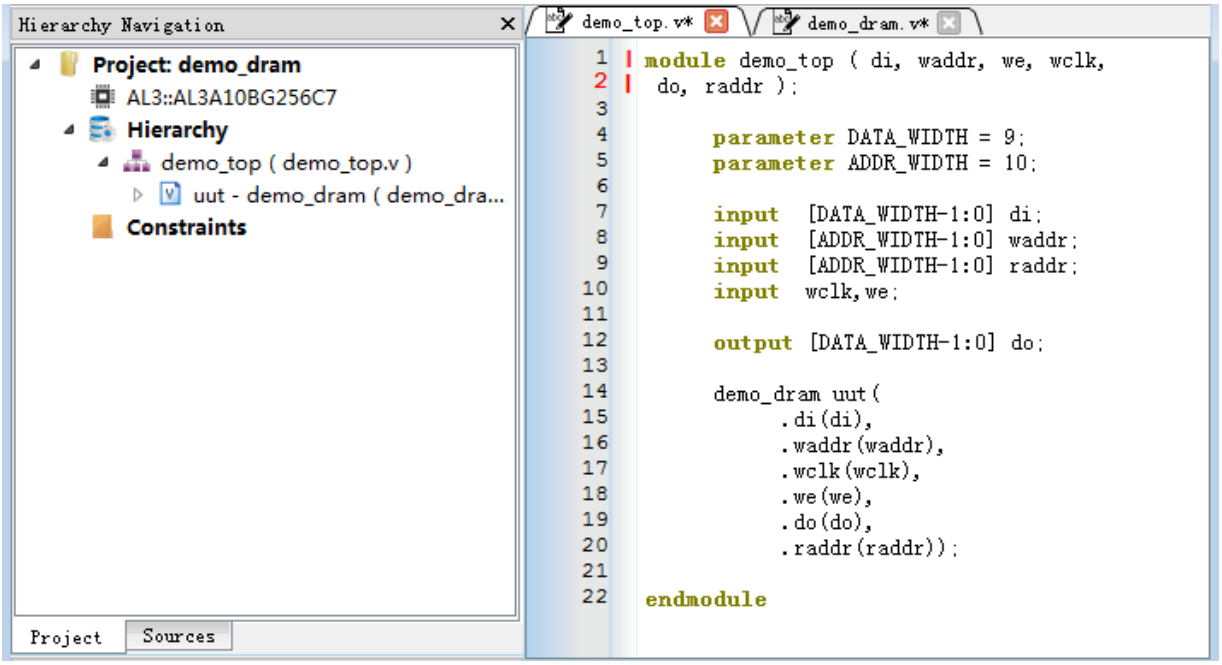


同样可通过“**Edit an existing IP core**”方式来打开并编辑已存在的 demo_dram.ipc。

3.8.2 例化 DRAM 模块

该手册以新建工程为例介绍例化 DRAM 模块的过程。用户也可以在已有工程的基础上进行例化，例化过程一致。

1. 新建工程，并为工程添加顶层模块。
2. 在工程中添加上一步生成的 demo_dram.v
3. 在顶层模块中调用 demo_dram 模块，并修改 inst 名称和端口名称，点击保存按钮，即完成了 DRAM 模块的例化。

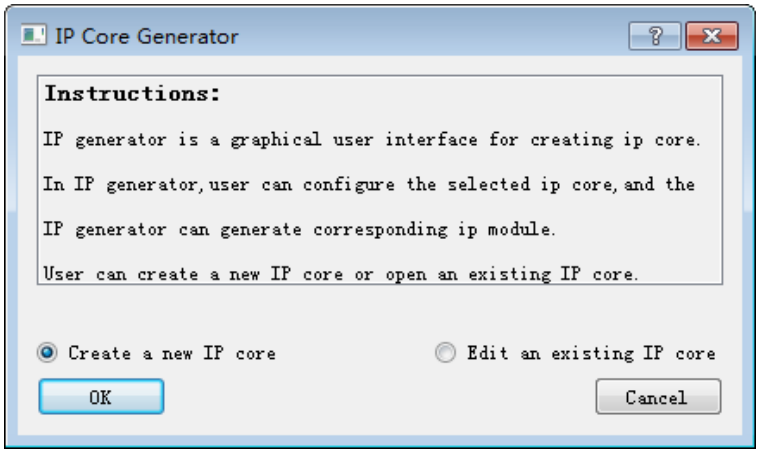


3.9 SDRAM 模块

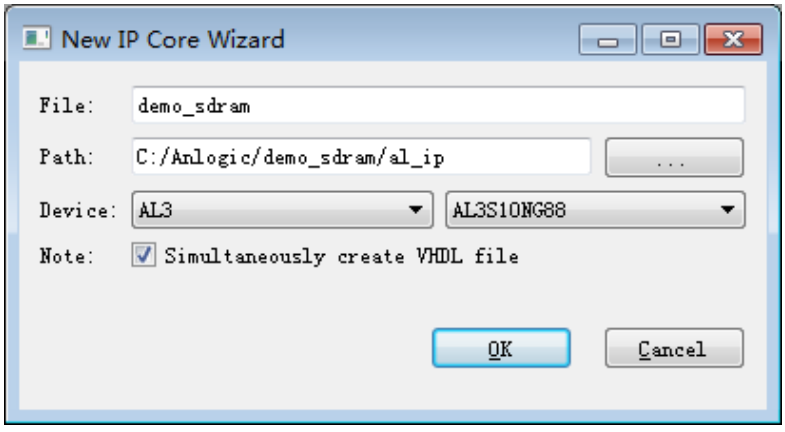
部分器件有内嵌的 SDRAM 模块，本节将介绍该模块的创建，实例化以及使用时的相关注意事项。

3.9.1 创建 SDRAM 模块

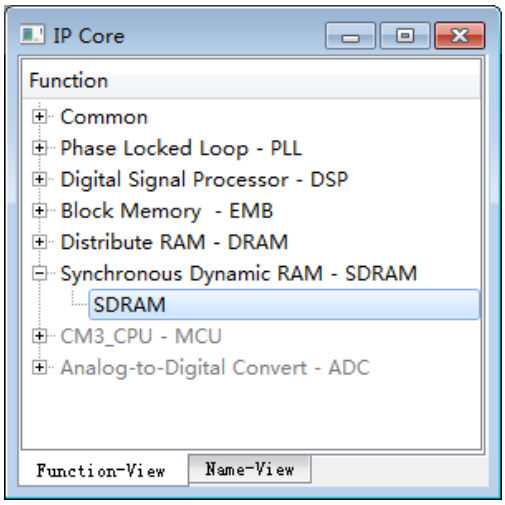
1. 选择 **Tools → IP Generator**，选择“**Create a new IP core**”

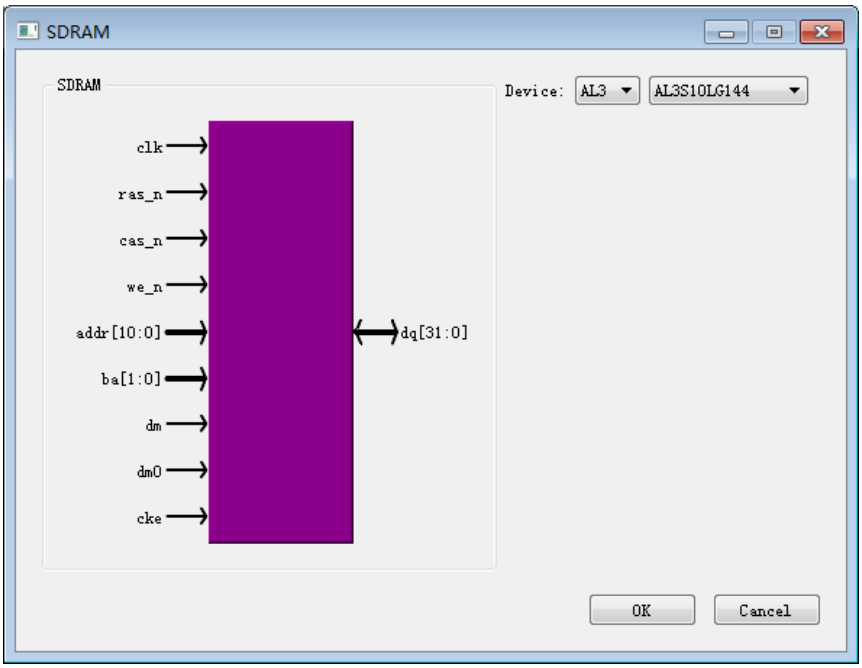


2. 输入模块名称并选择存储路径。此处，若是在有工程的基础上创建 SDRAM 模块，存储路径和器件名称将与工程保持一致。若在没有工程的基础上创建 SDRAM 模块，用户需手动设置保存路径和器件名称。若勾选“**Simultaneously create VHDL file**”, TD 将会生成相应的 VHDL 文件。

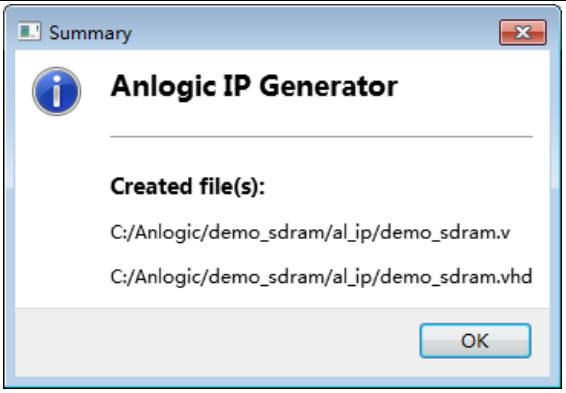


3. 在 Function 窗口中展开 **Synchronous Dynamic RAM**，双击 **SDRAM** 打开配置界面





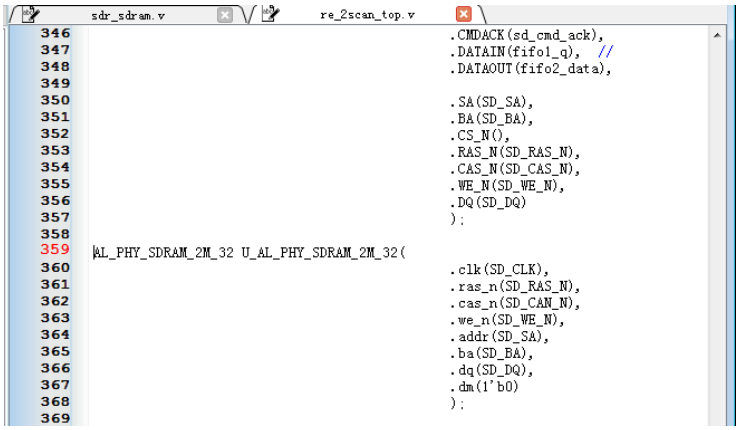
4. 点击“OK”完成设置，生成文件如下：



3.9.2 例化 SDRAM 模块

下面以 AL3S10 器件为例来介绍 SDRAM 模块的使用。SDRAM 与 FPGA 通过软件深度整合。所以如果要使用 SDRAM，只需要在顶层实例化如下 IP 模块即可。该 IP 的原型如下：

```
AL_PHY_SDRAM_2M_32    U_AL_PHY_SDRAM_2M_32(
        .clk(SD_CLK),              // SDRAM 时钟          1bit 位宽
        . ras_n(SD_RAS_N),         // SDRAM  行选通       1bit 位宽
        .cas_n(SD_CAS_N),          //SDRAM 列选通         1bit 位宽
        .we_n(SD_WE_N),            //SDRAM 写使能         1bit 位宽
        .addr(SD_SA),              //SDRAM 地址           11bits 位宽
        .ba(SD_BA),                // SDRAM BANK 地址     2bits 位宽
        .dq(SD_DQ),                // SDRAM  数据          32bits 位宽
        .dm1(1'b0)                 // SDRAM 数据屏蔽       1bit 位宽
);
```



SDRAM 的引脚分配如下：

SDRAM 引脚名称	SDRAM 引脚描述	引脚连接
DQ0 ~ DQ31	数据脚 0 ~ 数据脚 31	与 IP 相连
SA0 ~ SA10	地址脚 0 ~ 地址脚 10	与 IP 相连
BA0	BANK 地址脚 0	与 IP 相连
BA1	BANK 地址脚 1	与 IP 相连
WE_N	写使能	与 IP 相连
RAS_N	行选通	与 IP 相连
CAS_N	列选通	与 IP 相连
CLK	芯片时钟	与 IP 相连
CS_N	片选	内部拉低
DM0	数据 0-7 屏蔽	内部拉低
DM1	数据 8-15 屏蔽	与 IP 相连
DM2	数据 16-23 屏蔽	内部拉低
DM3	数据 24-31 屏蔽	内部拉低
CKE	时钟使能	内部拉高

使用 SDRAM 时，请注意以下几点：

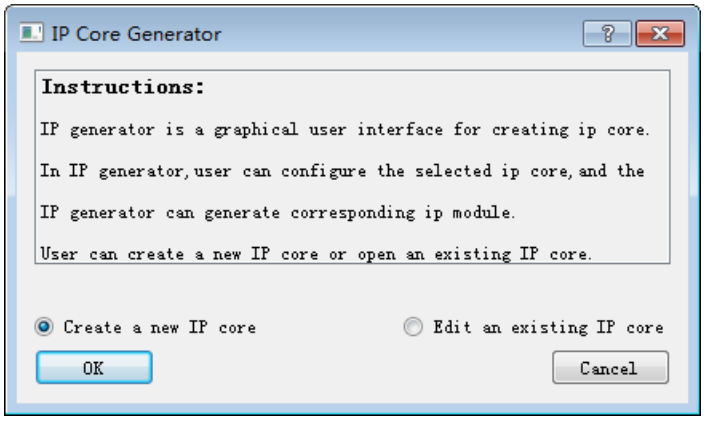
1. DQ0~DQ31 为双向数据脚；
2. SDRAM 工作时钟 CLK 与 FPGA 内部时钟需要 90 度相位差；
3. 给 DM1 赋 0 值时,32 位数据可用；给 DM1 赋 1 值,屏蔽 8-15 位数据,可降低功耗。

3.10 ADC 模块

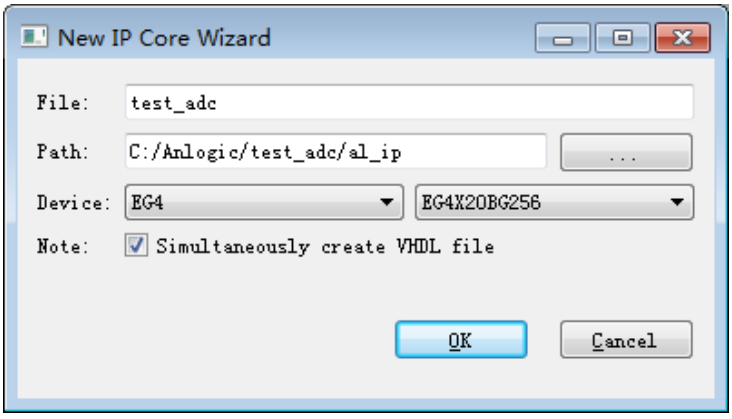
Eagle 系列新品内嵌有一个 8 通道的 12 位 1MSPS ADC，位于芯片的 BANK8。ADC 模块需要独立的 3.3V 模拟工作电压和模拟地以及一个独立的 VREF 电压输入。8 个通道输入和用户 IO 复用，当用户不需要使用 ADC 模块时可用作普通 IO 使用。当使用 ADC 时，BANK8 的 VCCIO 电压不应低于 ADC 模拟电源电压。

3.10.1 创建 ADC 模块

1. 选择 **Tools → IP Generator**，选择“**Create a new IP core**”

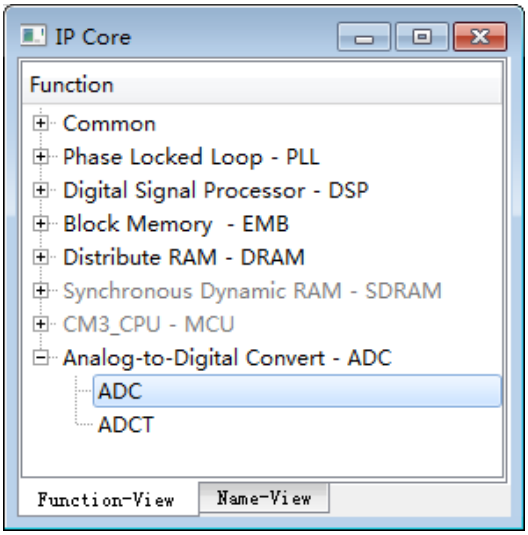


2. 输入模块名称并选择存储路径。此处，若是在有工程的基础上创建 ADC 模块，存储路径和器件名称将与工程保持一致。若在没有工程的基础上创建 ADC 模块，用户需手动设置保存路径和器件名称。若勾选“**Simultaneously create VHDL file**”，TD 将会生成相应的 VHDL 文件。

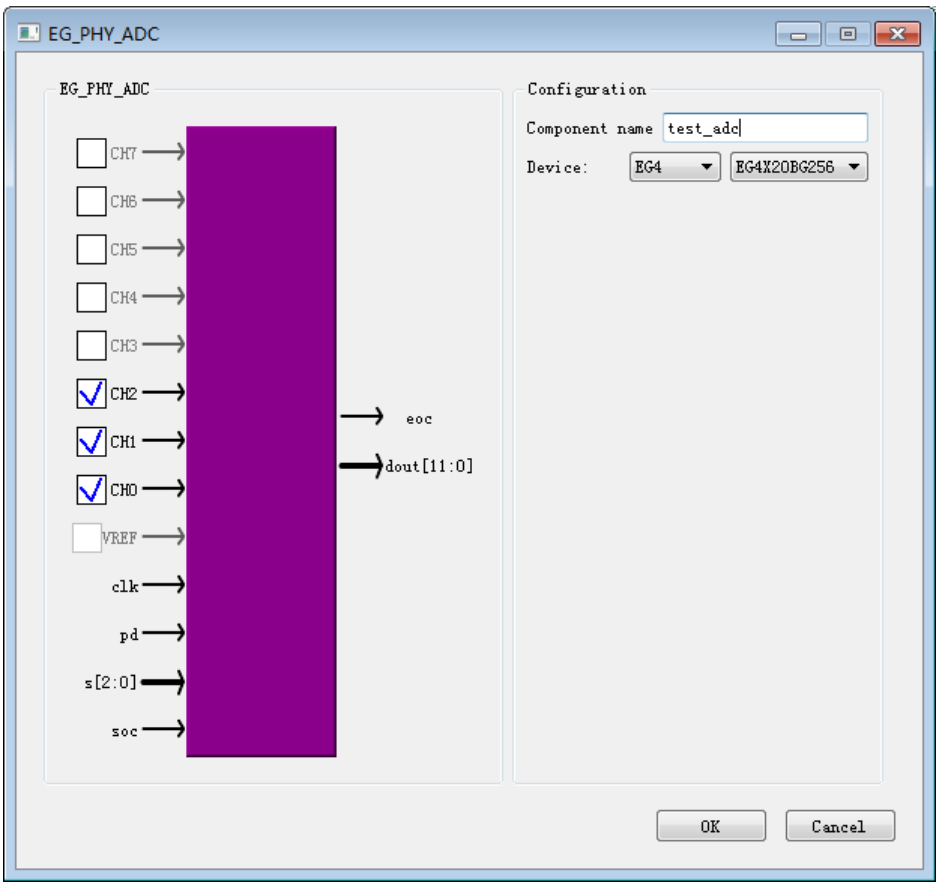


3. 在 Function 窗口中展开 **Analog-to-Digital Convert**，双击 ADC 打开配置界面。

   ADCT 模块在 ADC 模块的基础上添加了温度传感器。



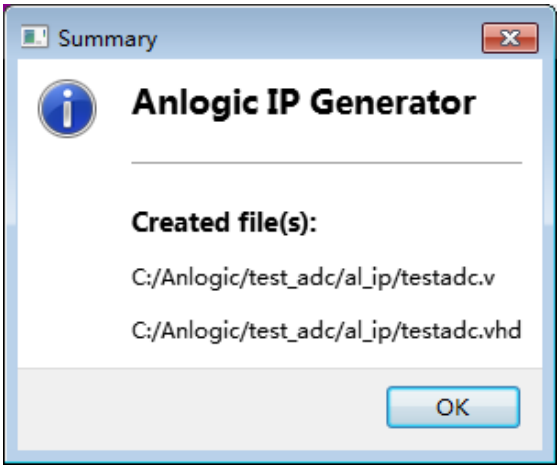
4. 填写“**Component Name**”，并选择 ADC 的通道，当前能够使用的通道，取决于当前器件的封装类型。



ADC 模块外部/内部端口说明如下：

芯片端口名	端口类型	说明
clk	外部电源 PAD	3.3V 模拟电源输入
pd	外部电源 PAD	3.3V 模拟地
VREF	外部 PAD	独立输入，采样参考模拟电位输入，输入电压范围 2.0V~3.3V，不大于 VDDA
CH<7：0>	外部 PAD	8 路采样信号输入，和用户 IO 复用
内部端口名	**端口方向**	**说明**
s<2:0>	输入（来自 FPGA）	ADC 通道选择信号输入
soc	输入（来自 FPGA）	ADC 采样使能信号输入，高有效
eoc	输出（到 FPGA）	ADC 转换完成输出，高有效
dout<11:0>	输出（到 FPGA）	对应通道的 ADC 转换结果

5. 点击“OK”完成设置，生成的文件如下：

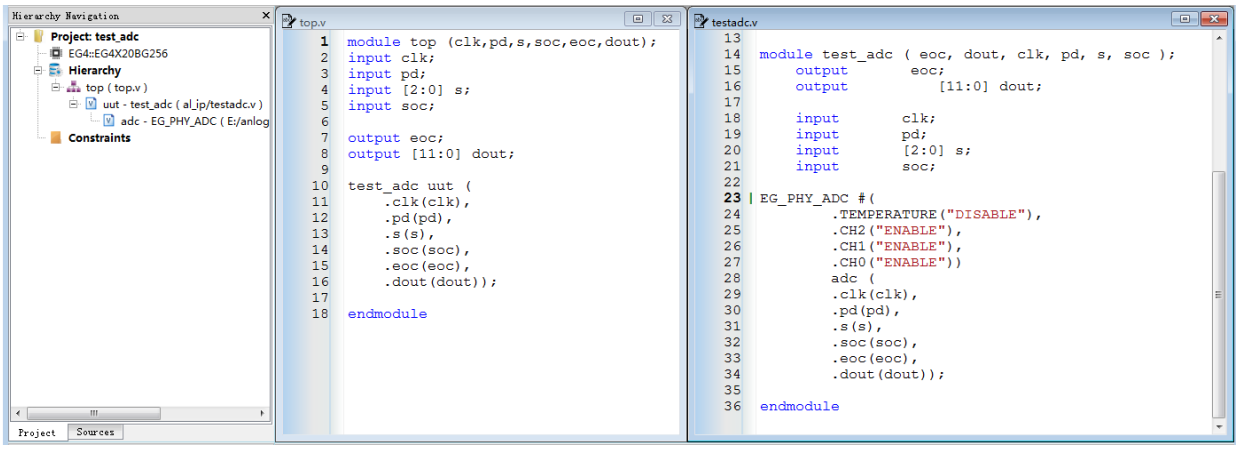


同样可通过“**Edit an existing IP core**”方式来打开并编辑已存在的 test_adc.ipc。

3.10.2 例化 ADC 模块

该手册以新建工程为例介绍例化 ADC 模块的过程。用户也可以在已有工程的基础上进行例化，例化过程一致。

1. 新建工程，并为工程添加顶层模块。
2. 在工程中添加上一步生成的 test_adc.v
3. 在顶层模块中调用 test_adc 模块，并修改 inst 名称和端口名称，点击保存按钮，即完成了 ADC 模块的例化。



4 用户约束

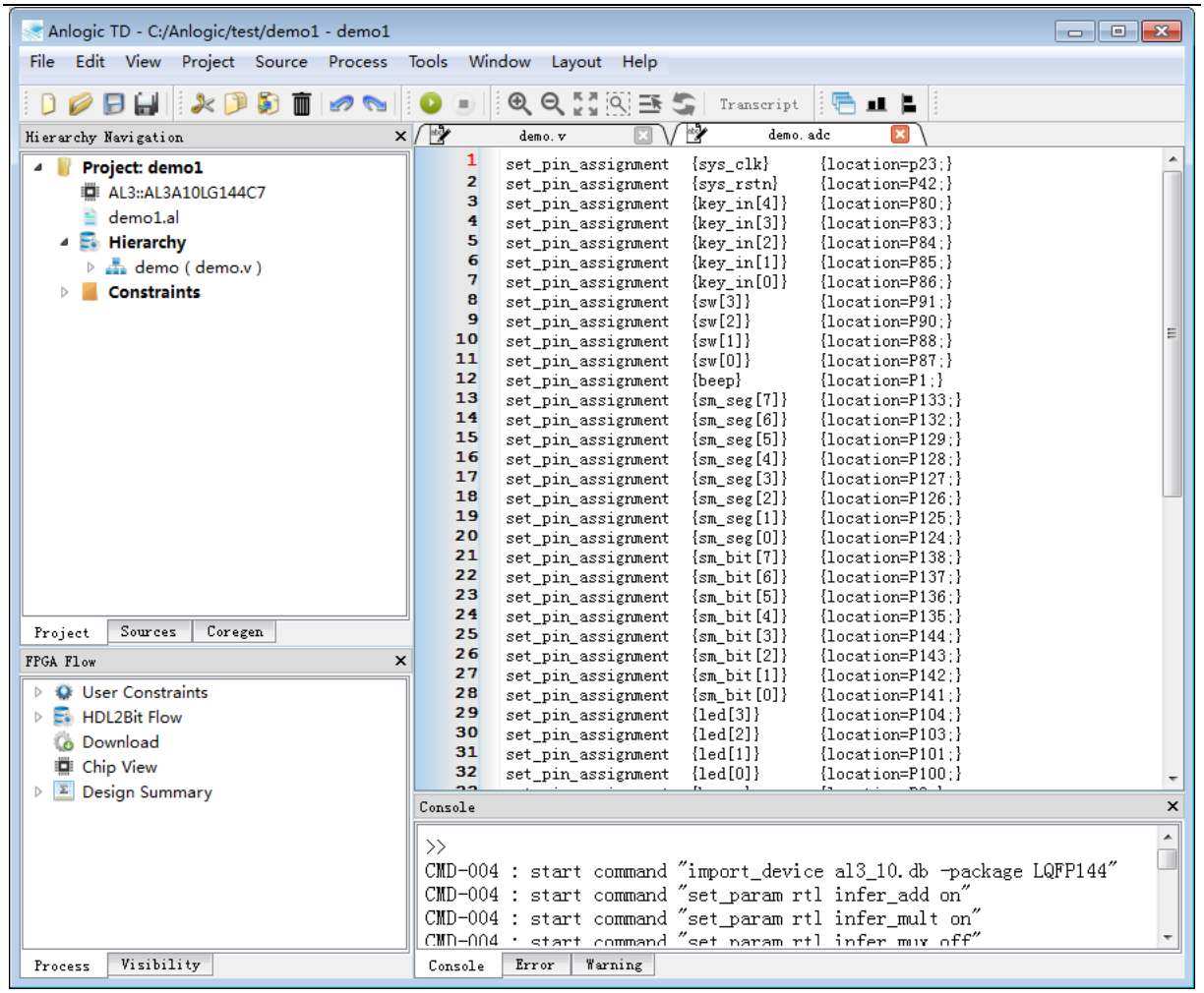
用户约束包括两个部分:物理约束和时序约束。物理约束采用 Anlogic 自定义的 ADC（Anlogic Design Constraint)格式,对 FPGA 芯片的管脚特性和内部单元特性进行约束。管脚特性包括 **Bank**, **Location**, **PullType**, **IOStandard**, **SlewRate**, **Drivestrength**, **VREF**, **DiffResistor**, **PCIClamp** 共 9 种。单元特性约束支持用户在 ADC 文件中用 set_inst_assignment 指定单元 **Location**。管脚特性约束可以通过界面进行设置。物理约束说明请参考附录 9.1。时序约束用工业界标准的 SDC（Synopsys Design Constraint）格式，对 FPGA 芯片的外部时延信息和内部时序需求进行约束，设计流程中和时延相关的优化会考虑 SDC 的约束。时序约束说明请参考附录 9.2。

4.1 物理约束

4.1.1 添加 IO 约束

新建 ADC 文件

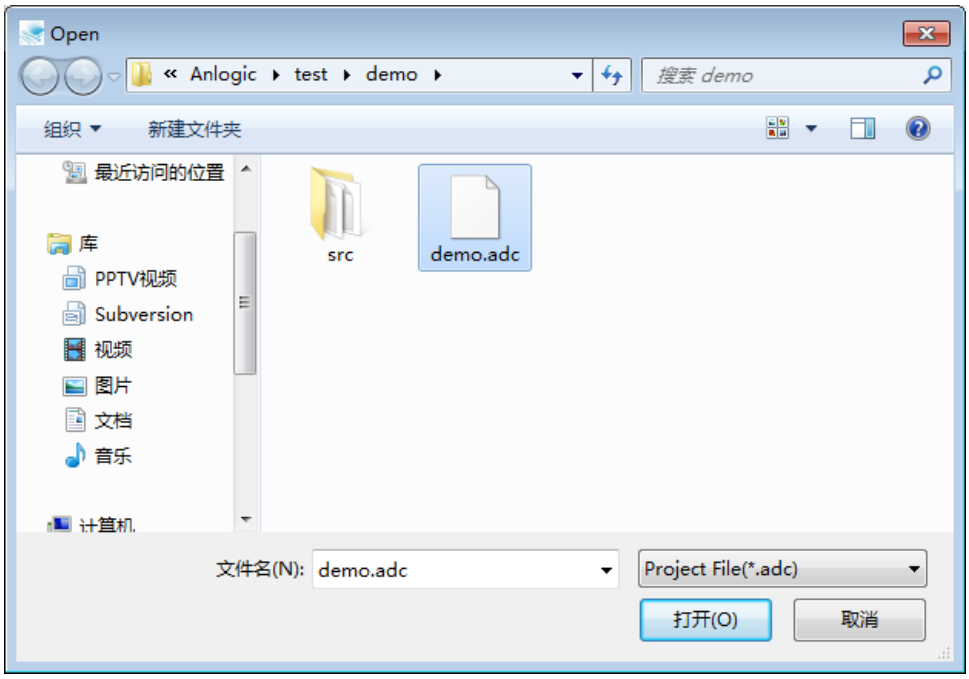
1. 点击新建按钮，在新打开的窗口中输入 ADC 命令



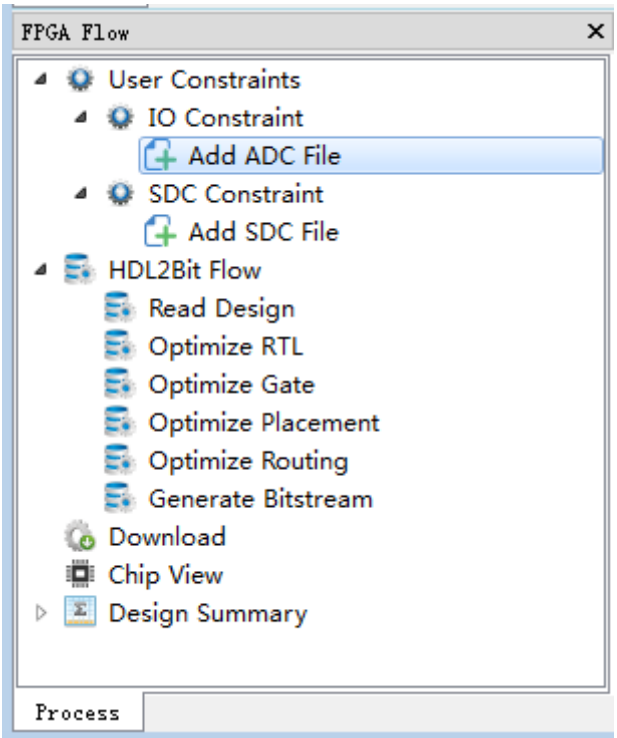
2. 点击保存按钮，将文件路径设置在项目目录下，文件名为 demo.adc，然后单击保存。

添加 ADC 文件

1. 右键单击 Hierarchy Navigation 面板中 Project 里的 **Constraints**，选择 **Add ADC File**，选择 demo.adc，打开。



或者通过双击 FPGA Flow 面板中 **User Constraints** 下的 **Add ADC File**，选择 demo.adc，点击打开。



4.1.2 界面设置 IO 约束

1. 在 FPGA Flow 面板中展开 **User Constraints**

2. 双击 **IO Constraint**

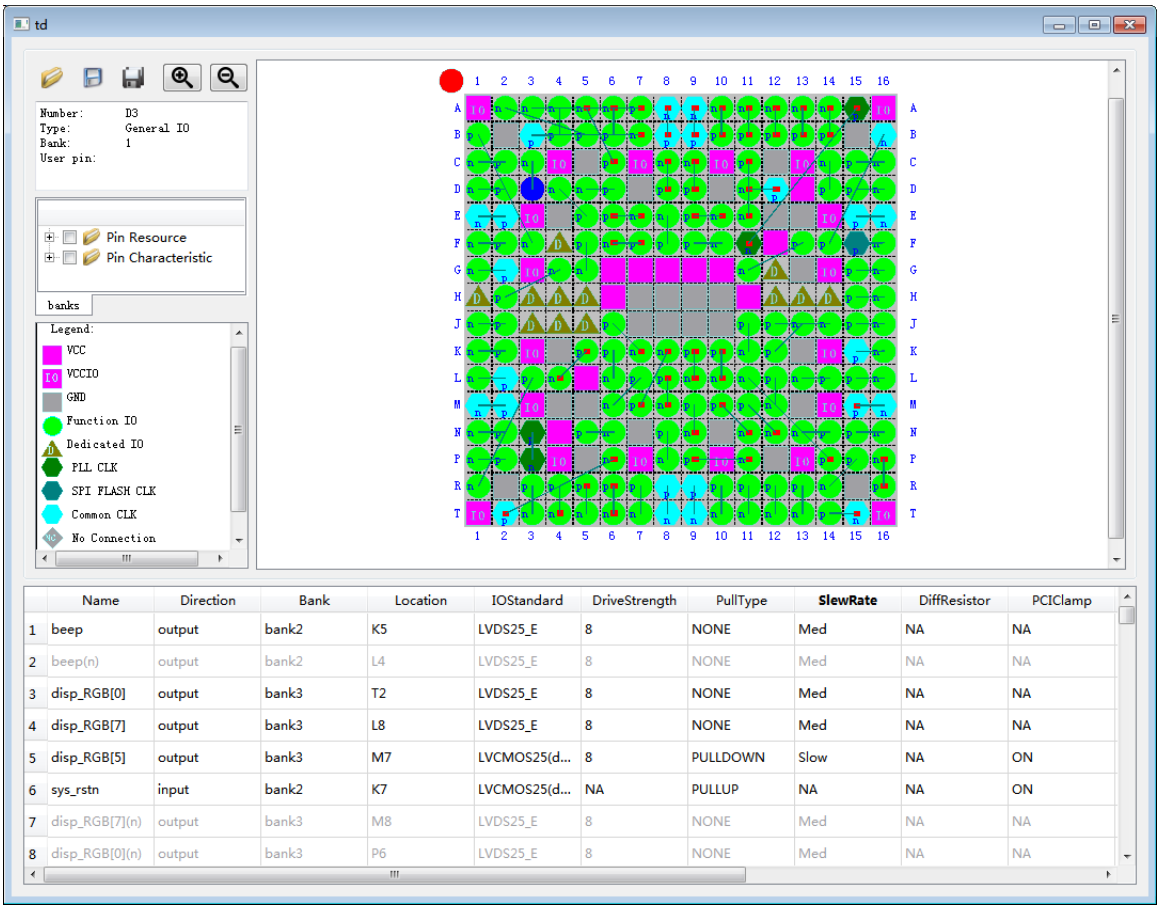
3. 设置每个端口的 **Bank**、**Location**、**PullType**、**IOStandard** 等参数。

芯片封装为 BGA256 时，IO 界面配置属性如下表所示

表4-1　IO 界面配置属性

类别	颜色	形状	功能
	品红色	正方形	电源
	灰色	正方形	地
	绿色	圆形	Function IO
	暗黄色	三角形	Dedicated IO
	蓝色	六边形	Common CLK
	深绿色	六边形	PLL
	深蓝色	六边形	Spi flash clock
	褐色	三角形	Dedicated IO
	浅灰色	菱形	No Connection
		短斜线	差分对

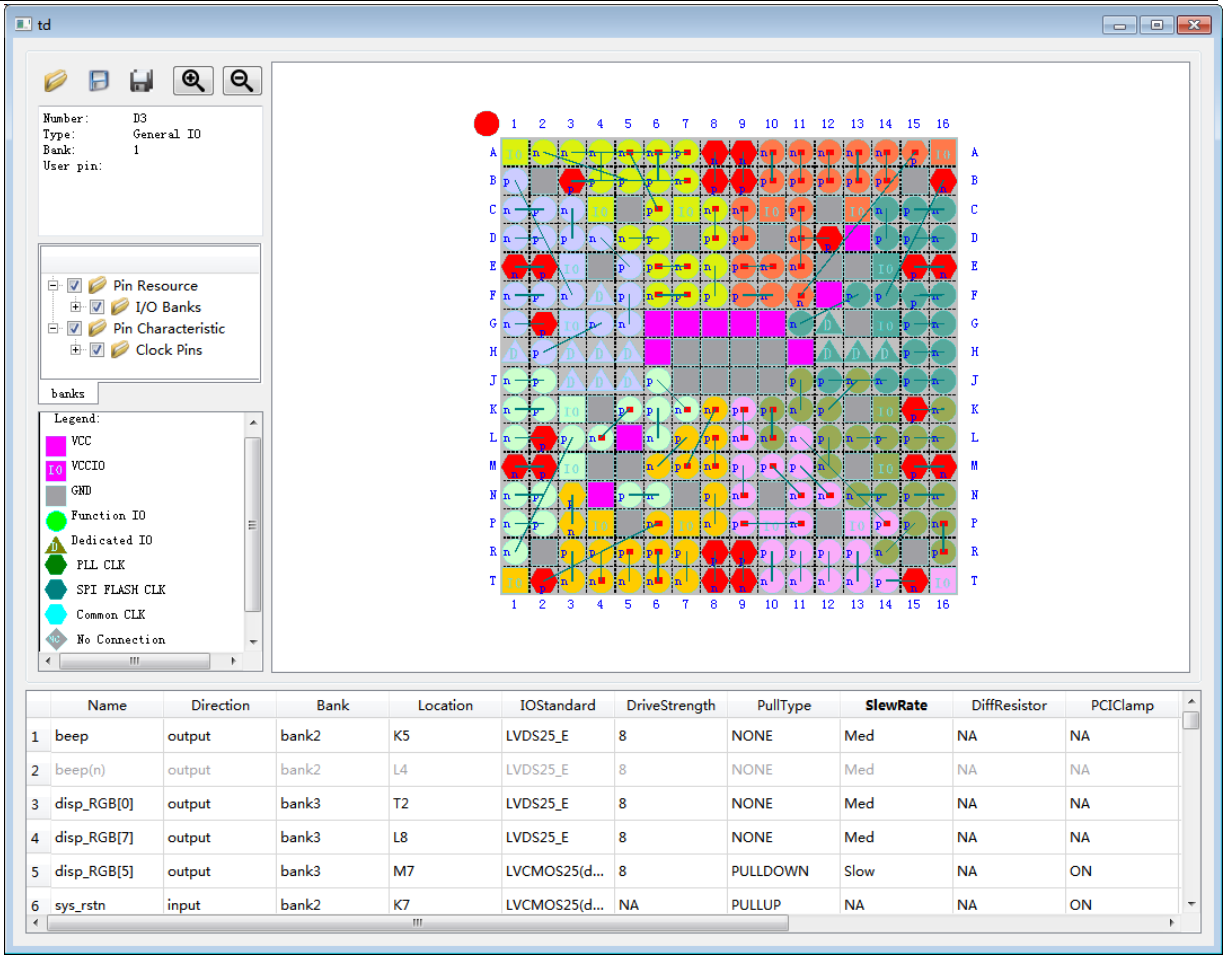
芯片封装为 BGA256 时，IO 约束的配置界面如下图所示



	Name	Direction	Bank	Location	IOStandard	DriveStrength	PullType	SlewRate	DiffResistor	PCIClamp
1	beep	output	bank2	K5	LVDS25_E	8	NONE	Med	NA	NA
2	beep(n)	output	bank2	L4	LVDS25_E	8	NONE	Med	NA	NA
3	disp_RGB[0]	output	bank3	T2	LVDS25_E	8	NONE	Med	NA	NA
4	disp_RGB[7]	output	bank3	L8	LVDS25_E	8	NONE	Med	NA	NA
5	disp_RGB[5]	output	bank3	M7	LVCMOS25(d...	8	PULLDOWN	Slow	NA	ON
6	sys_rstn	input	bank2	K7	LVCMOS25(d...	NA	PULLUP	NA	NA	ON
7	disp_RGB[7](n)	output	bank3	M8	LVDS25_E	8	NONE	Med	NA	NA
8	disp_RGB[0](n)	output	bank3	P6	LVDS25_E	8	NONE	Med	NA	NA

各形状中的红色小点表示该 IO 端口已被占用。

用户还可以通过不同颜色来区分不同 Bank，如下图所示。其中，红色：时钟、品红色：电源、 灰色：接地，这三种类型的端口不属于任何 Bank。 左上角的红色原点对应芯片上的小凹点，表示芯片引脚的起始点。

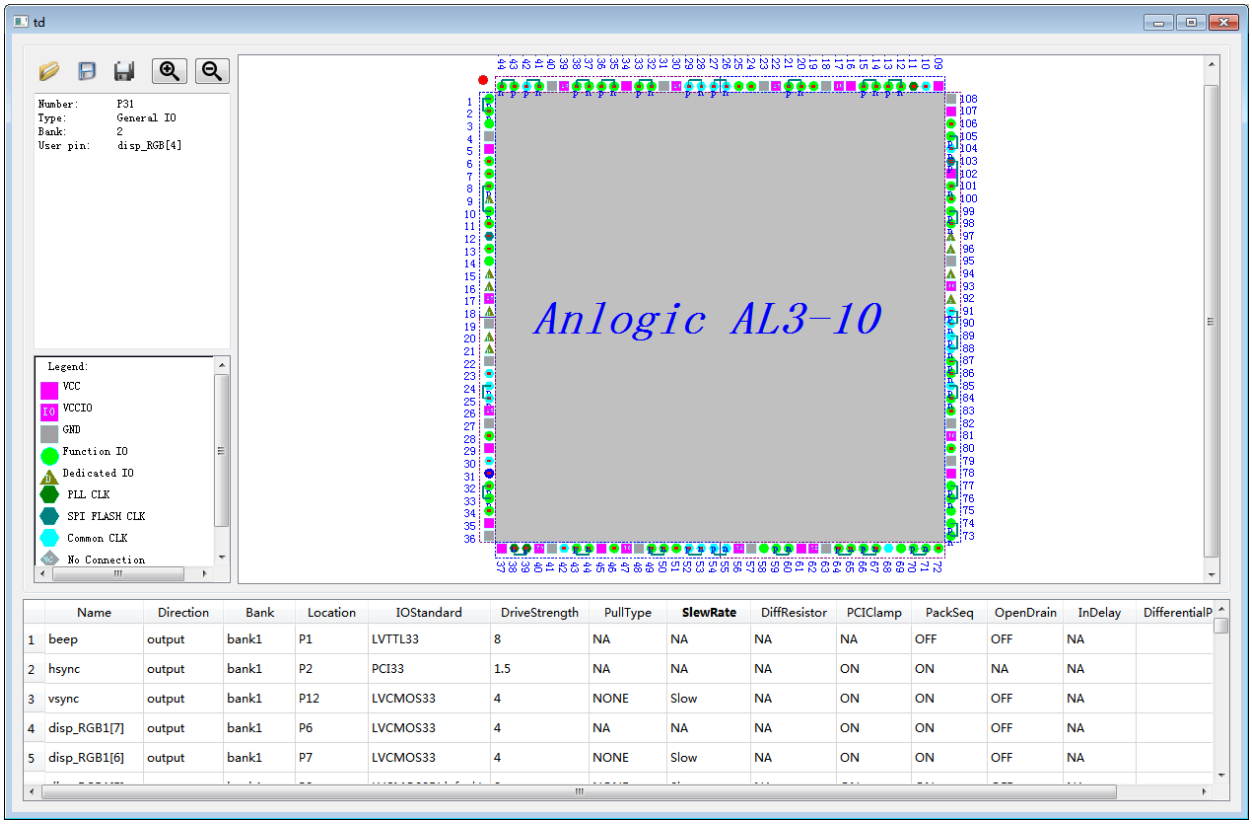


	Name	Direction	Bank	Location	IOStandard	DriveStrength	PullType	SlewRate	DiffResistor	PCIClamp
1	beep	output	bank2	K5	LVDS25_E	8	NONE	Med	NA	NA
2	beep(n)	output	bank2	L4	LVDS25_E	8	NONE	Med	NA	NA
3	disp_RGB[0]	output	bank3	T2	LVDS25_E	8	NONE	Med	NA	NA
4	disp_RGB[7]	output	bank3	L8	LVDS25_E	8	NONE	Med	NA	NA
5	disp_RGB[5]	output	bank3	M7	LVCMOS25(d...	8	PULLDOWN	Slow	NA	ON
6	sys_rstn	input	bank2	K7	LVCMOS25(d...	NA	PULLUP	NA	NA	ON

其中短斜线表示差分对，如上图中的 B1 (positive) 和 F3 (negative)。若选择 B1 的 IOStandard 属性为 LVDS25，则 TD 自动分配相应的差分对管脚 F3(negative)，F3 的 IOStandard 属性也为 LVDS25， 如下图的 beep 对应的差分信号为 beep(n)。

Name	Direction	Bank	Location	IOStandard	DriveStrength	PullType
vsync	output	bank1	P14	LVCMOS33	4	NONE
hsync	output	bank1	P3	PCI33	1.5	NONE
beep	output	bank1	P2	LVDS25	NA	NONE
beep(n)	output	bank1	P1	LVDS25	NA	NONE

当芯片封装为 LG144 时，IO 约束配置属性如表 4-1，配置界面如下图：



	Name	Direction	Bank	Location	IOStandard	DriveStrength	PullType	SlewRate	DiffResistor	PCIClamp	PackSeq	OpenDrain	InDelay	DifferentialP
1	beep	output	bank1	P1	LVTTL33	8	NA	NA	NA	NA	OFF	OFF	NA	
2	hsync	output	bank1	P2	PCI33	1.5	NA	NA	NA	ON	ON	NA	NA	
3	vsync	output	bank1	P12	LVCMOS33	4	NONE	Slow	NA	ON	ON	OFF	NA	
4	disp_RGB1[7]	output	bank1	P6	LVCMOS33	4	NA	NA	NA	ON	ON	OFF	NA	
5	disp_RGB1[6]	output	bank1	P7	LVCMOS33	4	NONE	Slow	NA	ON	ON	OFF	NA	

BANK和LOCATION的设置：

芯片封装不同时，管脚数和管脚名称不同，**Bank** 数也不同，以 Anlogic AL3-10 系列为例，IO 管脚共分为 8 个 **Bank**。管脚位置设置时可先指定 **Bank**，再指定 **Bank** 中的 **Location**，或直接指定 **Location**，TD 会自动匹配相应的 **Bank**。若只选择 **Bank**，而不指定 **Location** 时，TD 会默认为用户选择这个 **Bank** 中管脚号最小的 Pin 作为这个管脚的 **Location**。当指定 Location 为 **VirtualIO** 时，则该 Port 被指定到了一个非物理 IO，不属于任何 Bank，在综合布线时，也不会为其分配任何 IO 资源，VirtualIO 的所有电平参数都不可设置。

	Name	Direction	Bank	**Location**	IOStandard	DriveStrength	PullType	SlewRate
1	beep	output		virtualIO	LVTTL33	8	NONE	Slow
2	hsync	output		virtualIO	PCI33	1.5	NONE	Slow

	Name	Direction	Bank	Location	IOStandard	DriveStrength	PullType	SlewRate	DiffResistor
1	beep	output	bank4	P54	LVTTL33	8	NONE	Slow	NA
2	hsync	output		P1	PCI33	1.5	NONE	Slow	NA
3	vsync	output		P12	LVCMOS33	4	NONE	Slow	NA
4	disp_RGB1...	output		P6	LVCMOS33	4	NA	NA	NA
5	disp_RGB1...	output		P7	LVCMOS33	4	NONE	Slow	NA
6	disp_RGB1...	output		P8	LVCMOS25(default)	8	NONE	Slow	NA
7	disp_RGB1...	output		P10	LVCMOS25(default)	8	NA	NA	NA
8	disp_RGB1...	output		P11	LVCMOS25(default)	8	NA	NA	NA

bank1
bank2
bank3
bank4
bank5
bank6
bank7
bank8

IOstandard 的设置:

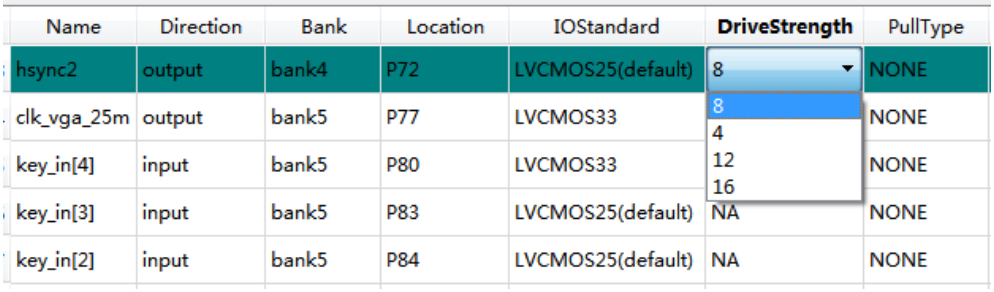
IOStandard 设置 IO 端口的电平标准。每个 **Bank** 都可以随意设置为支持该器件的电平标准,不同的电平标准在同一个 **Bank** 中的电平要一致。TD 提供 **LVCMOS**、**LVDS**、**LVTTL33**、**PCI33** 供用户选择。其中 **LVCMOS** 有 1.2v，1.5V，1.8V，2.5V，3.3V 的电压可选择。**LVDS** 为差分对输入输出,当所选的 IO 端口为输入信号时,只能选择 **LVDS25**,**LVDS33**，**LVPECL33,** 所选 IO 端口为输出信号时，可选 **LVDS25_E**，**LVDS33_E**，**LVPECL33_E**。默认的电平标准为 **LVCMOS25(default)**。

Name	Direction	Bank	Location	IOStandard	DriveStrength	PullType	SlewRate	DiffResistor	PCIClamp	PackSeq	OpenDrain	InDelay
hsync2	output	bank4	P72	LVCMOS25(default)	8	NA	NA	NA	NA	ON	NA	NA
clk_vga_25m	output	bank5	P77	LVCMOS33	8	NONE	Slow	NA	ON	ON	OFF	NA
key_in[4]	input	bank5	P80		NA	NONE	NA	NA	ON	ON	NA	NONE
key_in[3]	input	bank5	P83		NA	NONE	NA	NA	NA	ON	NA	NA
key_in[2]	input	bank5	P84		NA	NONE	NA	NA	NA	ON	NA	NA
key_in[1]	input	bank5	P85		NA	NONE	NA	NA	NA	ON	NA	NA
key_in[0]	input	bank5	P86		NA	NONE	NA	NA	NA	ON	NA	NA
sw[0]	input	bank5	P87		NA	NONE	NA	NA	NA	ON	NA	NA

LVCMOS12
LVCMOS15
LVCMOS18
LVCMOS25
LVCMOS33
LVTTL33
PCI33
LVDS25
LVDS25_E
LVPECL33_E

DriveStrength 的设置

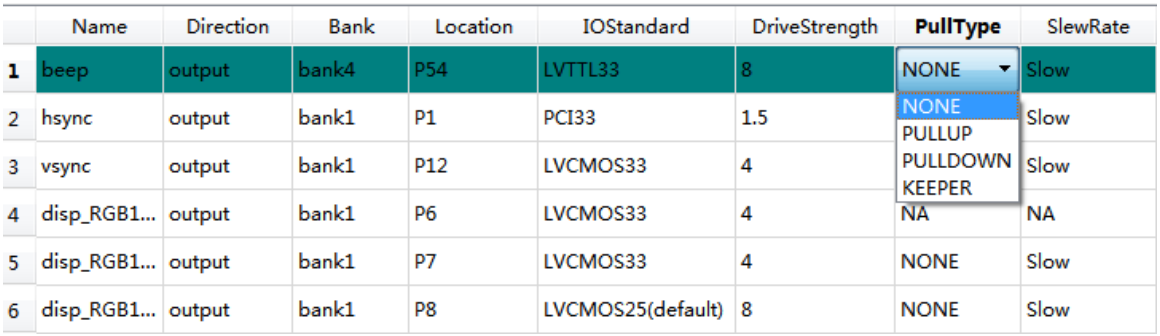
DriveStrength 设置 IO 端口的驱动能力。不同电平标准的驱动能力不同，如 **LVCMOS25** 的驱动能力为 **4,8,12,16**,单位为 mA。**DriveStrength** 的值越小，表示驱动能力越弱，**DriveStrength** 的值越大，表示驱动能力越强。



Name	Direction	Bank	Location	IOStandard	DriveStrength	PullType
hsync2	output	bank4	P72	LVCMOS25(default)	8	NONE
clk_vga_25m	output	bank5	P77	LVCMOS33	8	NONE
key_in[4]	input	bank5	P80	LVCMOS33	4	NONE
key_in[3]	input	bank5	P83	LVCMOS25(default)	12	NONE
key_in[2]	input	bank5	P84	LVCMOS25(default)	16	NONE

PullType 的设置:

PullType 设置 IO 端口的上拉类型。**PullType** 共有四种选择：**PULLUP**、**PULLDOWN**、**NONE**、**KEEPER**。数字电路有三种状态：高电平、低电平和高阻状态。当输入为无效信号的时候，可以通过上拉（**PULLUP**）电阻和下拉**(PULLDOWN)**电阻的方式使其处于稳定状态。当选择**(KEEPER)**时，使电平保持为上一个有效值。当 IO 端口设为 **LVDS** 的时候，**PullType** 只能设为 **None**。

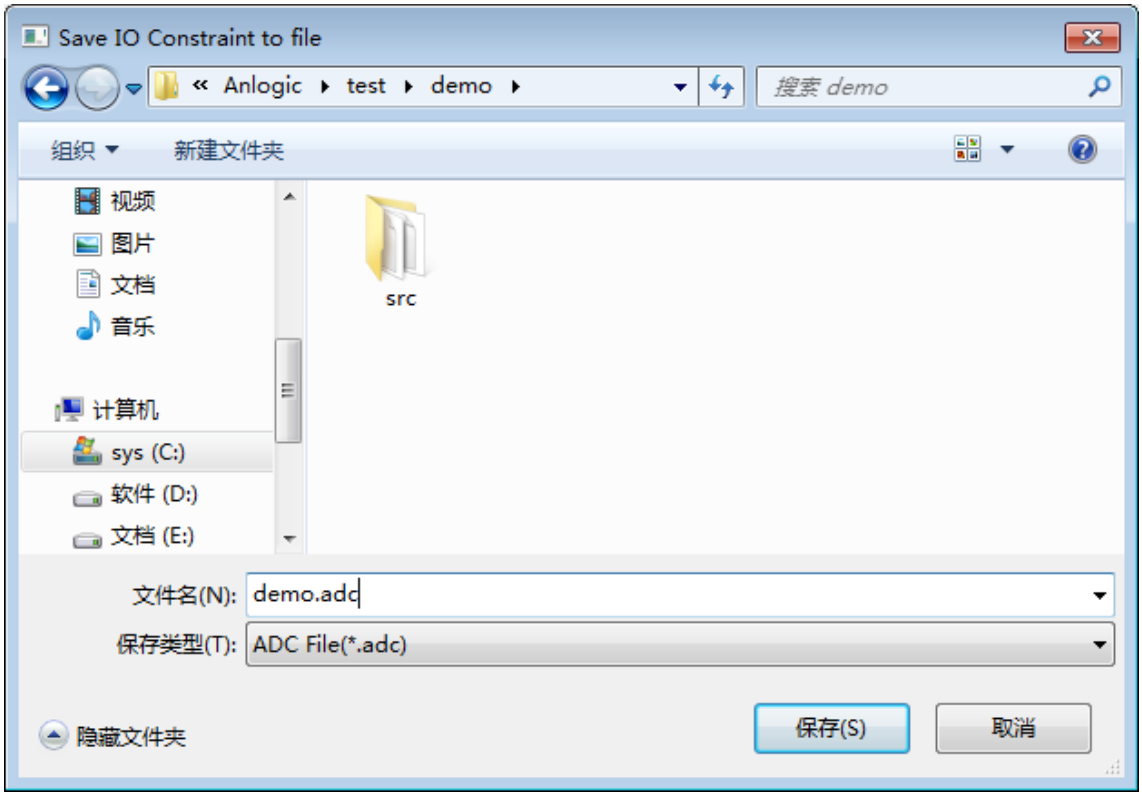


	Name	Direction	Bank	Location	IOStandard	DriveStrength	PullType	SlewRate
1	beep	output	bank4	P54	LVTTL33	8	NONE	Slow
2	hsync	output	bank1	P1	PCI33	1.5	NONE	Slow
3	vsync	output	bank1	P12	LVCMOS33	4	PULLUP	Slow
4	disp_RGB1...	output	bank1	P6	LVCMOS33	4	PULLDOWN	NA
5	disp_RGB1...	output	bank1	P7	LVCMOS33	4	KEEPER	Slow
6	disp_RGB1...	output	bank1	P8	LVCMOS25(default)	8	NONE	Slow

其他电平参数的含义及适用范围如下表所示：

参数	适用范围	含义	可选值
SlewRate	单端输出	输出压摆率	Slow, Med, Fast
DiffResistor	差分输入	差分端接电阻 100ohm	100, None
PCIClamp	单端输入、输出	PCI 电平标准要求有箝位	ON, OFF
PackSeq	输入、输出	将寄存器打包到 pad 中 Auto 表示按照全局设定值 ON，OFF 的优先级高于全局设定值	Auto, ON, OFF
OpenDrain	单端输出	关闭输出的上拉 P 管，这时只能输出 0，逻辑 1 依赖外部上拉电阻实现	ON, OFF
InDelay	输入	调节输入延时	实际延时取决于芯片及 IOB 的类型
OutDelay	输出	调节输出延时	实际延时取决于芯片及 IOB 的类型

当所有设置都完成后，用户可单击左上角的保存按钮，输入文件的名称，单击保存。若工程中已添加 adc 文件，打开 IO Constraint 界面时，会读取 adc 文件中的信息，更改设置后进行保存，将直接更新 adc 的内容。若工程中没有添加 adc 文件，保存后，将直接在工程中添加 adc 文件。

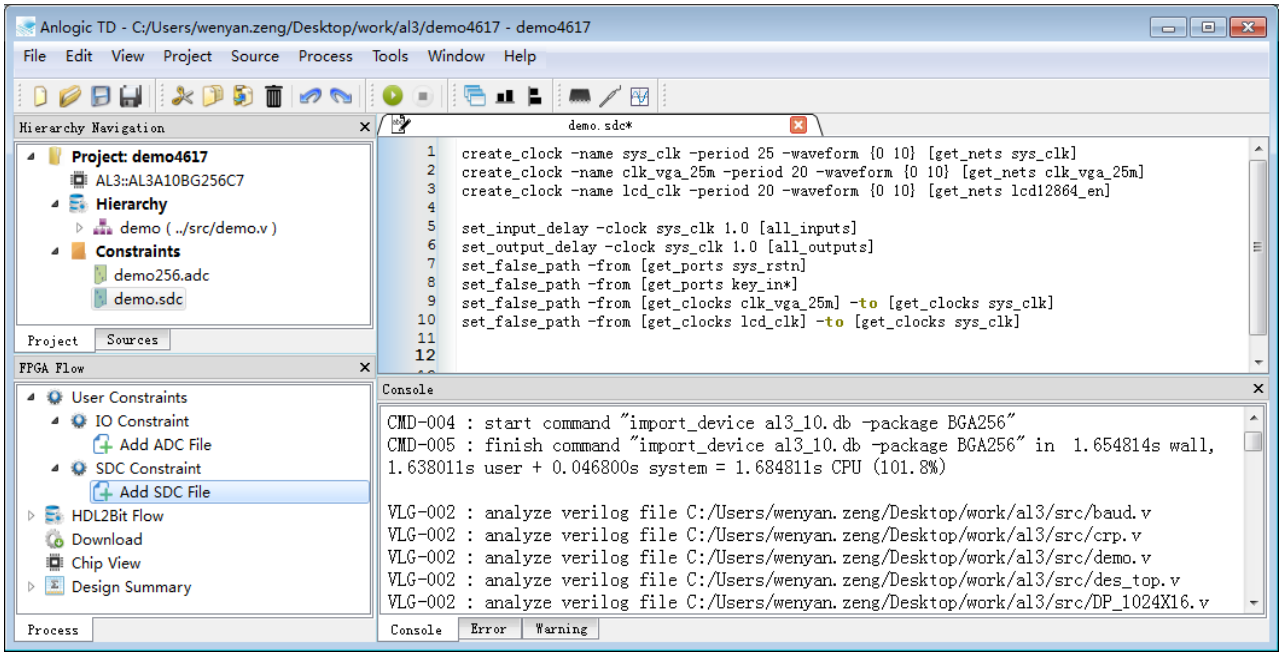


4.2 时序约束

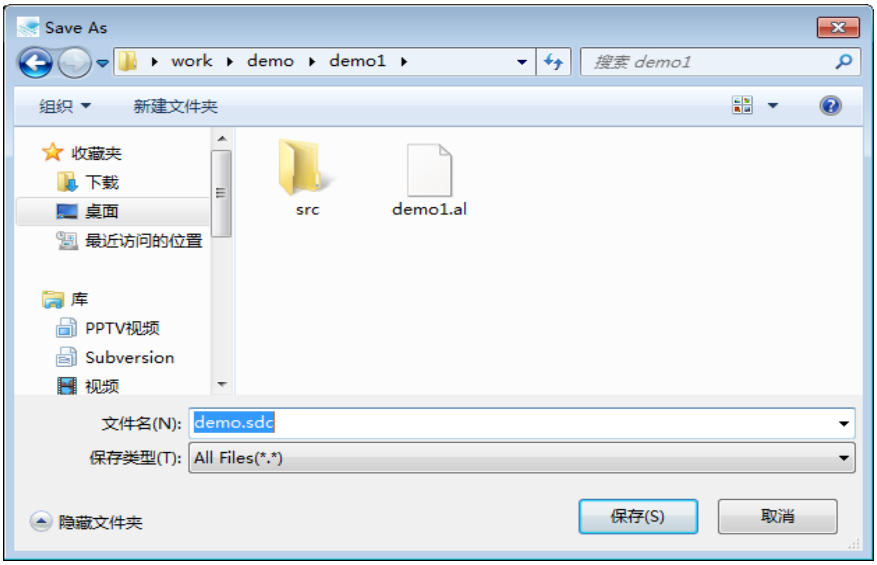
4.2.1 添加时序约束

新建 SDC 文件

1. 点击新建按钮，在新打开的窗口中输入 SDC 命令

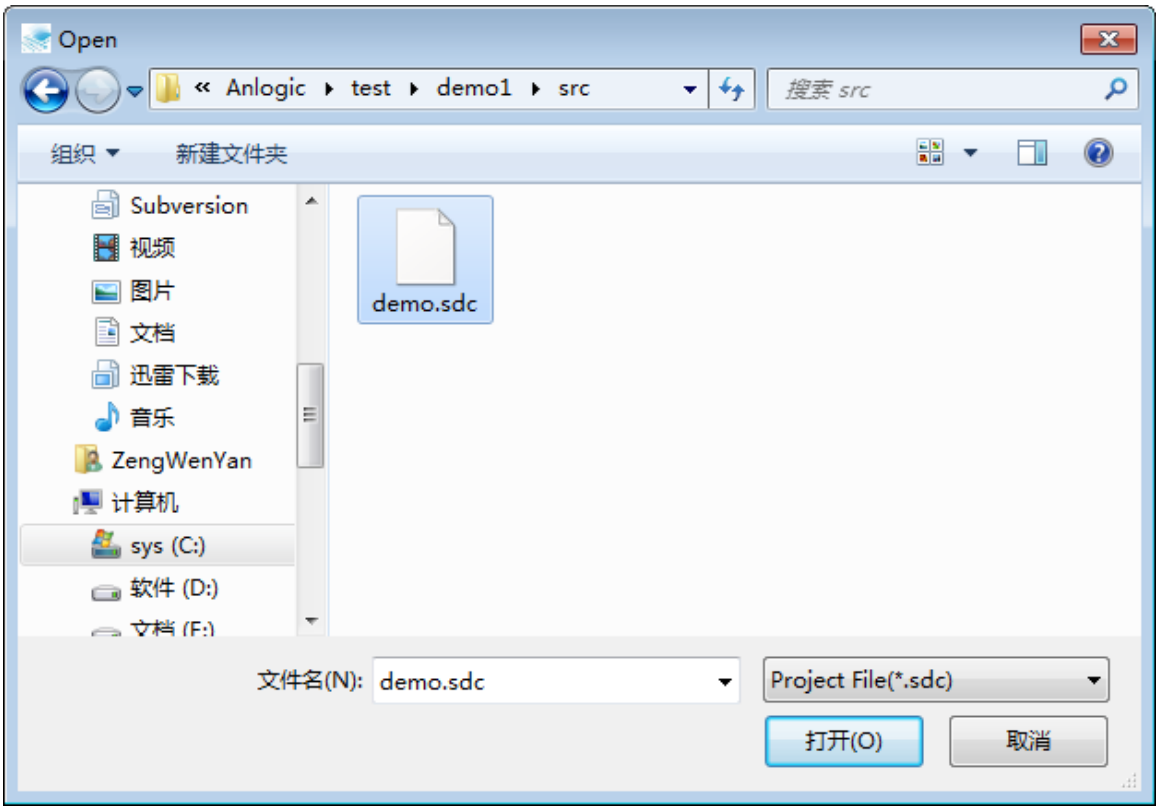


2. 点击保存按钮，将文件路径设置在项目路径下，文件名可设为 demo.sdc，然后单击保存。

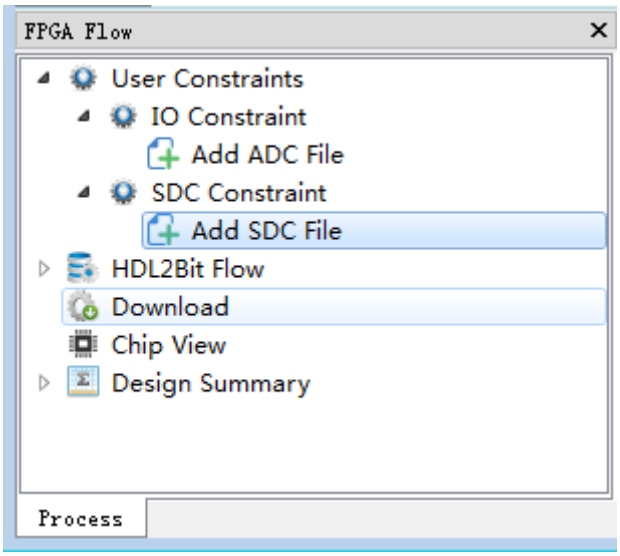


添加SDC文件：

1. 右键单击 Hierarchy Navigation 面板中 Project 里的 **IO Constraints**，选择 **Add SDC File**，选中 demo.sdc，点击打开。



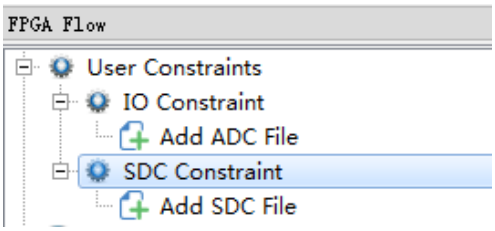
或者展开 FPGA Flow 中的 **User Constraints**，双击 **Add SDC File**，如上图选中 demo.sdc 点击打开。



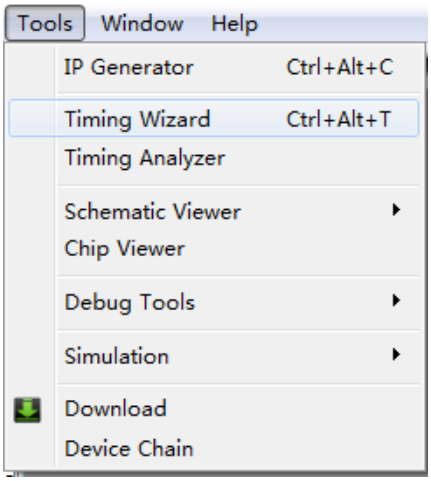
4.2.2 界面设置时序约束

有两种方式可以打开设置时序约束的界面：

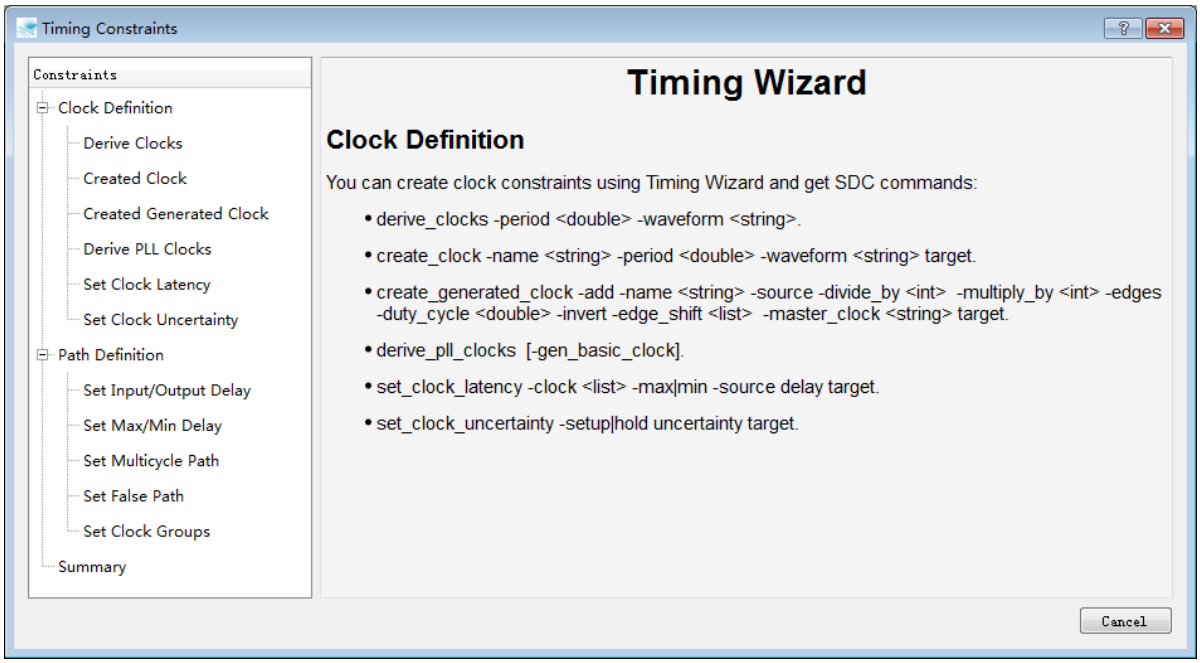
1. 在 FPGA Flow 面板中，展开 User Constraints，双击 SDC Constraint；



2. 在菜单栏中展开 Tools，双击 Timing Wizard，或使用快捷方式 Ctrl+Alt+C。



Timing Wizard 主界面如下图所示：



Timing Wizard 包含三个部分：

1. Clock Definition：该部分主要包含创建或定义时钟约束的 SDC 命令；

2. Path Definition：该部分主要包含创建或定义时序路径约束的 SDC 命令；

3. Summary：该部分总结已创建的时钟和时序路径的所有约束。

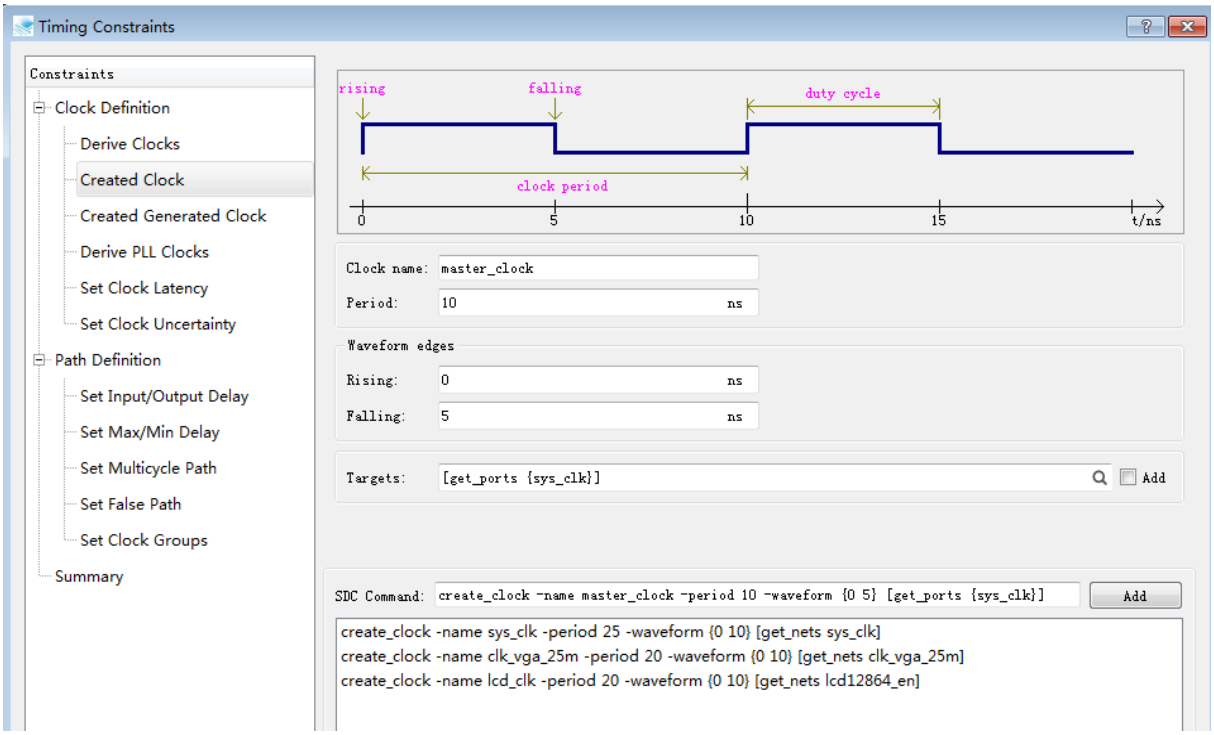
下面详细介绍每条命令的使用方法：

1. Created Clocks

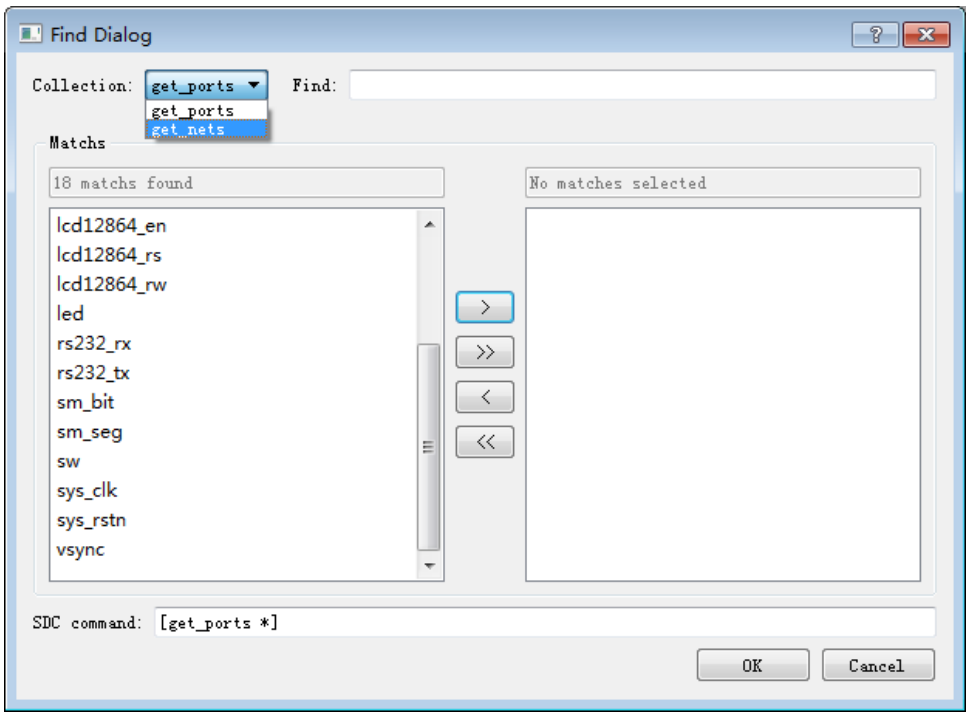
   格式：**create_clock** -add -name <string> -period <double> -waveform <string> target

   定义：定义一个时钟：target 指明了时钟源，可以是 nets 或 ports，如果 target 为空则说明定义了一个虚拟时钟；-name 指定了时钟名，如该项为空则时钟名为 target 列表的第一项；-period 为周期，该选项必须指定，且数值需大于 0；-waveform 指定了第一个上升沿与第一个下降沿的时间点，暂时仅支持每周期包含两个时钟沿的情况；-add 说明在 target 管脚上已经定义过时钟的情况下，将新时钟加入至该管脚，否则覆盖已有时钟，主要用于 clock multiplexer 的情况。

   参数设置如下图所示：



点击 Targets 后方的查找按钮，可选择不同类型的 target。



参数设置好后，点击 Add 将该命令添加至下方的方框，则可继续设置其他参数的该命令，否则该命令将不会添加至 Summary。若工程中已添加 sdc 文件，并且 sdc 文件中已经存在了 created_clock 命令,则会将已有命令也添加至 Summary

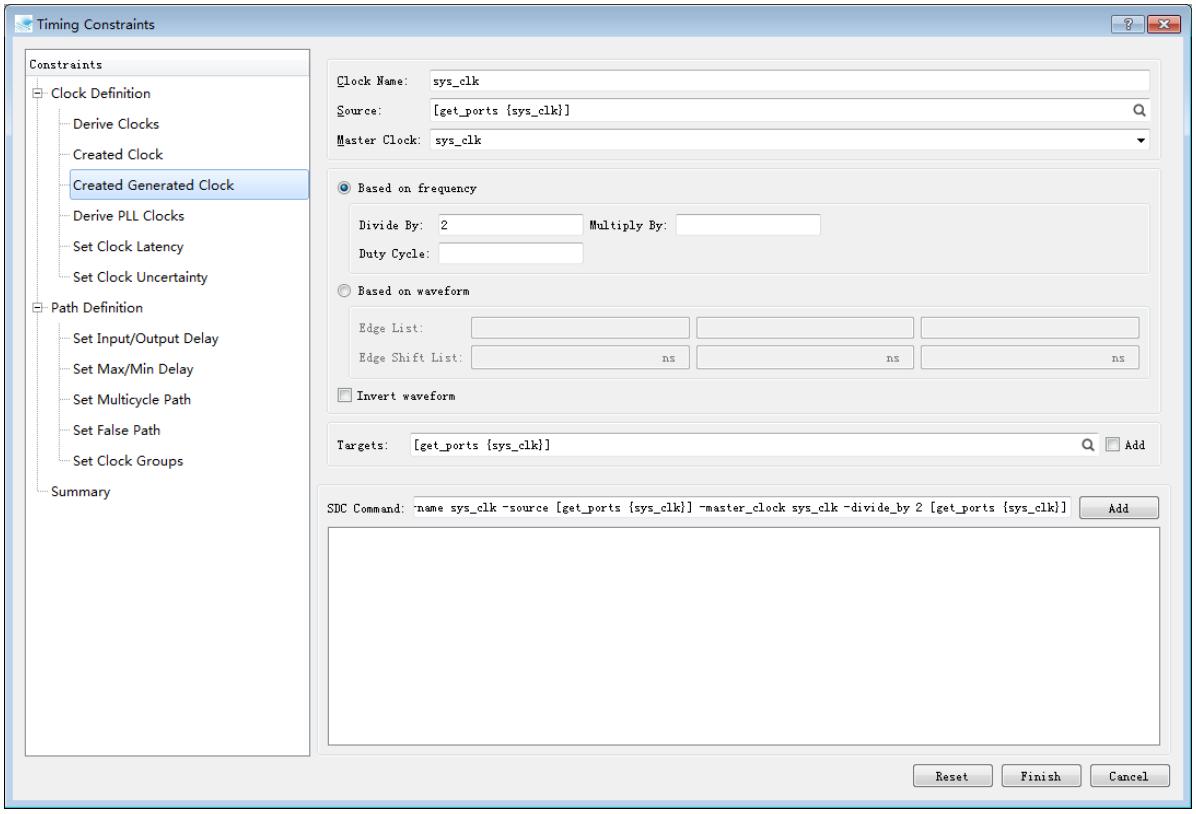
中。

2. Created Generated Clock

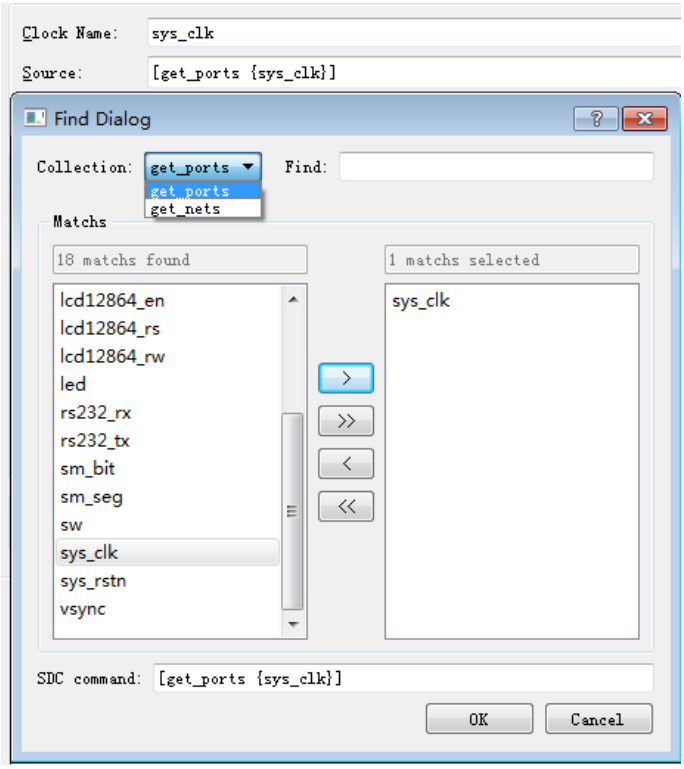
格式：**create_generated_clock** -add -name <string> -source <list> -divide_by <double> -multiply_by <double> -edges <string> -duty_cycle <double> -invert -edge_shift <string> -master_clock <string> target

定义：定义一个派生时钟，属于 master clock 所在的时钟域，在时序报告中也将具有和 master clock 相同的 start point。

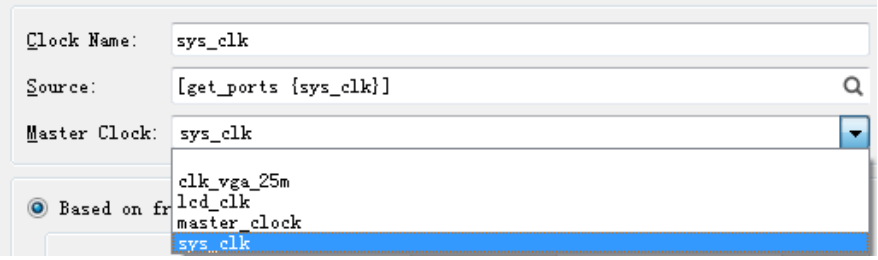
参数设置如下图所示：



-source 指定了其 master clock 所在的源点，可以是 nets 或 ports 的列表；点击 Source 栏后的查找图标，可选择不同类型的 source:

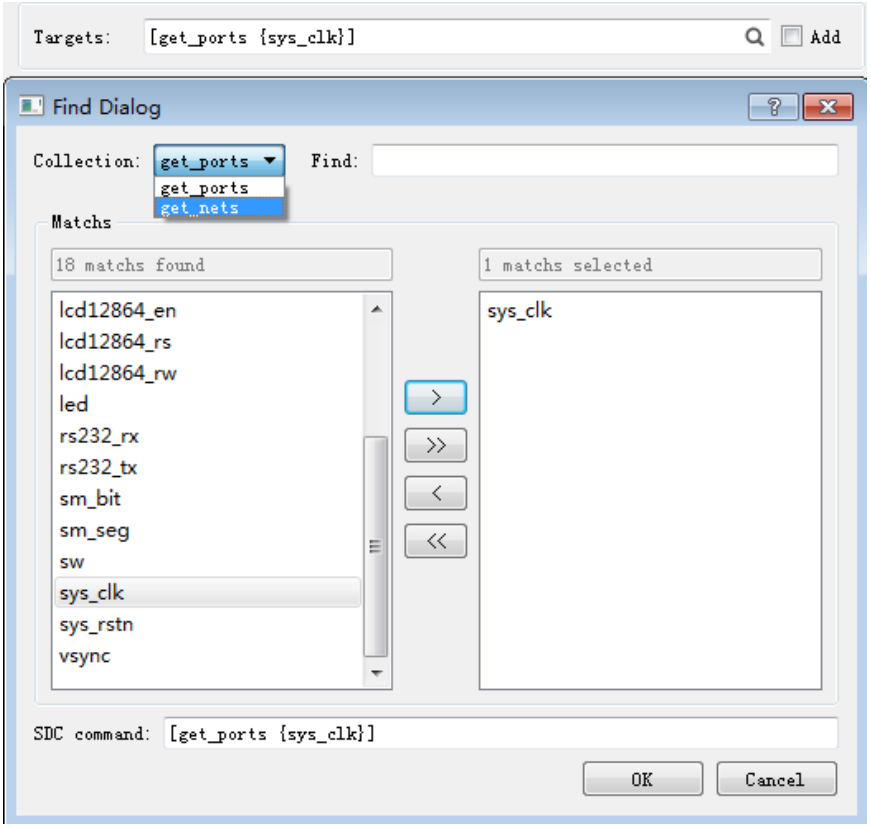


-master_clock 指定了 master clock 的名称，单击 Master Clock 的下拉菜单，可选择已创建的 clock；



-name 指定了时钟名，如果该项为空则时钟名为 source 列表的第一项；target 为当前生成时钟的源点。-add 说明在 target 管脚上已经定义过时钟的情况下，将新时钟加入至该管脚，否则当前生成时钟被忽略，这点与 master clock 定义时不同。

点击 Targets 栏的查找按钮，同样可以添加不同类型的 target。



生成时钟的周期与波形由 master clock 调整而来，-divide_by / -multiply_by 指频率除以/乘以指定倍数，-invert 与这两条选项配合使用，使时钟波形反转，而 -duty_cycle 配合-multiply_by 选项使用，用以调节占空比；默认选择 Based on frequency。

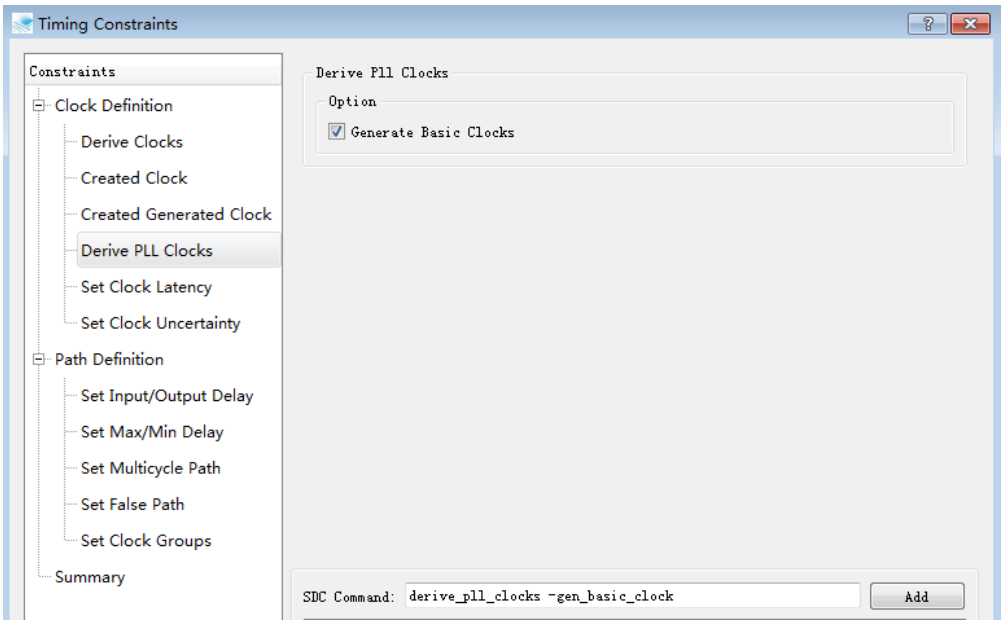
若选择 Based on waveform，则可设置 edge 选项。-edges 选项包含三个整数，分别指定了新生成时钟的第一个上升沿，第一个下降沿，第二个上升沿对应源时钟的第几个边沿；-edge_shift 选项将指定-edges 选项中三条时钟沿的偏移量，因此将包含 3 个正负皆可的整数，单位为基本时间单位（默认为 ns）。



3. Derive PLL Clocks

格式：**derive_pll_clocks** [-gen_basic_clock]

定义：自动在所有用到的 PLL clkc[x]端口生成时钟约束，生成时钟的频率、相位都将严格按照 PLL 内部的参数设定。



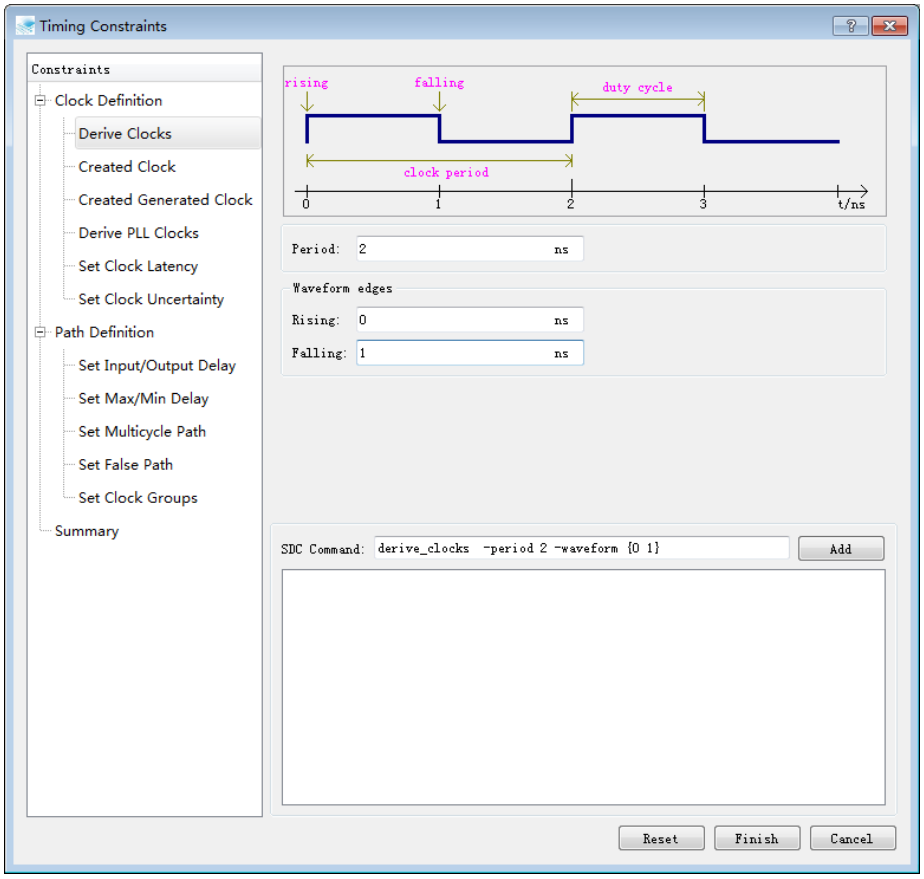
-gen_basic_clock 将在对应的 PLL refclk 上定义 FIN 频率的基准时钟，否则将自动搜索 refclk pin 以及所连 net 上定义的时钟。该命令生成的时钟将在 flow 运行时才生效，因此在 timing wizard 后续的设置中还无法引用。

4. Derive Clocks

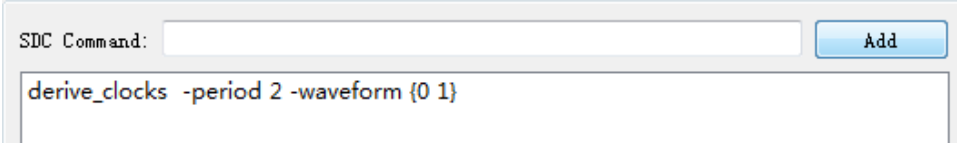
格式：derive_clocks -period <double> -waveform <string>

定义：在各个未定义时钟的时钟管脚上指定一个默认时钟，-period 为周期，该选项必须指定，且数值需大于 0；-waveform 指定了第一个上升沿与第一个下降沿的时间点，暂时仅支持每周期包含两个时钟沿的情况。

参数设置如下图所示：



点击 Add 将该命令添加至下方的方框，则可继续设置其他参数的该命令，否则该命令将不会添加至 Summary。



5. Set Clock Latency

格式：**set_clock_latency** -clock <list> -max -min -source delay

定义：设定时钟的延时，delay 为延时的数值；

-clock 指定时钟的列表；

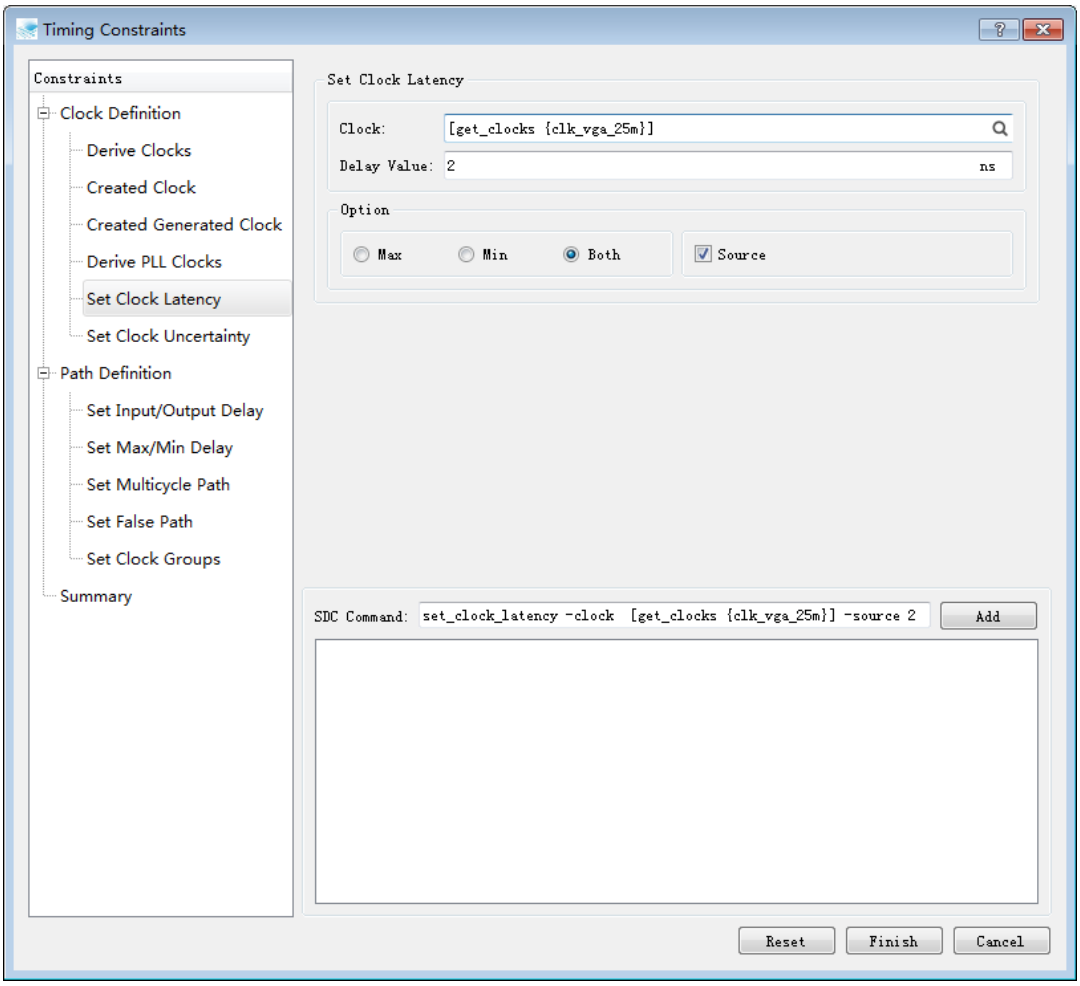
-max/ -min 选项指定本命令所指定的为最大/最小延时，默认为两者相同；

-source 选项说明指定的为 source latency，否则为 network latency。

network latency 在布线后的时序分析中，将被实际的互联延时所取代。

单击 Clock 栏后的查找按钮，可指定已创建的 clock。

单击 Add 按钮将该命令添加至 Summary。



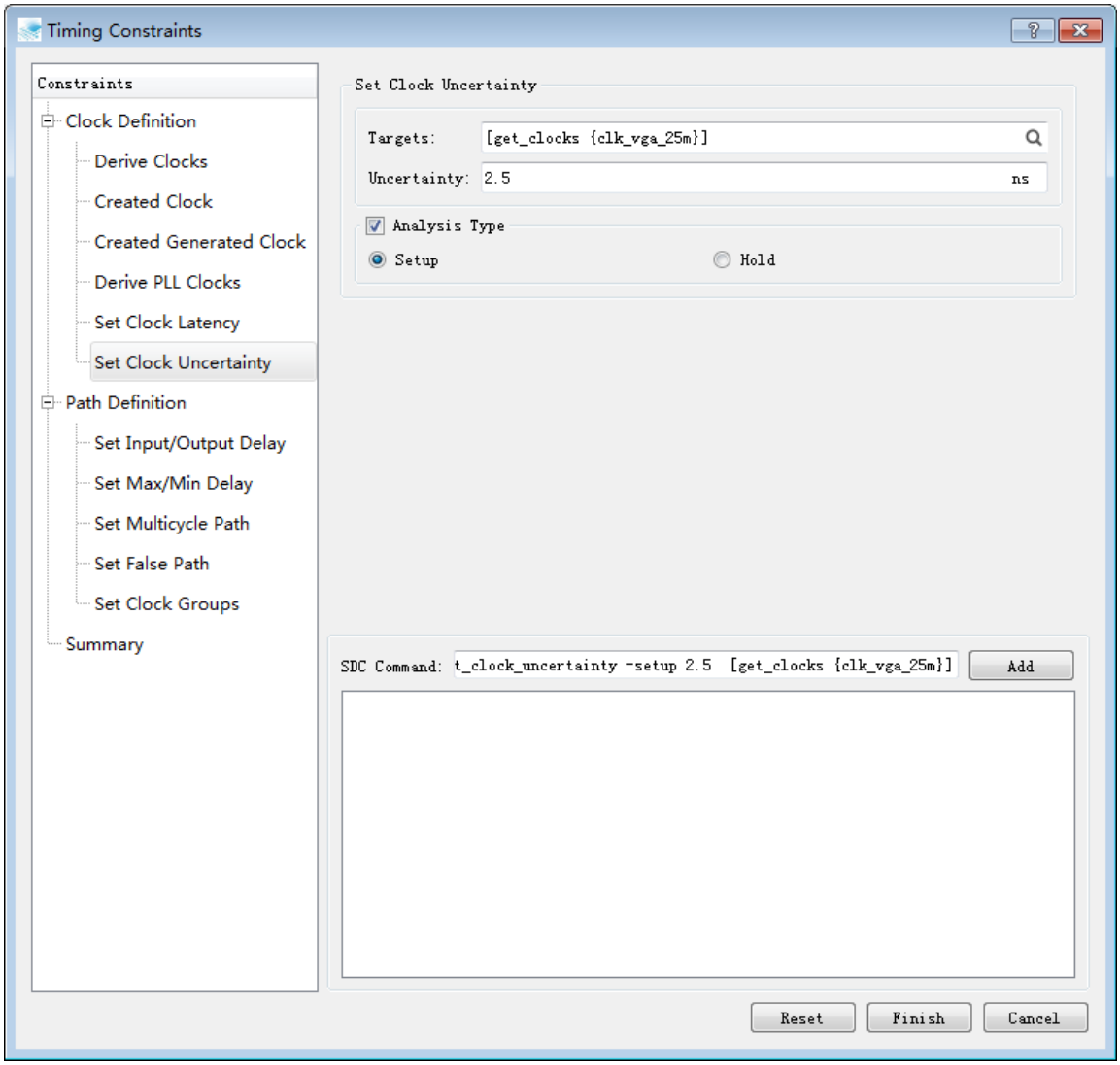
6. Set Clock Uncertanty

格式：**set_clock_uncertainty** -setup -hold uncertainty target

定义：设定时钟的 uncertainty 数值，目前仅支持对于单个时钟本身设置而不支持跨时钟域的 uncertainty 定义；target 为 clock 的列表，uncertainty 为数值项；-setup/-hold 选项指明本命令所对应的是最大/最小路径时序分析时所对应的数值，如果该选项没有指明，则对两种检查同时生效。

单击 Add 将该命令添加至 Summary。

参数设置如下图所示：

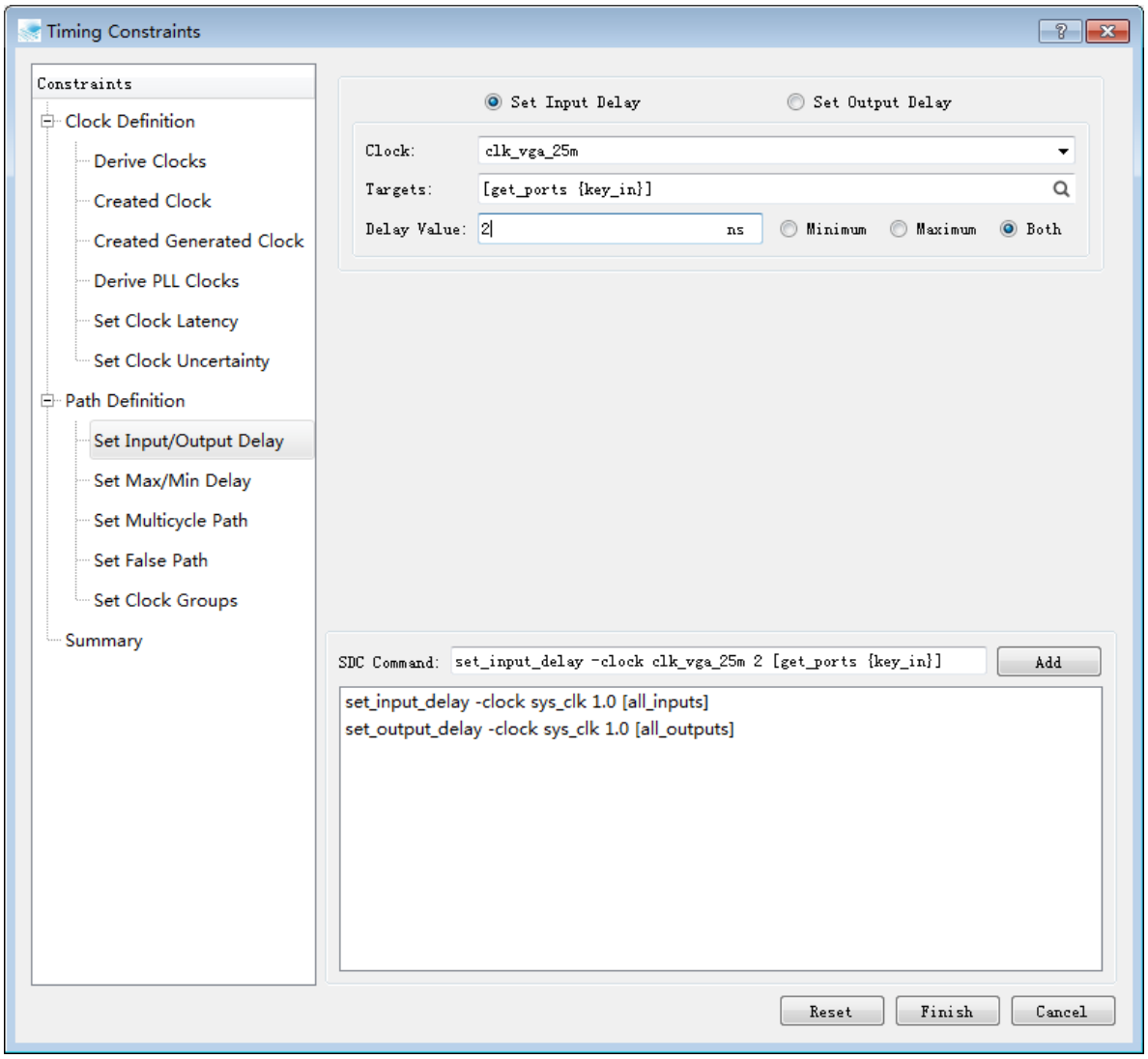


7. Set Input/Output Delay

格式：**set_input_delay / set_output_delay** -clock <list> -max -min delay target

定义：设置输入/输出端口的延时，delay 项为延时数值，target 可以是 pins 或 ports 的对象列表；-clock 指定了延时所对应的时钟信息；-max/-min 选项则指定了本命令所设置的值为最大/最小延时，默认为相同。

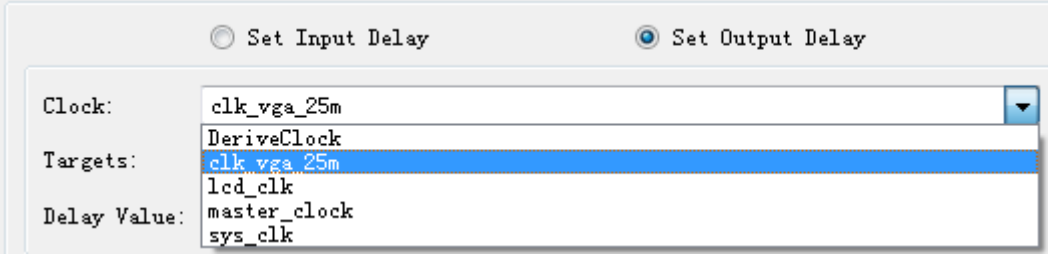
参数设置如下图所示：



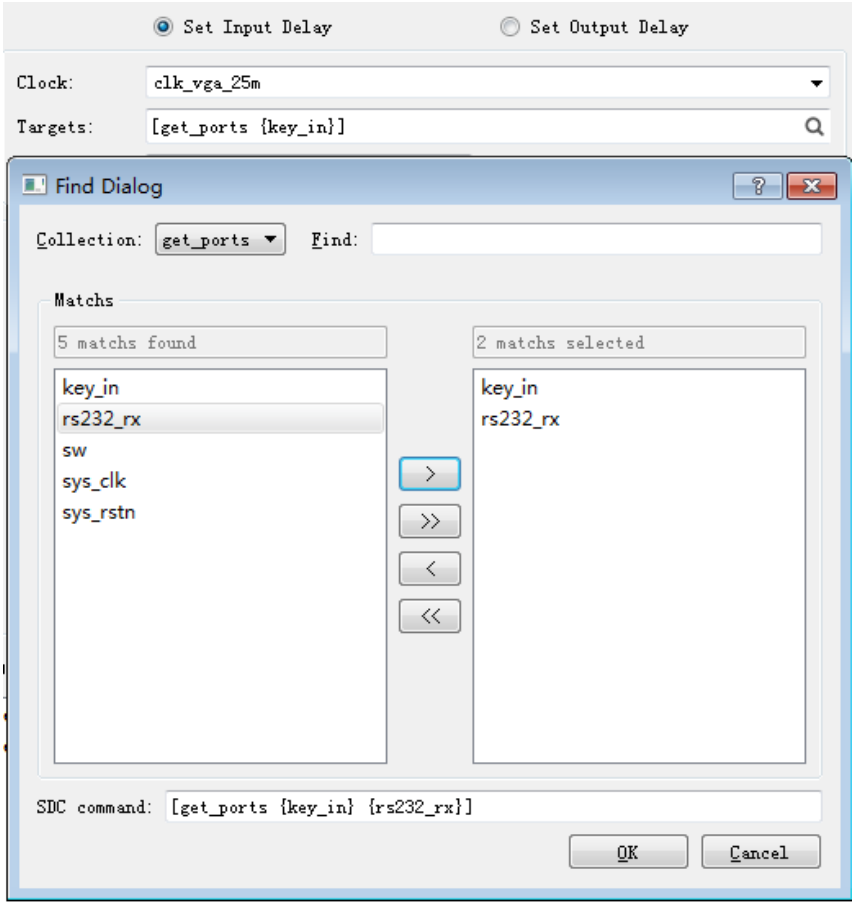
默认为设置 input delay 相关参数；

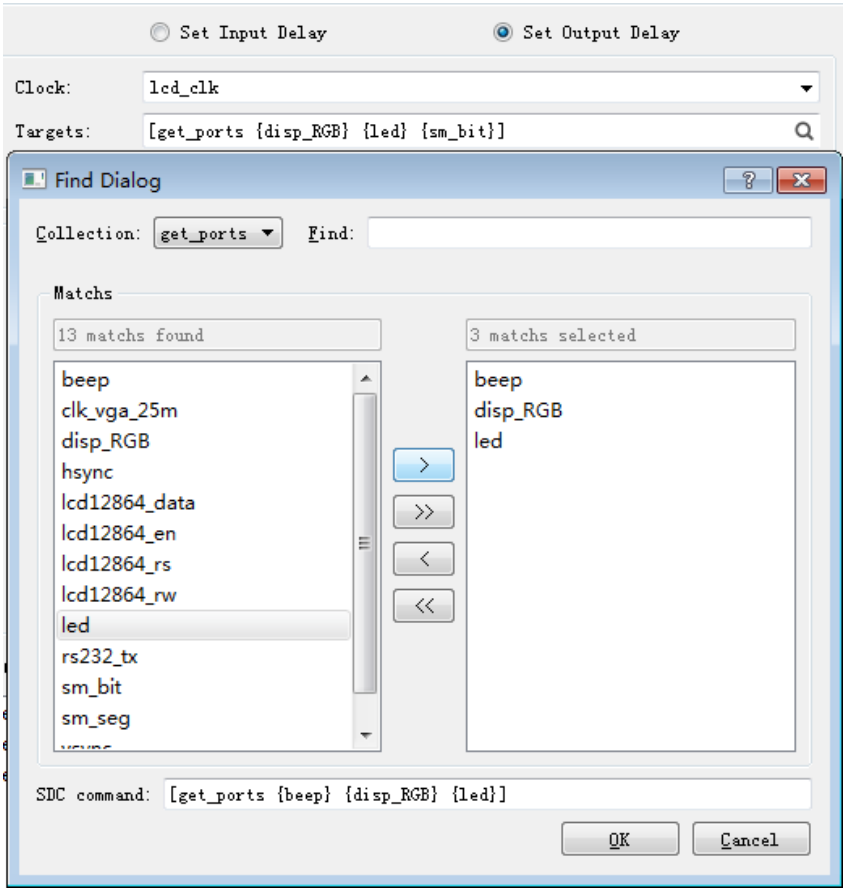
若需设置 output delay 相关参数，需选择 Set Output Delay，参数设置同 input delay。

单击 Clock 栏的下拉菜单，可指定已创建的时钟。

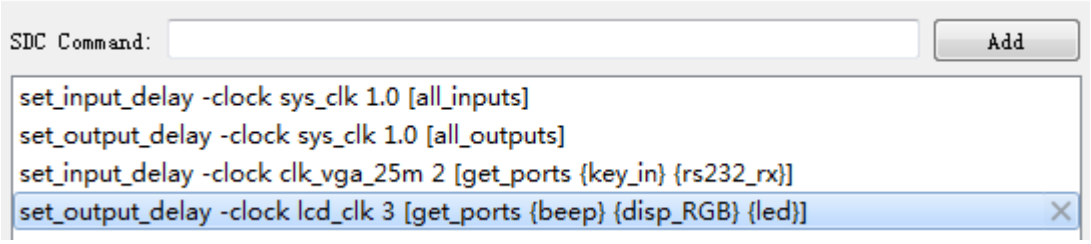


单击 Targets 栏的查找按钮，可指定相应的 ports，若选择的为 Set Input Delay，则仅有 input/inout ports 可见；若选择的为 Set Output Delay，则仅有 output/inout ports 可见。





单击 Add 将命令添加至 Summary。



若要删除已经生成的 SDC Command，可在 Summary 中选中该命令，点击行末的删除按钮“X”。

8. Set Max/Min Delay

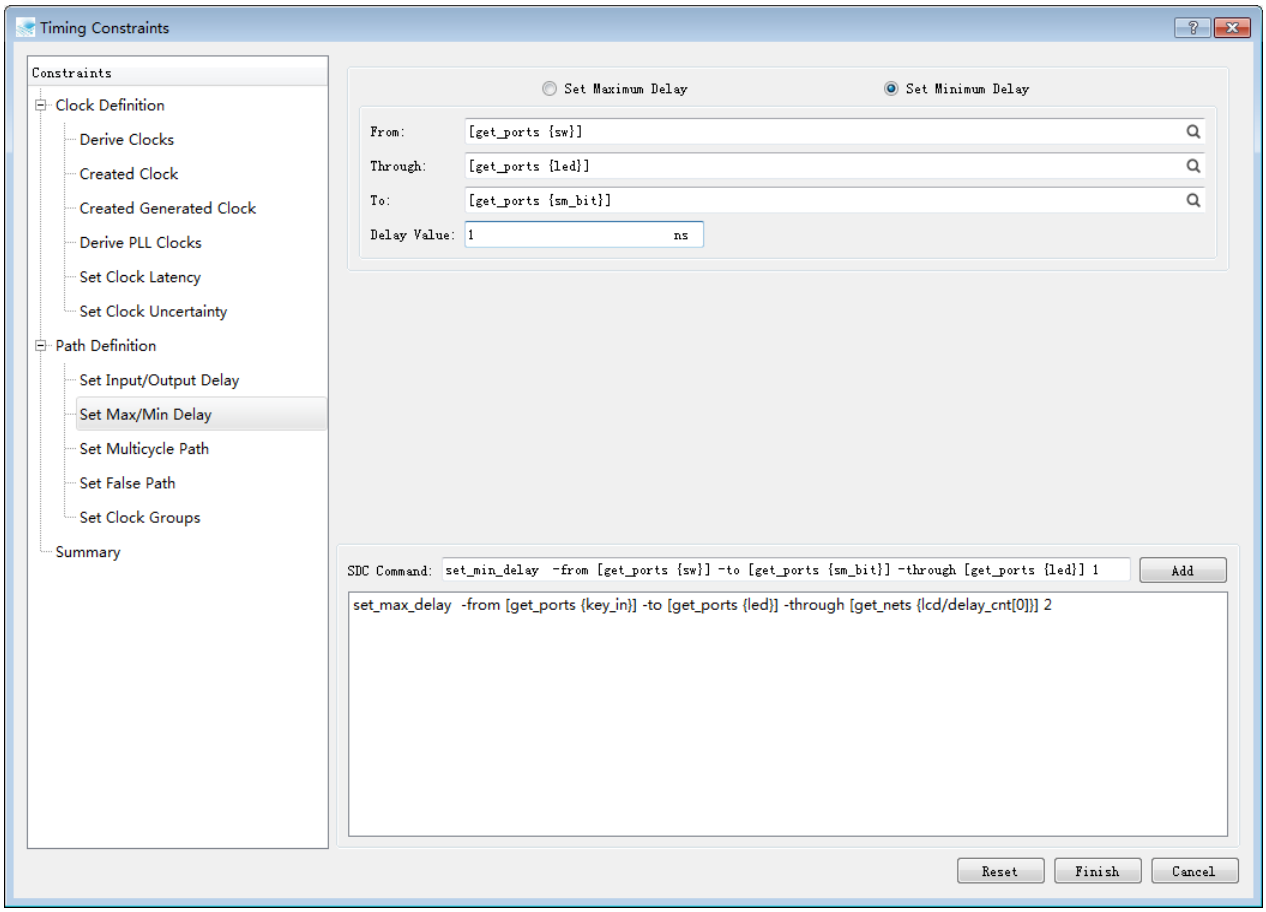
格式：**set_max_delay / set_min_delay** -from <list> -to <list> -through <list> delay

定义：设定时序路径的允许最大/最小延时，delay 项为延时数值；-from 必须为时序路径的起点，即 input 的列表；-to 必须为时序路径的终点，即 output 的列表；-through 选项可以是 nets，ports 列表，指定了该时序路径必须通过的中间

点，当有多个 through 选项时，目标时序路径必须依次经过每一个中间点。

参数设置如下图所示：

默认为 Set Maximum Dealy，若要设置 Minimum Delay，需选择 Set Minmum Delay，两者参数设置相同。



单击 Add 将设置好的命令添加至 Summary。

9. Set Multicycle Path

格式：**set_multicycle_path** -setup -hold -start -end -from <list> -to <list> -through <list> multiplier

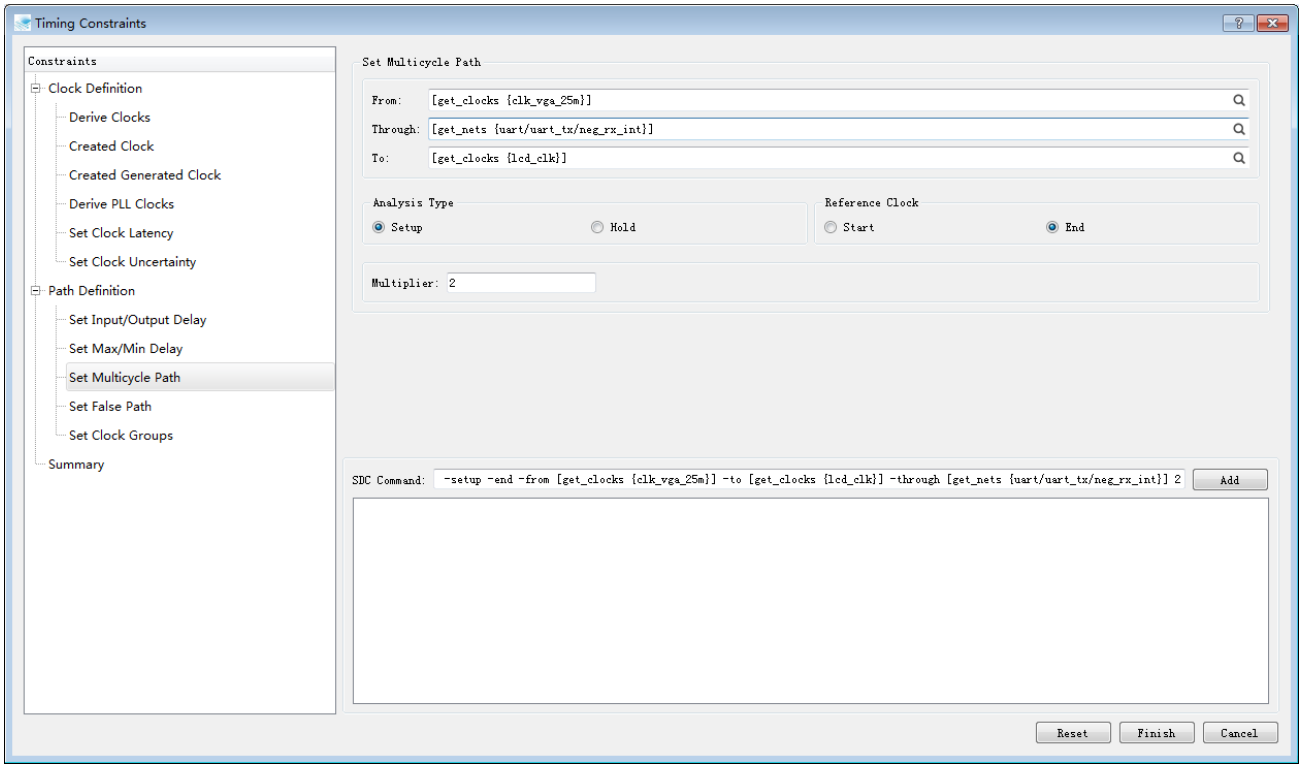
定义：设置允许多个时钟周期延时的时序路径，multiplier 项为时钟周期数；-setup/-hold 选项设定该命令所约束的时序路径类型，仅使用-setup 选项时，setup check 将允许时序路径最多使用 N 个时钟周期，但同时 hold check 会变为要求时序路径最短经过 N-1 时钟周期。仅使用-hold 选项时，则将 hold check 的约束设

置为 N 而不影响 setup check 的约束。

在常规使用场景下，为了让 setup check 能使用多个时钟周期而不影响 hold check，需要两条命令搭配使用，即设定一条 N 个周期的 setup 约束，再配合一条 N-1 个周期的 hold 约束。

-start/-end 为跨时钟域时使用的选项，指定延时为 launch/capture 时钟对应的周期，默认情况下，setup check 使用 capture 时钟而 hold check 使用 launch clock。-from 为时序路径的起点，可以为 clocks 或 inputs 的列表，-to 为时序路径的终点，可以为 clocks 或 outputs 的列表，-through 指定了该时序路径必须通过的中间点，可是是 ports 或 nets 的列表。

参数设置如下图所示：



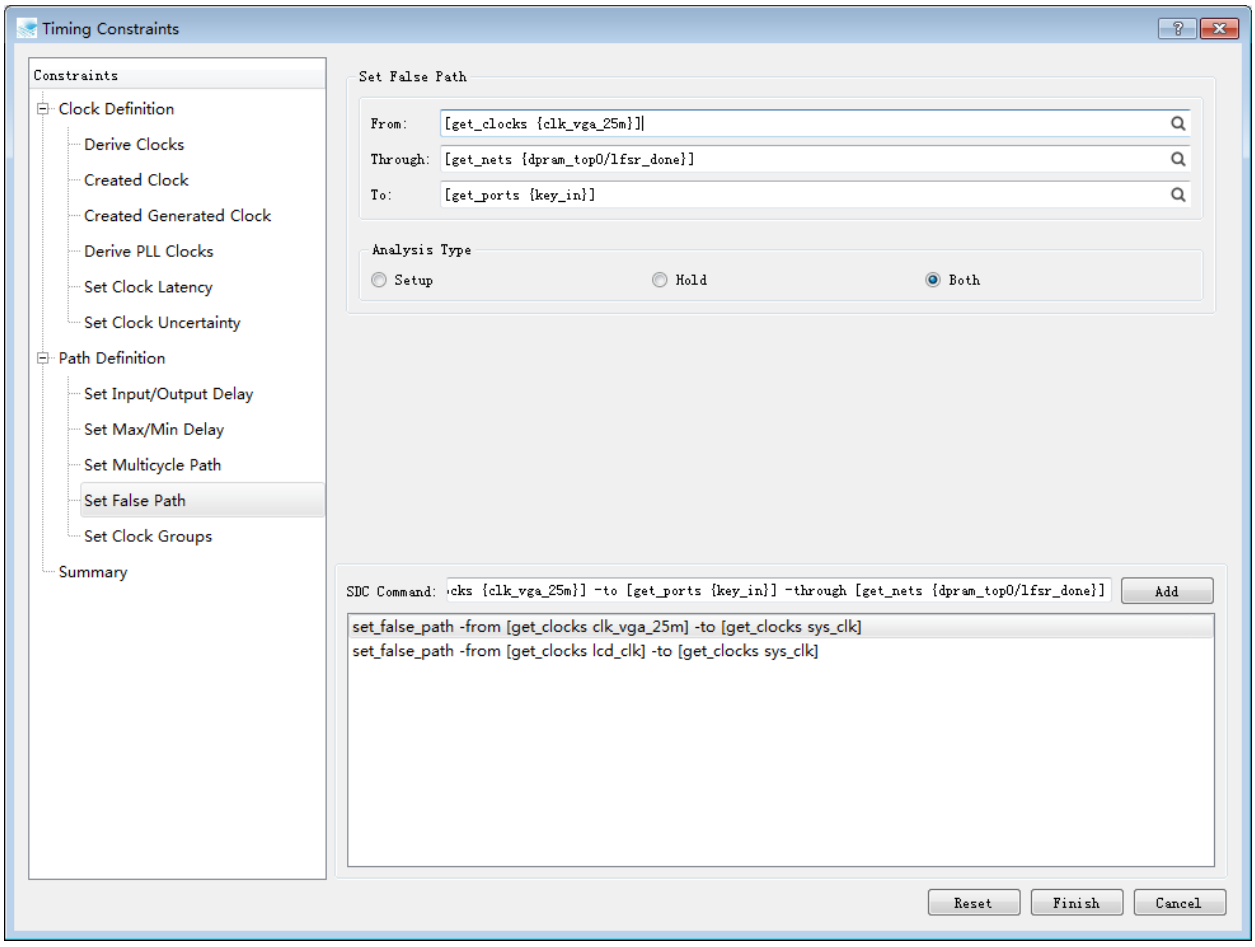
点击 Add 将生成的命令添加至 Summary。

10. Set False Path

格式：**set_false_path** -setup -hold -from <list> -to <list> -through <list>

定义：设定时序路径为虚假路径，因而不对其进行时序分析；-setup/-hold 选项指定了在 setup/hold check 时不分析目标路径。-from 为时序路径的起点，可以是 clocks 或 inputs 的列表，-to 为时序路径的终点，可以是 clocks 或 outputs 的列表，-through 指定了该时序路径必须通过的中间点，可是是 ports 或 nets 的列表。

参数设置如下图所示：



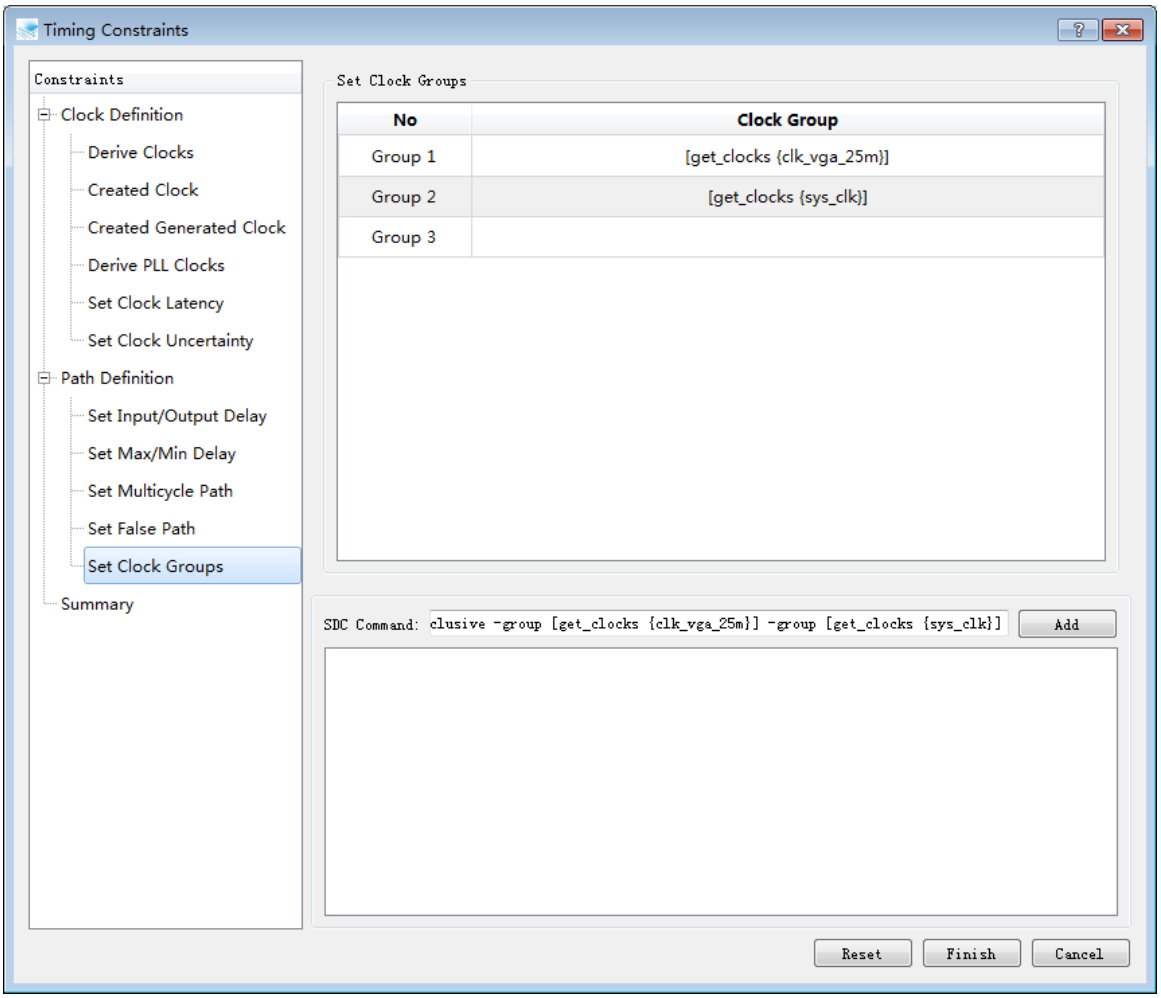
点击 Add 将生成的命令添加至 Summary。

11. Set Clock Groups

格式：**set_clock_groups** -exclusive -asynchronous -group <list>

定义：为时钟域设定分组，不分析跨时钟组的时序路径。一般来说，-exclusive 选项表示这些时钟组在逻辑上不会同时出现，而-asynchronous 表示完全不相干的时钟，不过在具体实现以及效果上，这两个选项是一样的，该命令会在所有 -group 列出的时钟之间定义 false path 约束集合。

参数设置如下图所示：

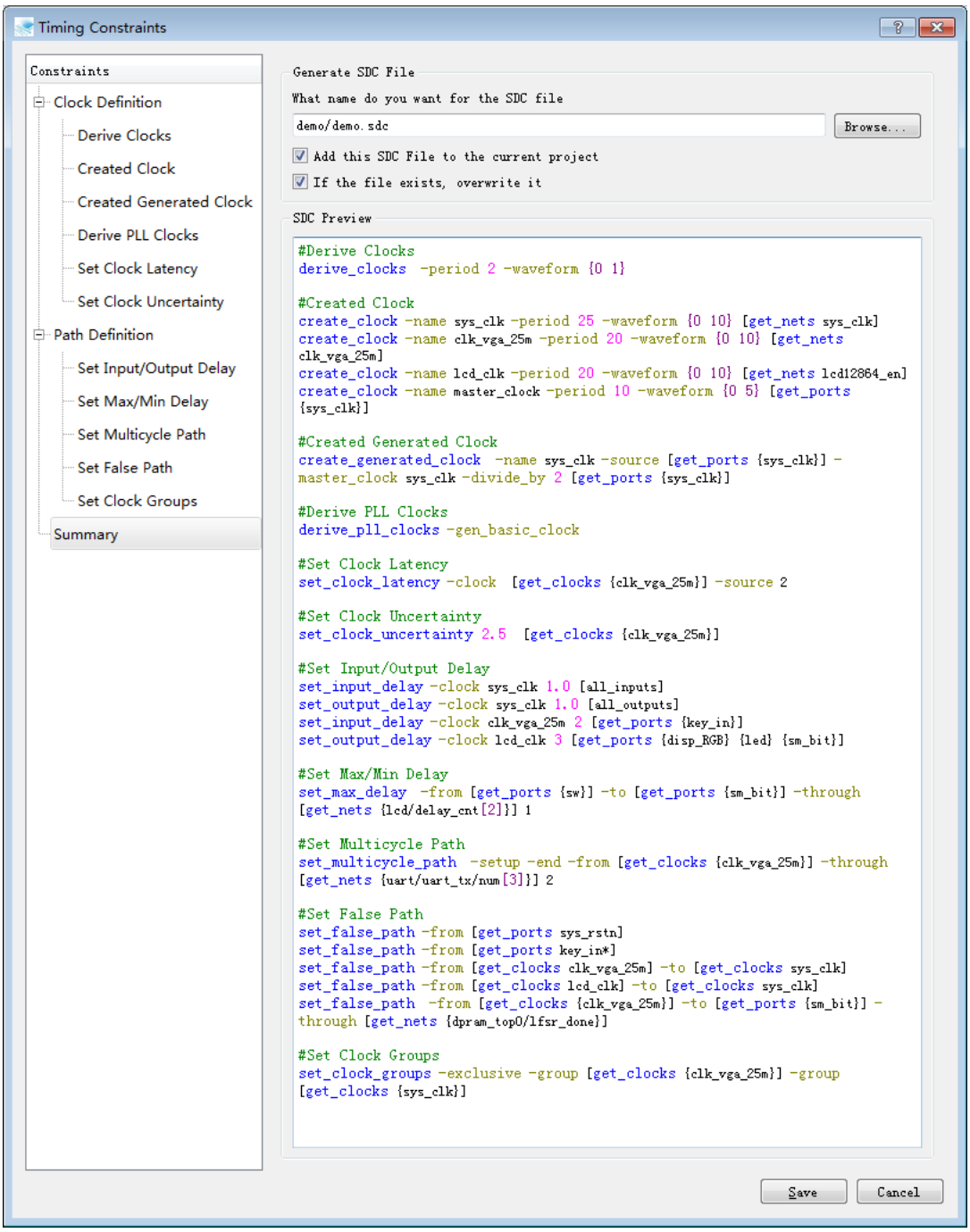


点击 Add 将生成的命令添加至 Summary。

12. Summary

Summary 中记录了所有已生成的命令。

指定文件保存的路径，并选择是否添加至当前工程，点击 Save，完成各命令的设置。注意，如果原工程中已添加 sdc 文件，请谨慎勾选“If the file exist, overwrite it”。该选项勾选后，将会替换掉原有文件。



5 HDL2Bit 流程

在输入设计源文件和约束文件后，下一步进入 HDL2Bit 的设计实现流程。HDL2Bit 流程包括设计读入（**Read Design**）、RTL 级优化（**Optimize RTL**）、门级优化（**Optimize Gate**）、布局优化（**Optimize Placement**）、布线优化（**Optimize Routing**）和生成位流（**Generate Bitstream**）六个步骤。

在多数情况下，用户只要双击 **HDL2Bit**，软件自动运行全部流程。用户也可以用 Process 下拉菜单中的 **Run**，**Rerun**，**Stop** 来控制。Process 菜单中的 **Run** 和 **Stop** 在导航栏中有相应的按钮（和），用户可以直接点击操作。Process 菜单中还有 **Properties** 栏目，**Properties** 提供 **HDL2Bit** 流程中主要步骤的详细参数控制选项，用户可对整个运行流程进行微调控制，Global Option 的参数设置如下：

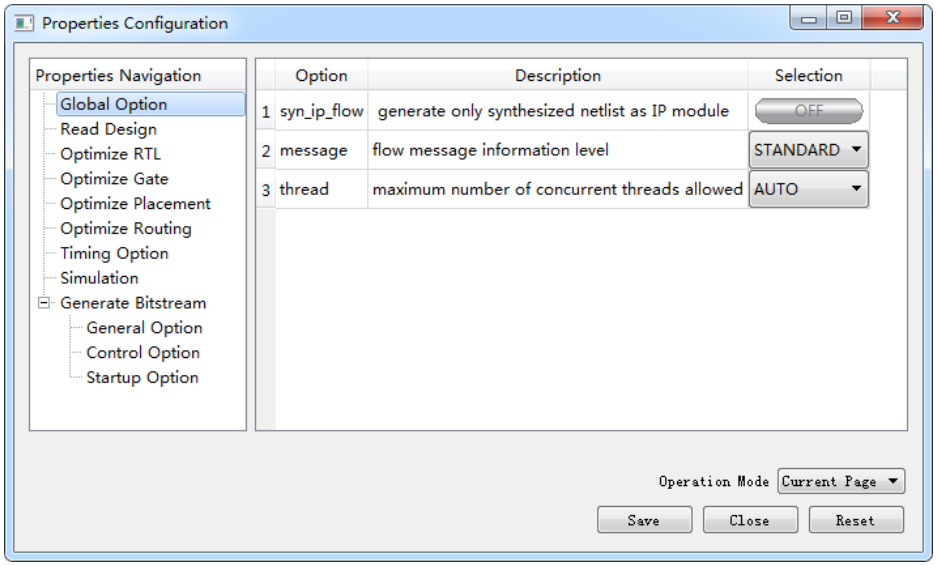


表 5-1 Global Option

Property	Comments	Default
syn_IP_flow	生成可综合的 IP 模块	OFF
message	输出信息的冗余级别	STANDARD
thread	运行 HDL2Bit 时的线程数	AUTO

5.1 读入文件

该步分析用户源文件的语法语义的正确性，并产生原始行为级电路结构。

1. 在 FPGA Flow 面板中展开 **HDL2Bit**

2. 双击 **Read Design**，或右键单击 **Read Design**，选择 **run**

3. 参数配置

在菜单栏中，展开 **Process→Properties**，弹出 Properties Configuration 窗口，选择 **Read Design**，用户可根据需要自定义想要的功能。

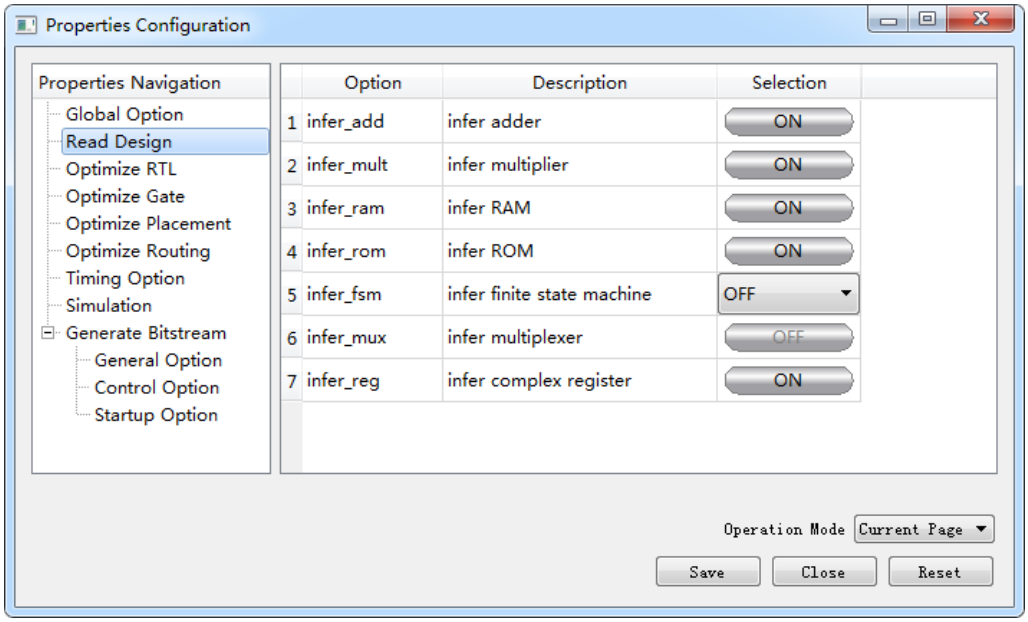


表 5-1 Read Design Properties

Property	Comments	Default
infer_add	自动识别行为级加法描述	ON
infer_mult	自动识别行为级乘法描述	ON
infer_ram	自动识别行为级 RAM 描述	ON
infer_rom	自动识别行为级 ROM 描述	ON
infer_fsm	自动识别行为级有限状态机描述	OFF
infer_mux	自动识别行为级多路选择器描述	OFF
infer_reg	自动识别行为级复杂寄存器描述	ON

5.2 RTL 级优化

在读入设计文件后，TD 将对设计进行 RTL 级优化。本阶段将进行多路选择器优化、数据通路优化、特殊功能模块自动识别等。RTL 级优化将产生包含基本门(AND、OR、FF/Latch)和特殊功能模块的电路。

1. 在 FPGA Flow 面板中展开 **HDL2Bit**
2. 双击 **Optimize RTL,** 或右键单击 **Optimize RTL**，选择 **Run**，此时将会产生面积报告文件 rtl.area，该文件的内容将与 keep_hierachy 参数的设定有关。
3. 参数配置

   在菜单栏中，展开 **Process→Properties**，弹出 Properties Configuration 窗口，选择 **Optimize RTL** 进行设置。

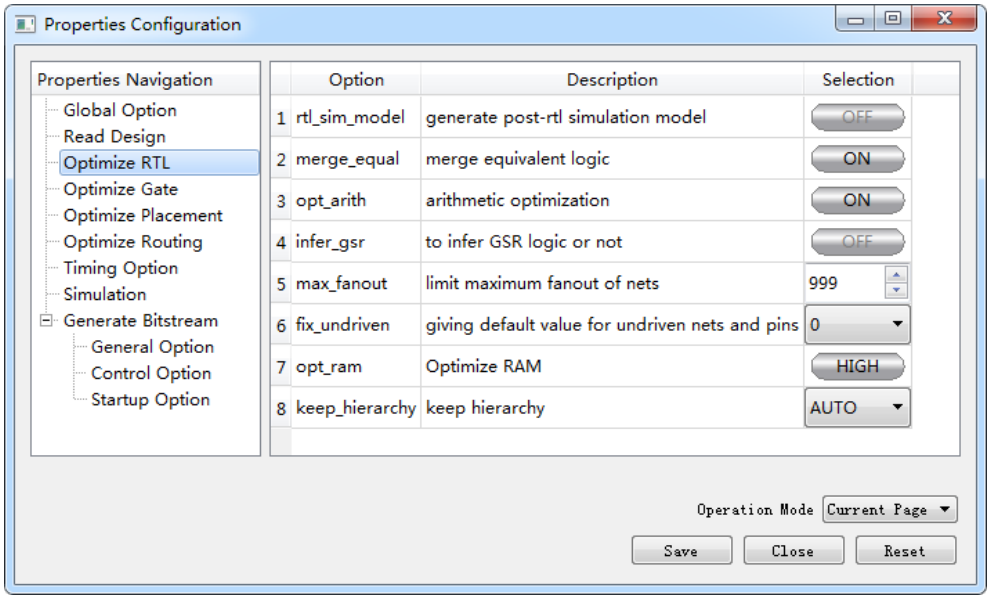


表 5-2　Optimize RTL Properties

Property	Comment	Default
rtl_sim_model	生成 RTL 级电路仿真模型	OFF
merge_equal	合并功能等价的逻辑模块	ON
opt_arith	算术优化	ON
infer_gsr	全局 gsr 优化	OFF
max_fanout	限制逻辑线网的最大扇出数	999
fix_undriven	对于用户没有赋值的线网或者端口，给与常值 0/1	0
opt_ram	回路初始化的多阵列优化成 dram	HIGH
keep_hierarchy	对用户通过 synthesis directive 选择的模块保留层次	MANUAL

*keep_hierarchy 选项说明：选择 flatten 时，td 将把用户设计打平处理；当选择 manual 时，td 将对用户通过 synthesis directive 选择的模块保留层次；当选择 auto 时，td 会自动选择较大的模块保留层次，其余部分打平处理。

5.2.1 Synthesis Keep

使用 Synthesis keep 命令可以保证信号不会被后续流程优化掉,从而方便用户后期调试。具体做法如下:

1. 在 verilog 文件中，为想要保留的信号，添加相应的注释，注释的写法为:

   //synthesis keep = 1;　　/*synthesis keep=1*/;
   //synthesis keep = true;　　/*synthesis keep=true*/
   //synthesis keep;　　/*synthesis keep*/

   如:

```
module test_keep (clk, a, b, c, out);

input clk;

input [3:0] a;
input [3:0] b;
input [3:0] c;
output reg [3:0] out;

wire [3:0] sig; //synthesis keep

assign sig  = a & b;

always@(posedge clk)
begin
   out <= sig | c;
end

endmodule
```

2. 保存文件并编译;

3. 在使用 debug 工具时，可查看到该内部信号。

4. 在 vhdl 中的写法如下:

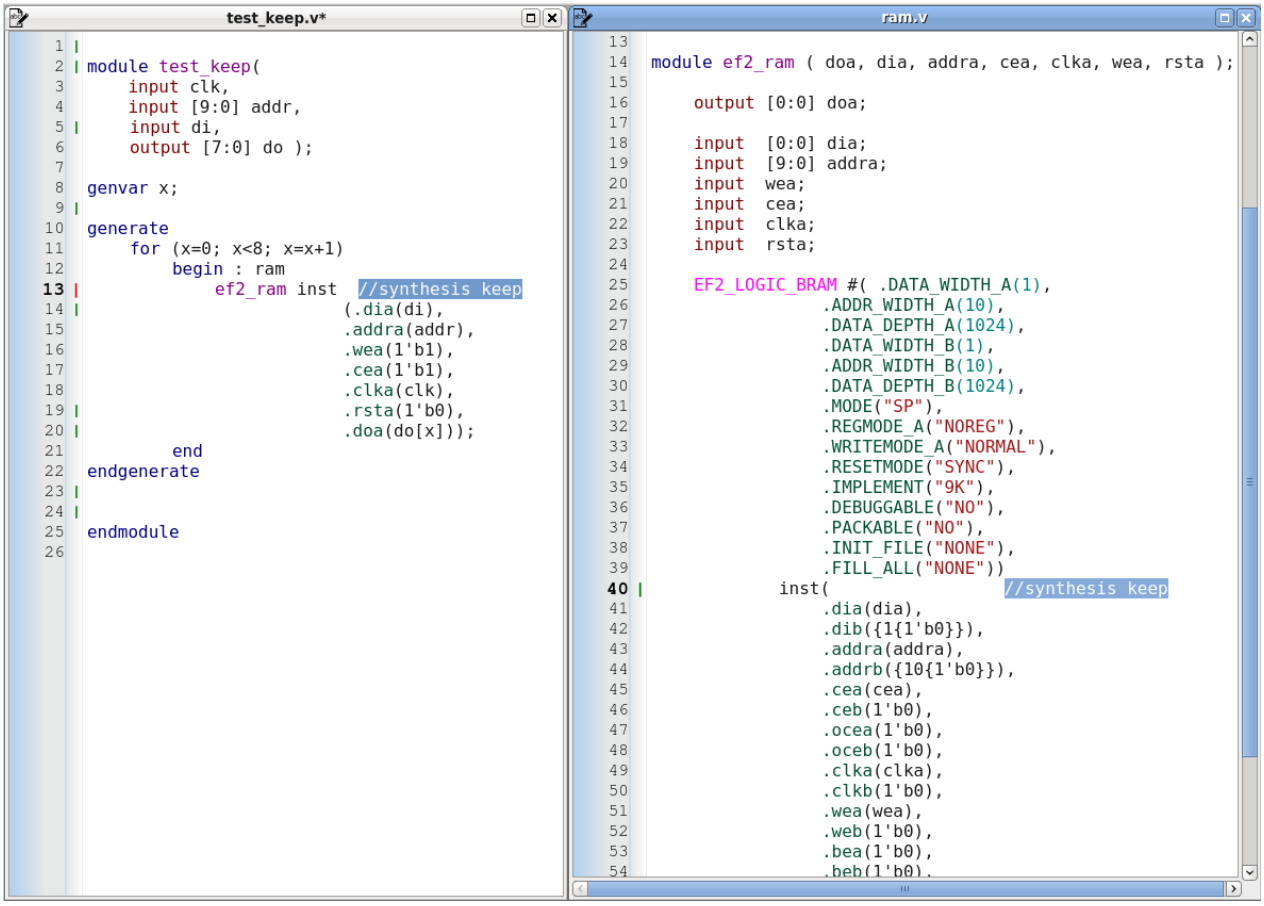
```
attribute keep : BOOLEAN;
attribute keep of clkc_wire : signal is TRUE;
```

其中，clkc_wire 为需要 keep 的 net / bus。

如果想要保留 design 中多个相同的模块不被优化掉，TD 提供了 keep instance 的方

法。

如下图中的模块 ef2_ram，在 top module 中被完全等价地例化了多次，如果此时不使用 synthesis keep 的功能，则所有 instance 将会被合并成一个 BRAM。要想保留所有的 instance 不被优化，需要在该 instance 的每一个层级都添加注释 “//synthesis keep”。



5.2.2 Synthesis Directive

在 keep hierarchy auto/manual 的模式下，可以在 HDL 中添加 synthesis directive 命令来指定特定模块的层次是否要保留。Flatten 模式下，synthesis directive 无效。

具体做法如下：

1. 为想要保留的模块，在源文件中添加相应的注释，注释的写法为：

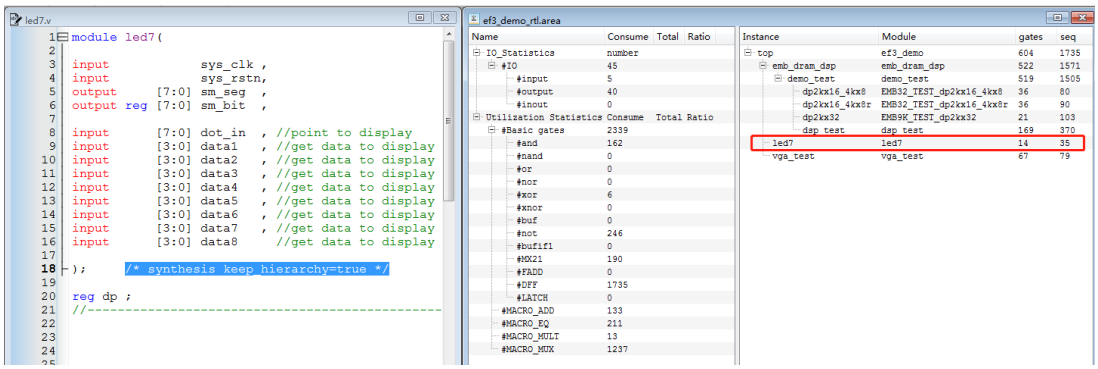
1）保留整个 module

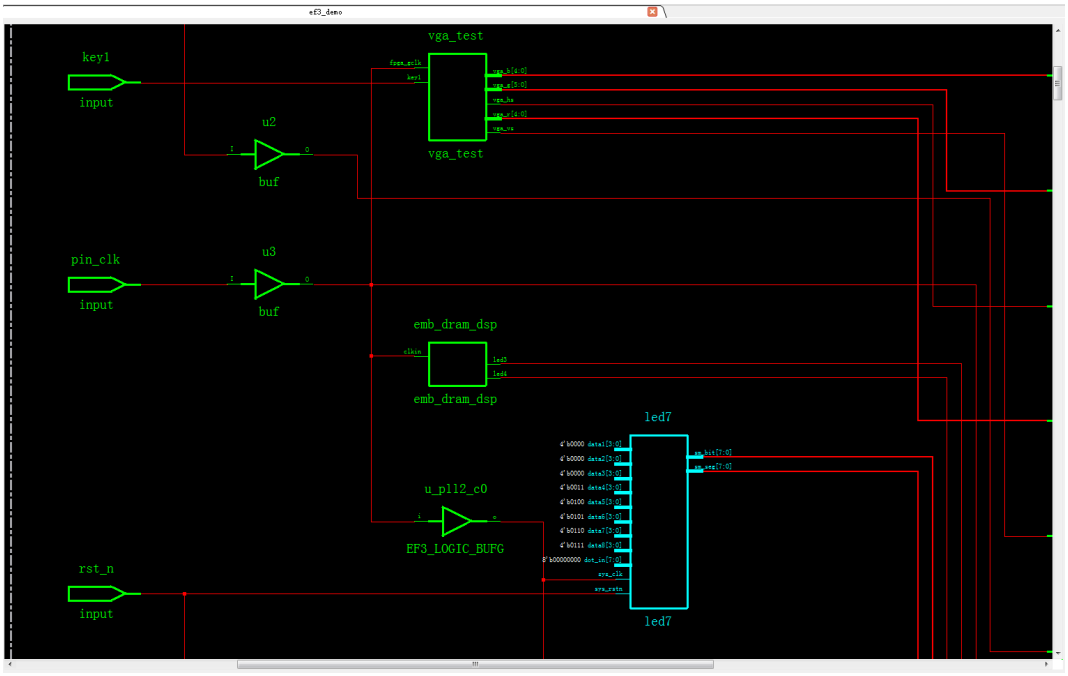
Verilog：

```
module SomeModule(
...
…
);              /* synthesis keep_hierarchy=true */
....
endmodule
```

VHDL：

```
architecture rtl of SomeModule is
    attribute keep_hierarchy : string;
    attribute keep_hierarchy of rtl : architecture is "true";
    ....
end rtl;
```





2）保留例化的模块

Verilog：

SomeModule u1(　　/* synthesis keep_hierarchy=true */

...

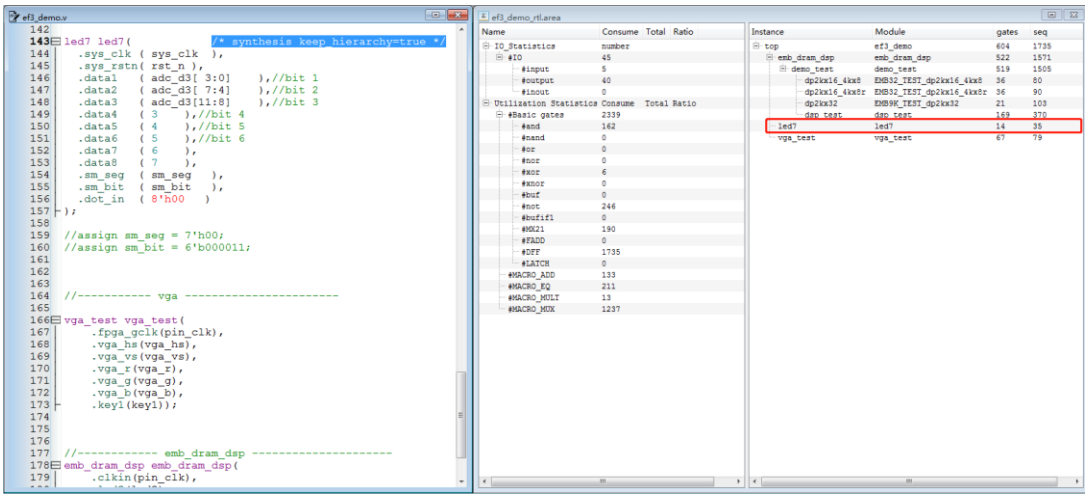
…

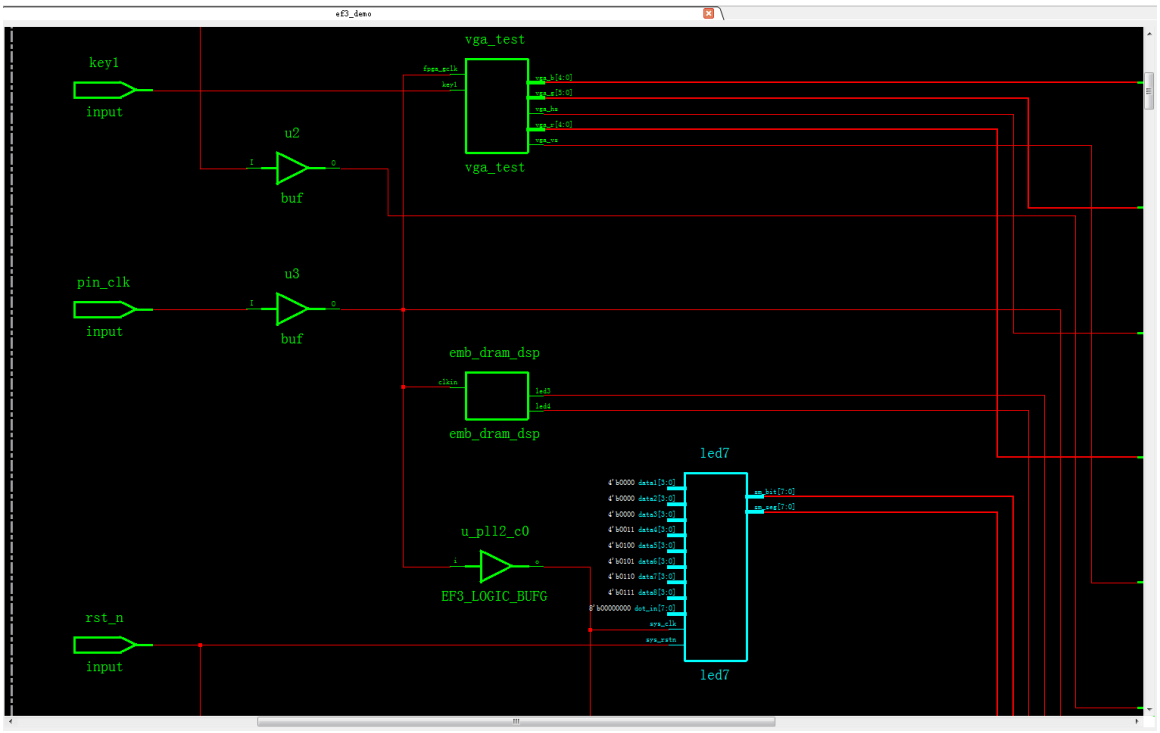
…

);

VHDL：

attribute keep_hierarchy : string;
attribute keep_hierarchy of u1 : label is "true";





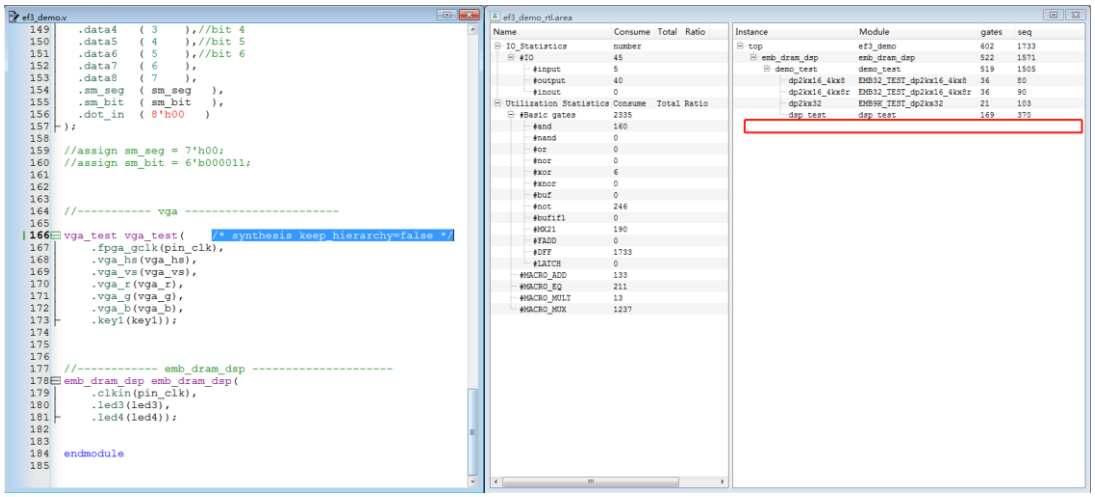
3）不保留例化的模块

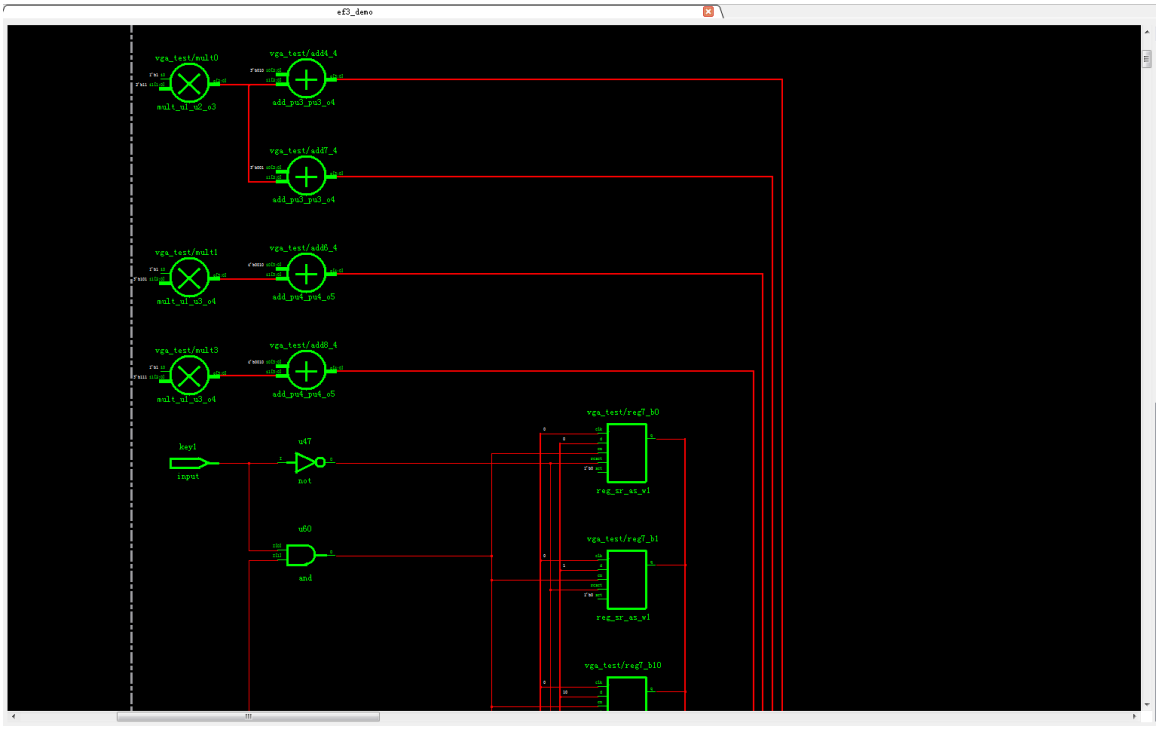
Verilog：

```
SomeModule u2(                    /* synthesis keep_hierarchy=false */
...
…
…
);
```

VHDL：

```
attribute keep_hierarchy : string;
attribute keep_hierarchy of u2 : label is "false";
```





2. 保存文件并编译；

3. 在 RTL/Gate Summary 可以看到被保留下来模块的资源使用情况，在 Schematic View 的 RTL Schematic/Gate Schematic 可以看到被保留下来模块的连线情况。

5.3 门级优化

门级优化包括普通逻辑的优化和映射、特殊逻辑的优化和映射等综合优化。门级优化将产生包含逻辑单元和专用功能单元的电路。

1. 在 FPGA Flow 面板中展开 **HDL2Bit**
2. 双击 **Optimize Gate**，或右键单击 **Optimize Gate**，选择 **Run**，此时将产生文件：gate.area，该文件的内容将与 keep_hierachy 参数的设定有关。
3. 参数配置

在菜单栏中，展开 **Process→Properties**，弹出 Properties Configuration 窗口，选择 **Optimize Gate** 进行设置。

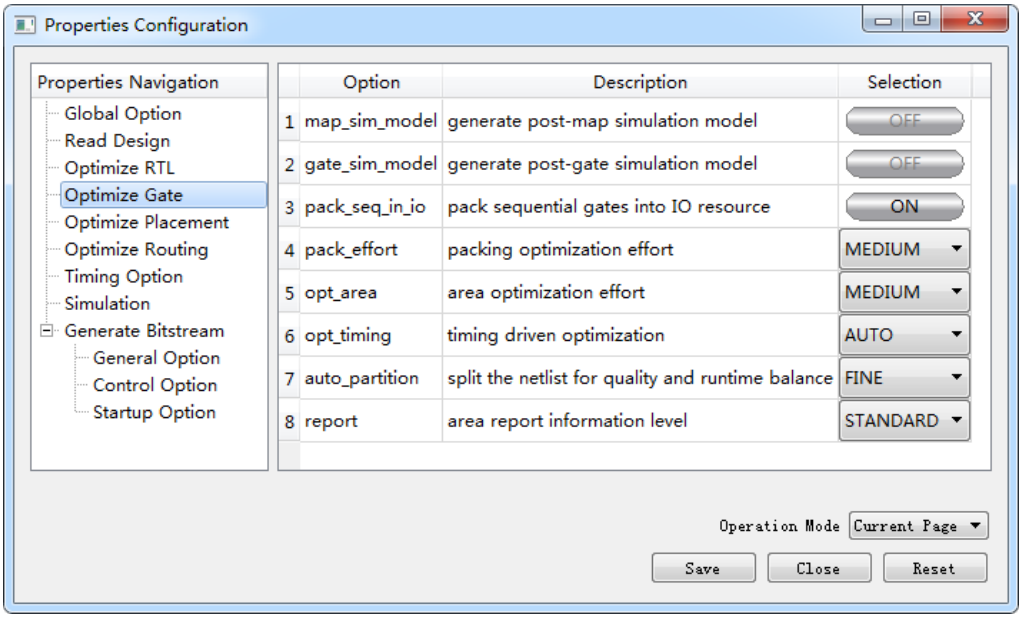


表 5-3 Optimize Gate Properties

Property	Comments	Default
map_sim_model	生成映射后电路仿真模型	OFF
gate_sim_model	生成门级电路仿真模型	OFF
pack_seq_in_io	吸收寄存器逻辑进 IO 模块	ON
pack_effort	逻辑包装优化级别	MEDIUM
opt_area	组合逻辑优化级别	MEDIUM
opt_timing	时序优化级别	AUTO
auto_partition	运行时可将网表进行分区	FINE
report	报告信息级别	STANDARD

5.4 布局优化

在得到正确的物理单元网表以后，需要对设计进行物理布局优化、IO 单元布局、物理单元布局和物理级逻辑优化等。布局优化将产生并处理只含有物理功能块(IOPAD、SLICE、RAM、DSP 等)的电路。

1. 在 FPGA Flow 面板中展开 **HDL2Bit**
2. 双击 **Optimize Placement**，或右键单击 **Optimize Placement**，选择 **Run**
3. 参数配置

在菜单栏中，展开 **Process→Properties**，弹出 Properties Configuration 窗口，选择 **Optimize Placement** 进行设置。

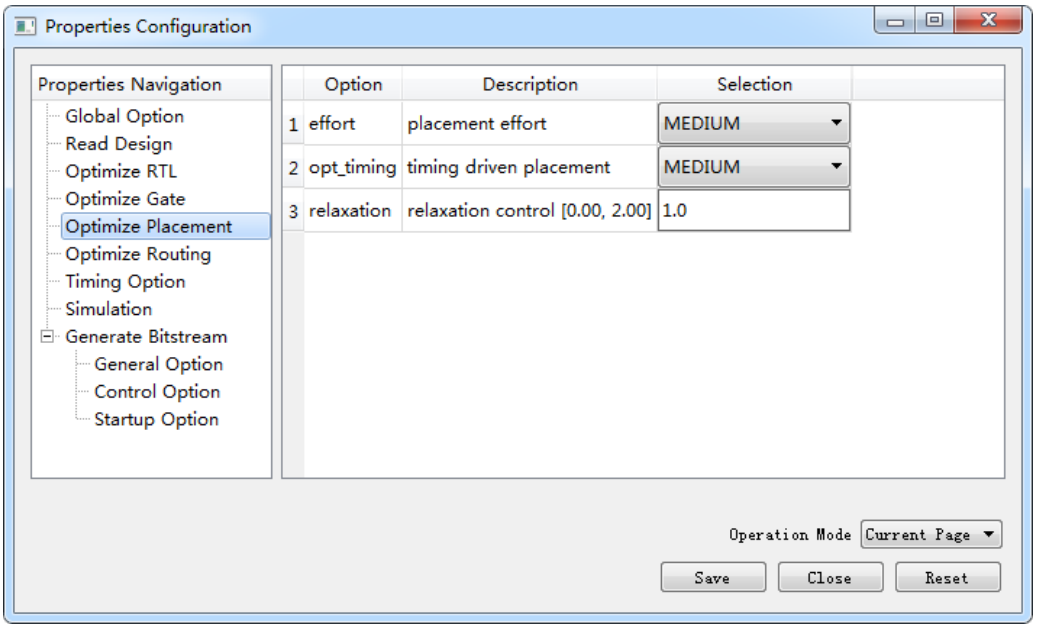


表 5-4 Optimize Placement Properties

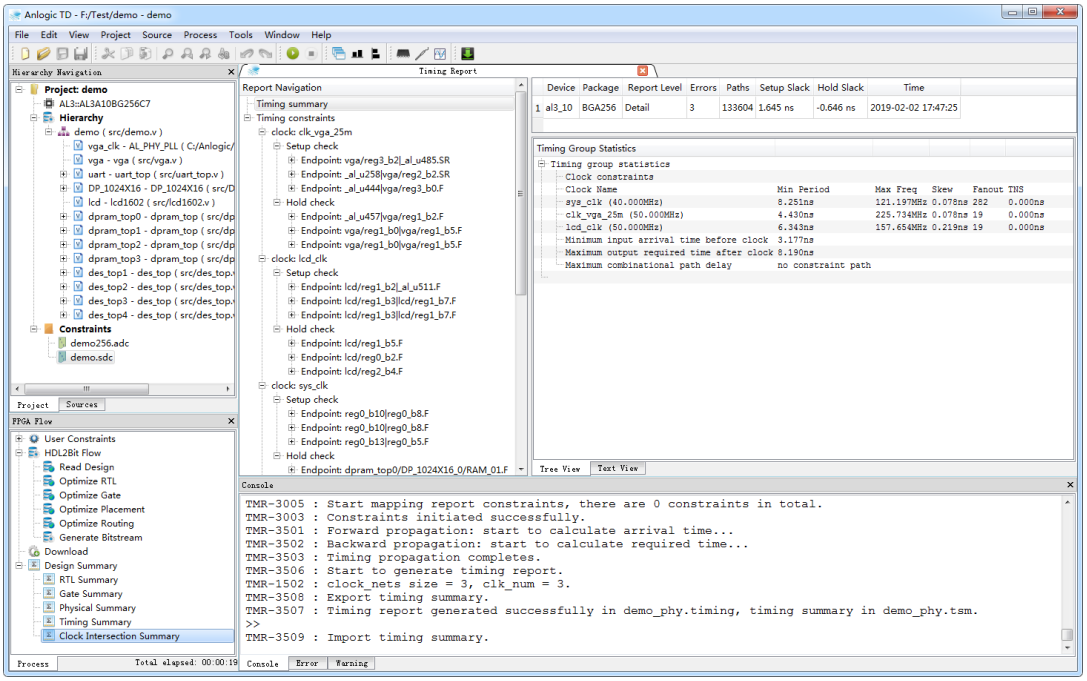
Property	Comments	Default
effort	布线优化级别	MEDIUM
timing	时延优化级别	MEDIUM
relaxation	放松程度控制范围 [0.00, 2.00]	1.0

5.5 布线优化

布局优化后，进行布线优化。布线优化将完成所有模块互联信号的物理连接。这一步也是用户设计实现的最后一步。这一步完成后，所有的物理信息都被确定。布线优化后可查看设计的详细信息，也可获得准确的电路时序信息。

1. 在 FPGA Flow 面板中展开 **HDL2Bit**

2. 双击 **Optimize Routing**，或右键单击 **Optimize Routing** 并选择 **Run**，此时将产生文件：phy.area。若用户已为工程添加 SDC 约束，在布线结束后，TD 会默认为用户生成时序分析报告。在 FPGA Flow 面板中展开 **Design Summary**，选择 **Timing Summary** 可查看最终时序报告。若用户没有为工程添加 SDC 文件，查看 **Timing Summary** 时，TD 会提示用户没有时序约束。



3. 参数配置

在菜单栏中，展开 **Process→Properties**，弹出 Properties Configuration 窗口，选择 **Optimize Routing** 进行设置。

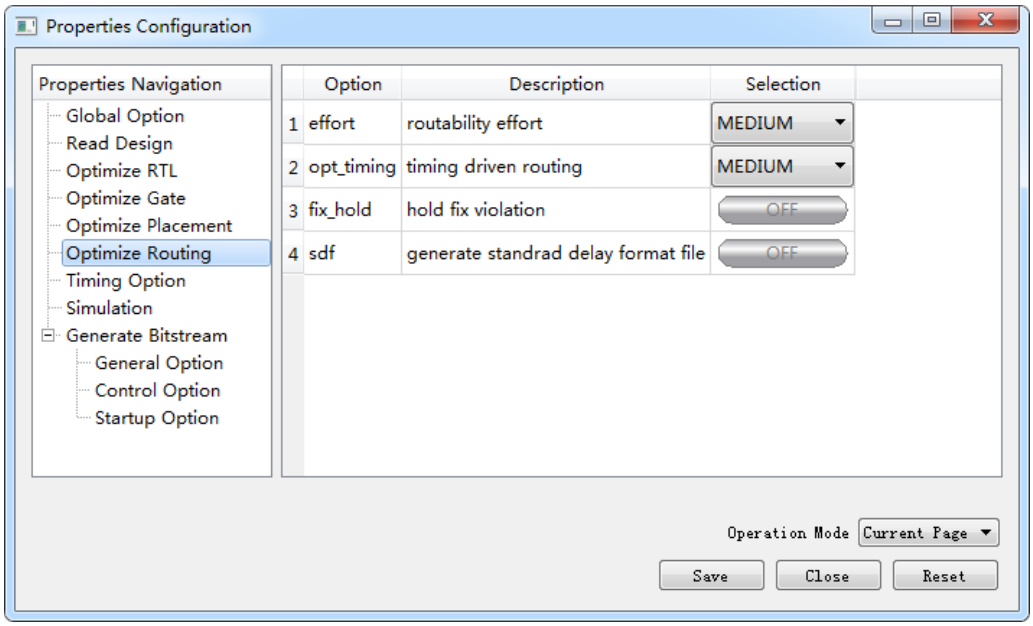


表 5-5 Optimize Routing Properties

Property	**Comments**	**Default**
effort	步通率优化级别	MEDIUM
opt_timing	时延优化级别	MEDIUM
fix_hold	修复 hold violation	OFF
sdf	生成标准时序反标文件	OFF

5.6 生成位流文件

Generate Bitstream 是将 FPGA 芯片中可编程开关的配置信息用二进制 0、1 的格式表示成位流(bitstream)数据供编程下载用。位流生成器为器件编程产生位流文件，下载工具将位流文件载入到外部的 SPI Flash 存储芯片或直接载入 FPGA 内部的配置存储器中。

1. 在 FPGA Flow 面板中展开 **HDL2Bit**
2. 双击 **Generate Bitstream**，或右键单击 **Generate Bitstream**，选择 **Run**，此时将产生文件：

   Your_Project_Name.bit，该文件为用二进制 0、1 的格式表示的可编程开关的配置信息

3. 参数配置

在菜单栏中，展开 **Process→Properties**，弹出 Properties Configuration 窗口，选择 **Generate Bitstream** 进行设置。

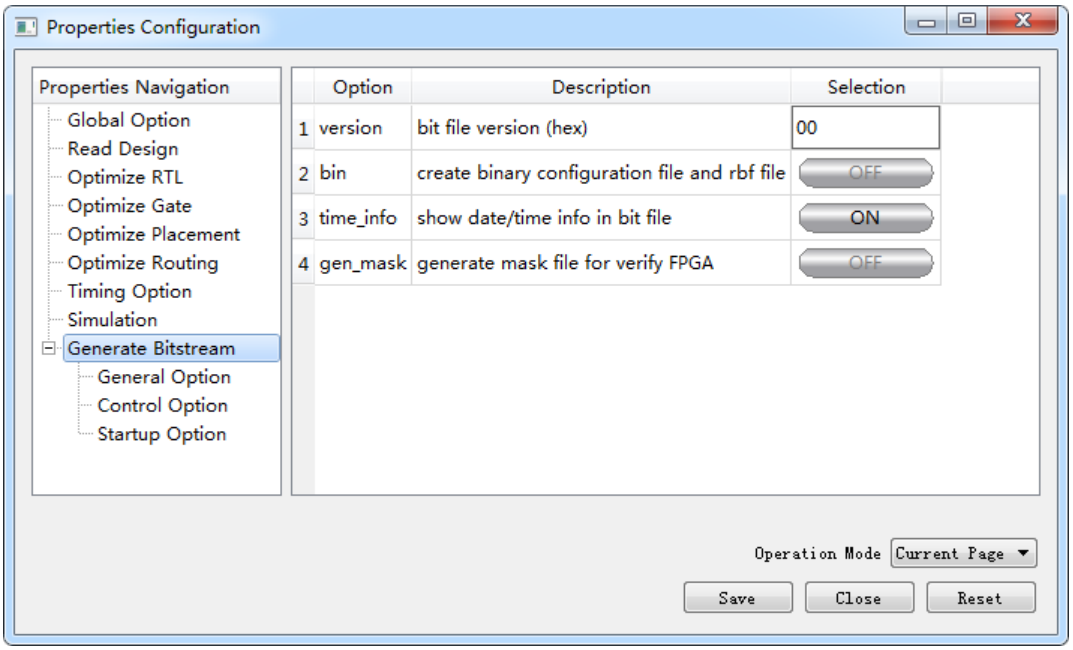


表 5-6　General Option

Property	Comments	Default
version	定义位流文件的编号	00
bin	生成纯二进制位流文件，即不包含常规 bit 文件的文件头	OFF
time_info	在位流文件中显示日期/时间信息	ON
gen_mask	生成一个用于 verify FPGA 的遮罩文件	OFF

Control Option 为一个 32 位的控制寄存器,分为若干个控制选项,改变各项的取值,可实现不同的控制效果。

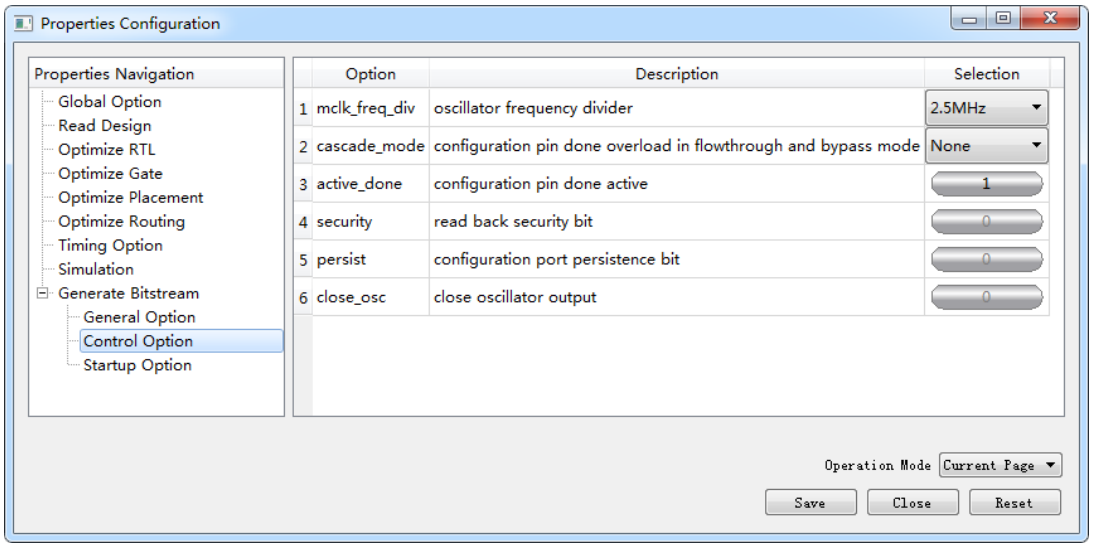


表 5-7　Control Option

Property	Comments	Default
mclk_freq_div	spi flash 的时钟分频系数	2.5MHz
cascade_mode	级联时使用，done pin 做为 input (2'b11: flowthrougth mode 2'b10: bypass mode)	None
active_done	done pin 处于 active 状态，由 cfg 内部决定	1
security	为 1 时，禁读 sram	0
persist	spi 的 pin 是否继续作为 cfg pin 在用户模式使用	0
close_osc	关闭振荡器	0

Startup Option 是跟启动项相关的 32 位控制寄存器，其控制原理同 Control Option.

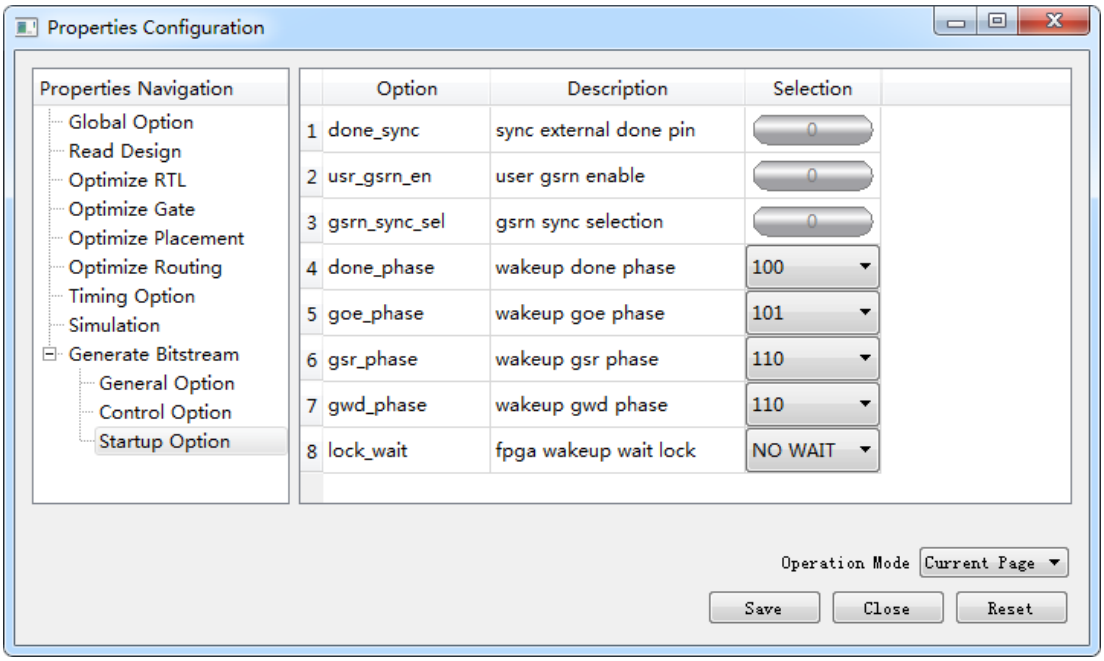
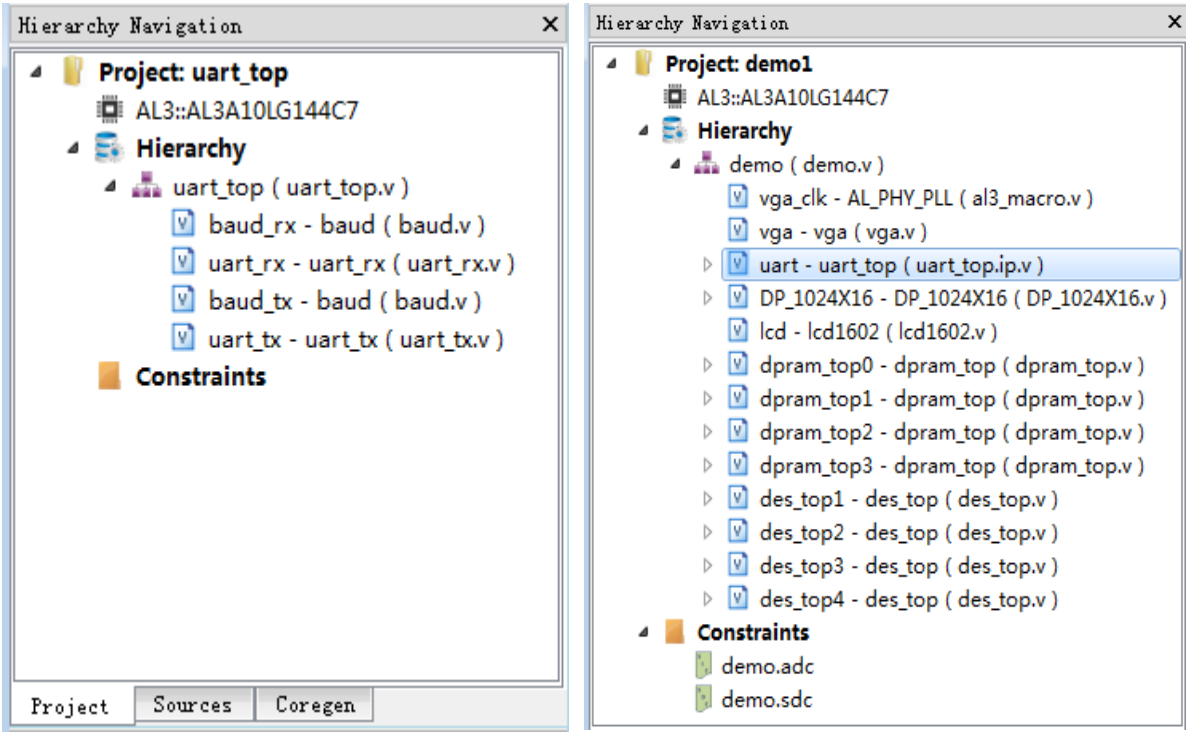


表 5-8 Startup Option

Property	**Comments**	**Default**
done_sync	是否同步 done pin 的值	0
usr_gsrn_en	是否使能用户的 gsrn 信号	0
gsrn_sync_sel	是否同步 gsrn	0
done_phase	决定在哪个 phase 放出 done	3'b100
goe_phase	决定在哪个 phase 放出 goe	3'b101
gsr_phase	决定在哪个 phase 放出 gsr	3'b110
gwd_phase	决定在哪个 phase 放出 gwd	3'b110
lock_wait	fpga 唤醒等待时钟 (NO WAIT: 0000 PLL0: 0001 PLL2: 0100 PLL0&PLL2: 0101)	NO WAIT

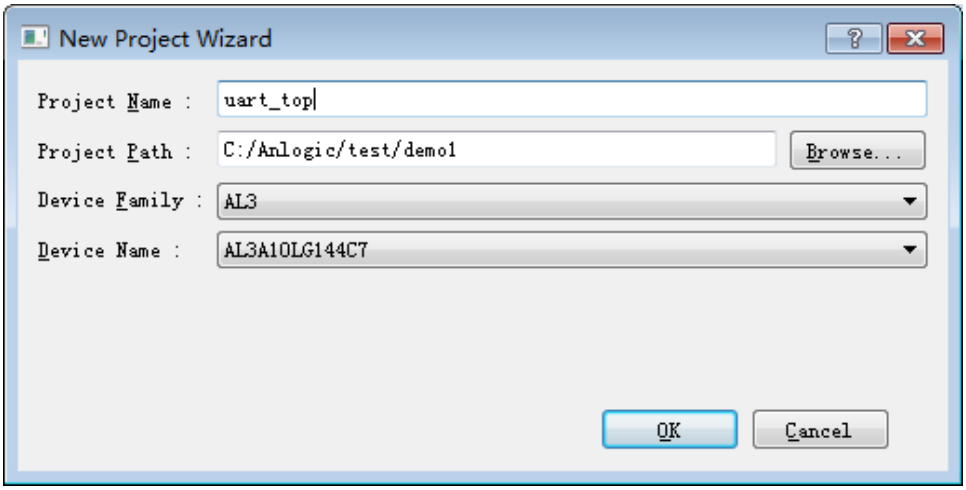
5.7 syn_ip_flow

在 FPGA 的设计过程中，用户若不愿让第三方看到自己的源代码，可以对设计的源代码实施保护措施，即将源代码单独做成一个 IP 模块供第三方调用。为此，TD 给用户提供了 syn_ip_flow 的功能，syn_ip_flow 将用户的高层电路描述综合成门级网表，可在一定程度上保护源代码。用户在使用该功能时，需建立两个工程，在第一个工程中添加需要保护的源代码并运行 syn_ip_flow，在第二个工程中添加外围电路并调用 syn_ip_flow 生成的网表文件。 如：该手册使用的 demo 模块中，用户想保护的子模块为 uart_top，则需建立如下两个工程：

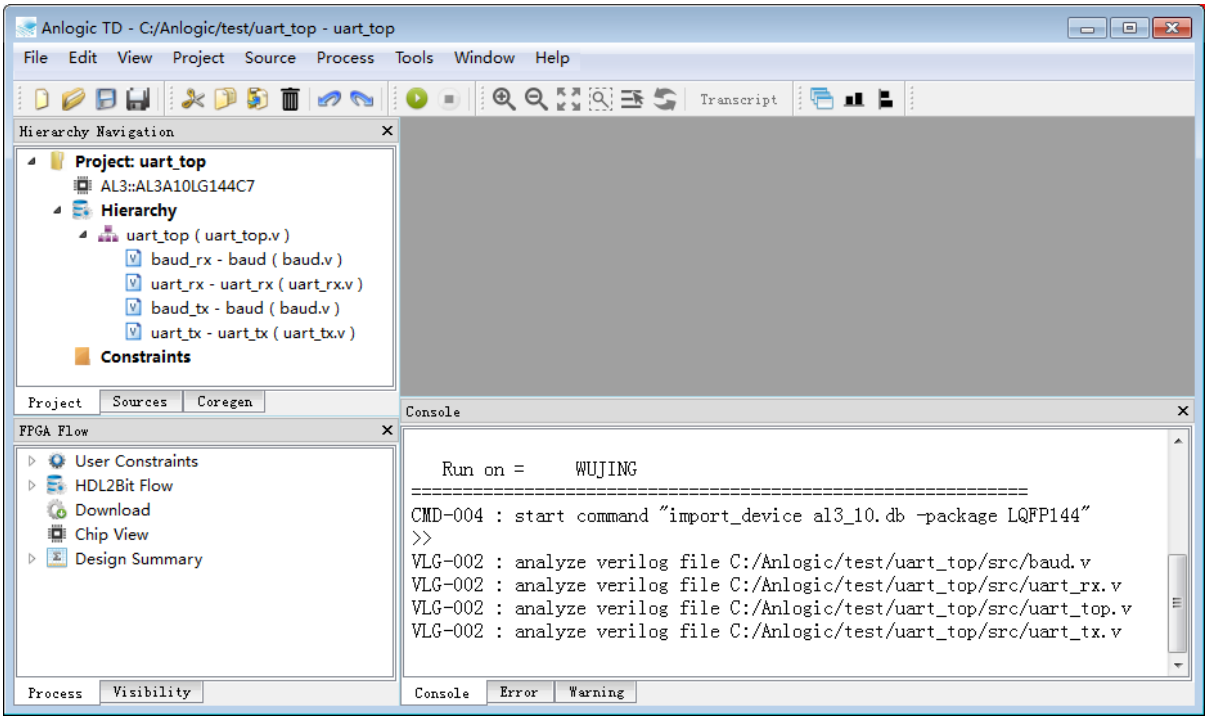


具体操作步骤如下：

1. 将需要保护的子模块单独建立工程。在本例中，以 uart 模块为例单独建立工程。

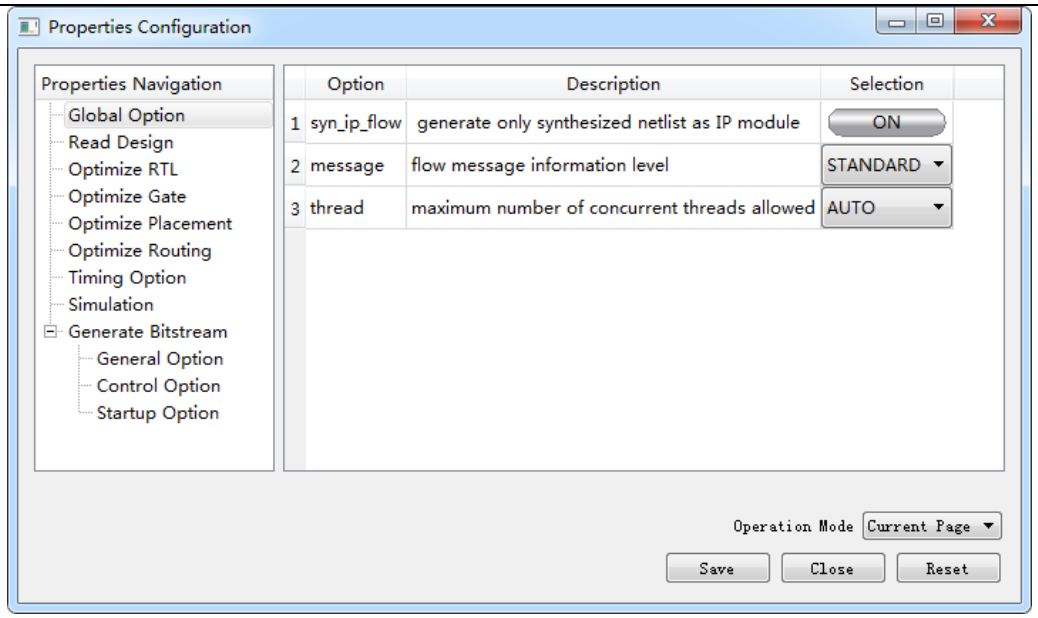


2. 为工程添加源文件



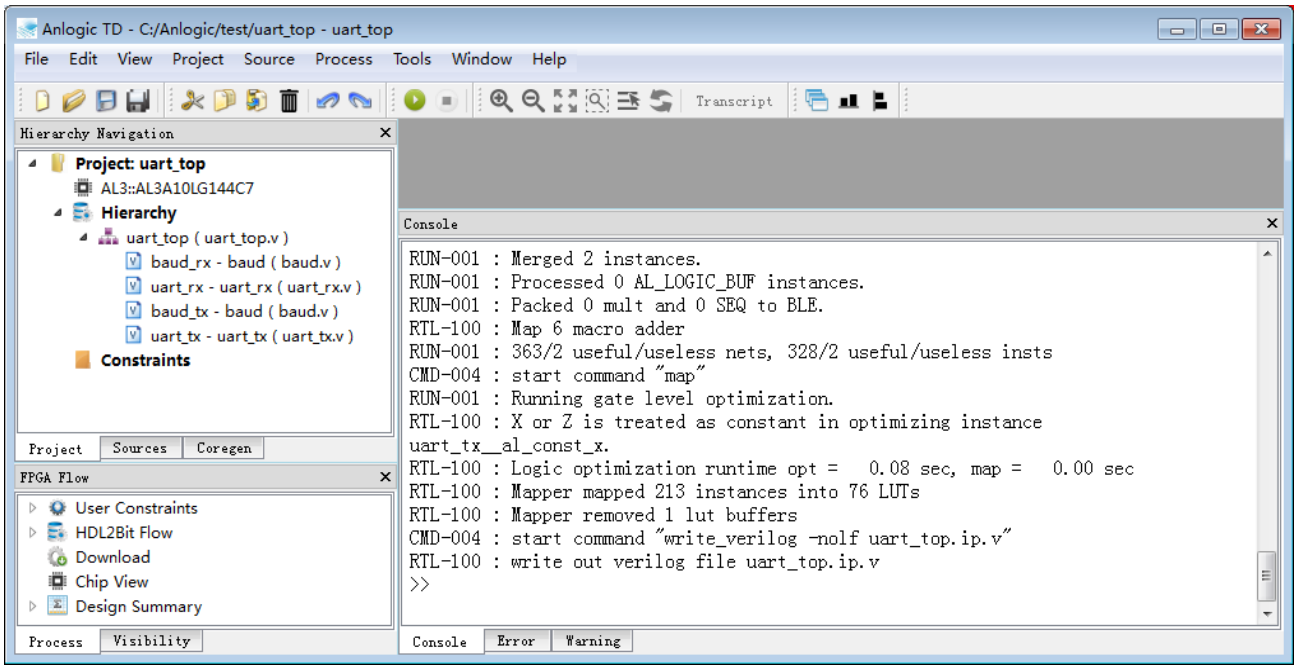
3. 运行 syn_IP_flow 时，需先设置参数：

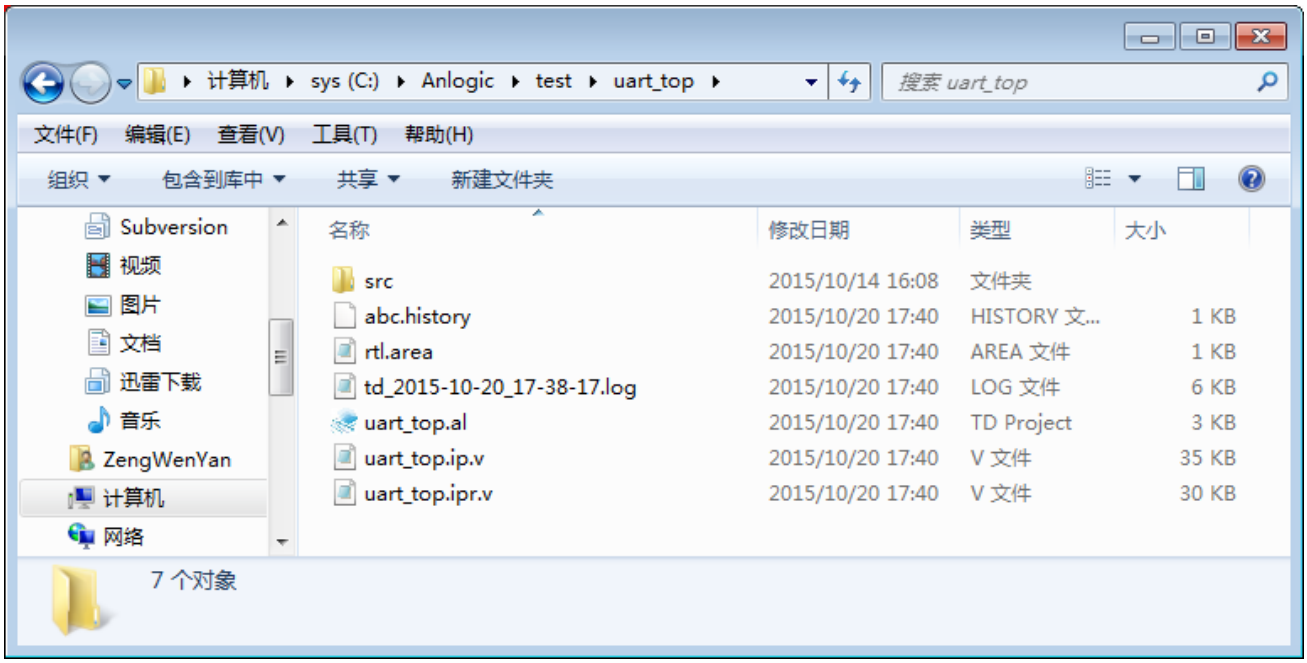
Process →Properties →Global Option →syn_ip_flow，将该参数设置为 ON，并点击“**Save**”进行保存。



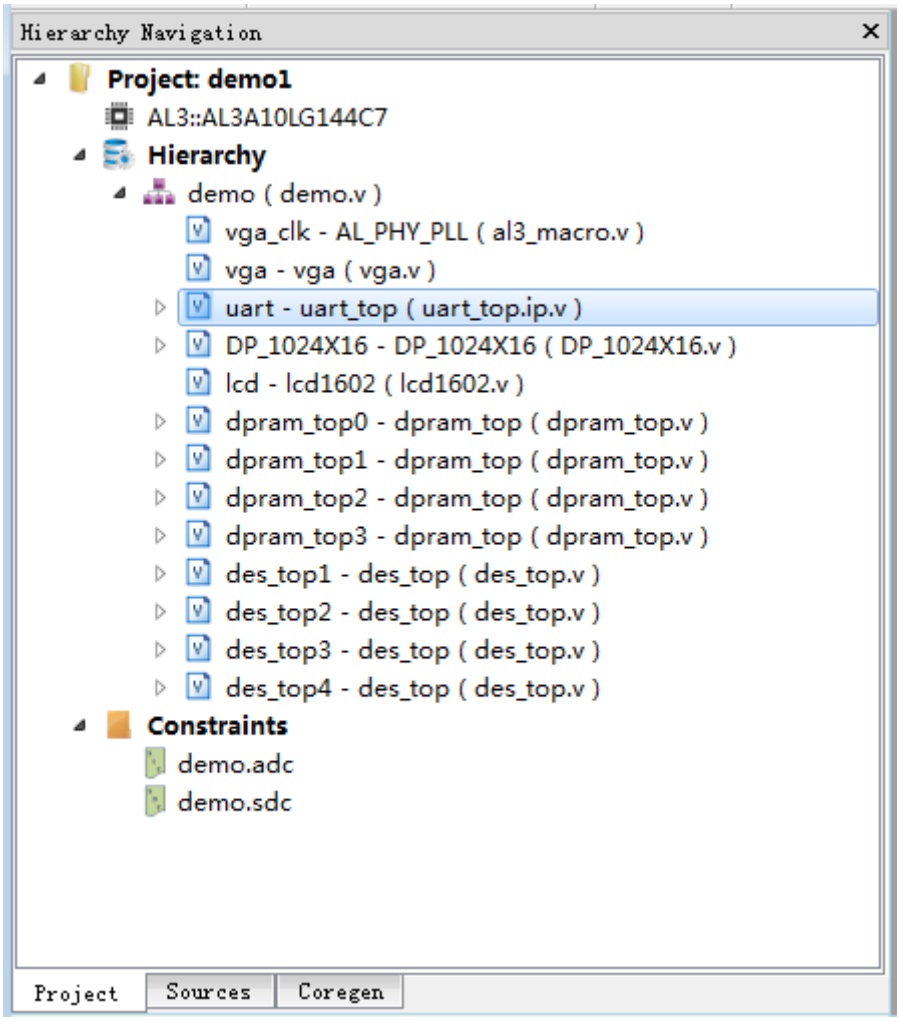
4. 参数设置完成后，点击图标运行工程，在工程运行完成后，TD 将生成两个文件： project_name.ip.v 和 project_name.ipr.v，并存放在工程目录下，这两个文件在功能上完全等价，并都可以用于仿真。

   但是，project_name.ip.v 中包含了源代码中的命名与声明等信息，便于 Debug。project_name.ipr.v 中隐藏了源代码中的命名与声明等信息,可更好的保护源代码。





5. 创建新工程，并调用上一步生成的 project_name.ip.v 或 project_name.ipr.v，并为工程添加相应的约束文件，运行整个工程，则完成了 syn_ip_flow 的整个流程。



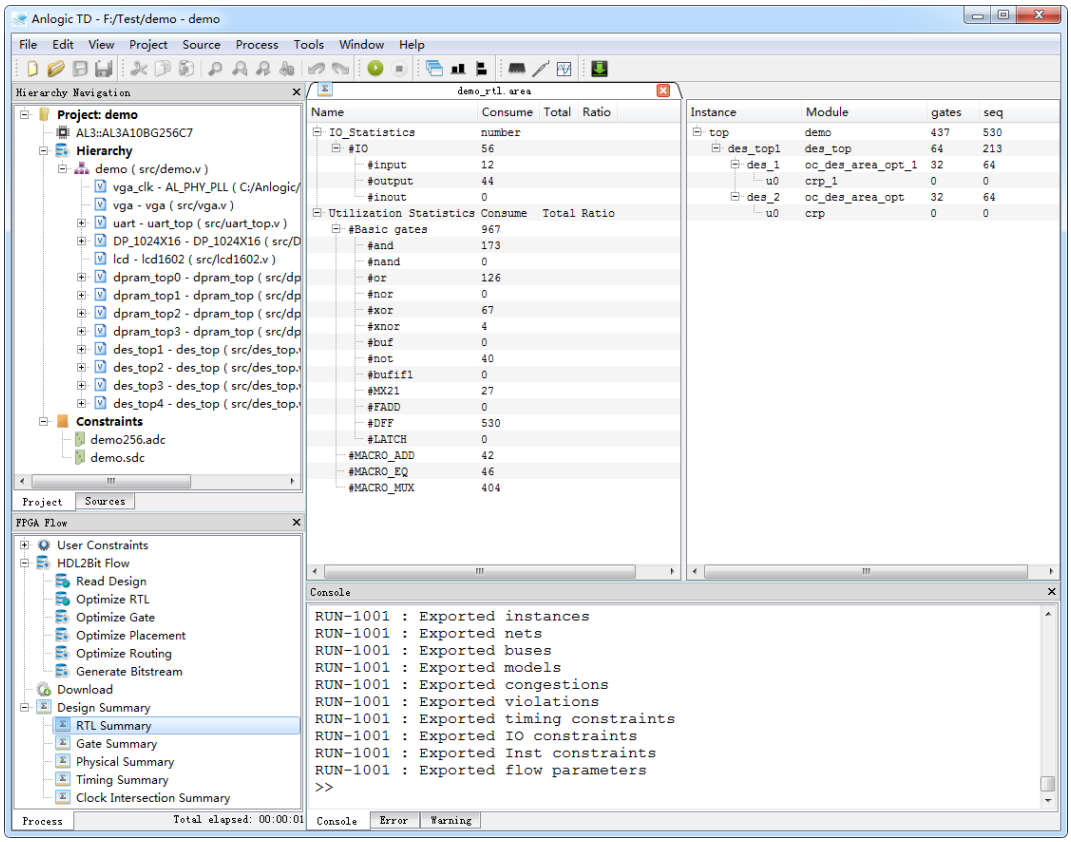
5.8 Design Summary

5.8.1 RTL Summary

当 HDL2Bit Flow 运行完 Optimize RTL，可展开 **Design Summary**，双击查看 **RTL Summary**。

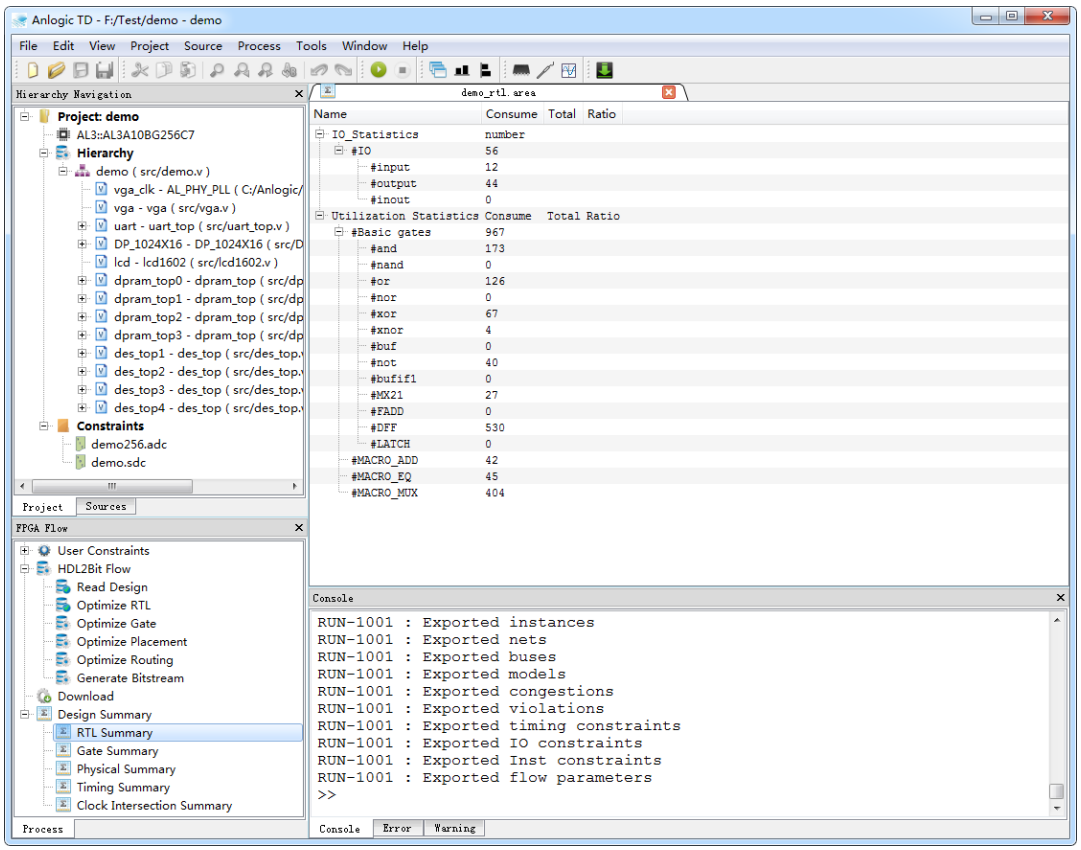
当 keep_hierarchy 选择为 manual 时，td 仅对用户通过 synthesis directive 选择的模块保留层次,该文件会层次化地给出分模块的资源使用;当 keep_hierarchy 为 auto 时，td 会自动选择较大的模块保留层次，该文件会层次化地给出分模块的资源使用。

Hierachy RTL Summary：



当 keep_hierarchy 为 flatten 时，该文件包含了源文件中的所有输入输出端口以及各逻辑门的使用情况。

Flatten RTL Summary：

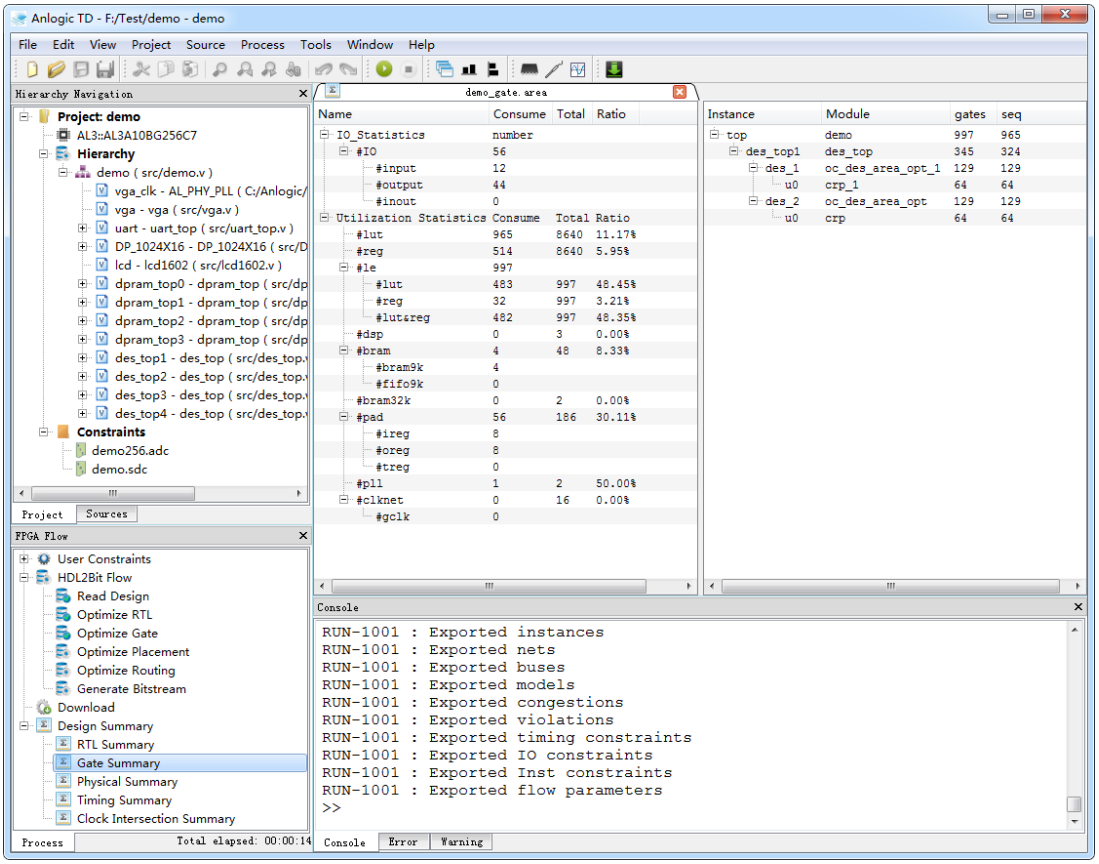


5.8.2 Gate Summary

当 HDL2Bit Flow 运行完 Optimize Gate，可展开 **Design Summary**，双击查看 **Gate Summary**。

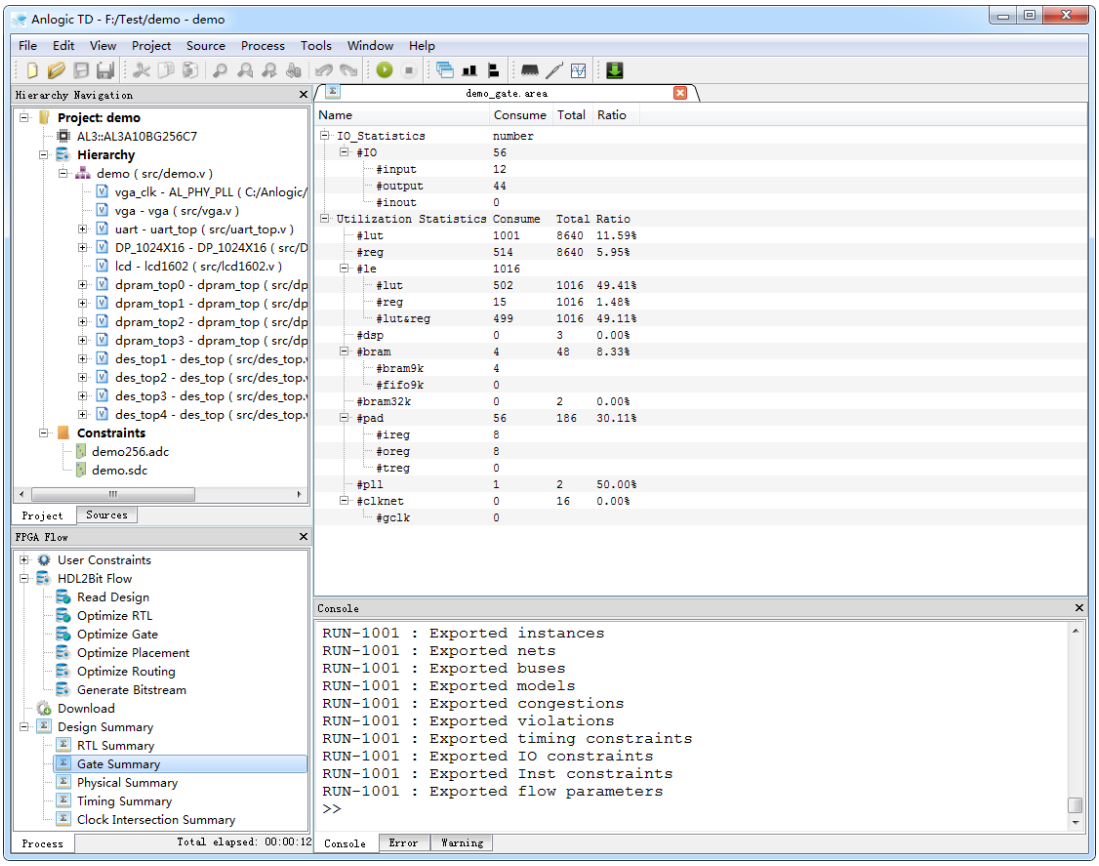
当 keep_hierarchy 选择为 manual 时，td 仅对用户通过 synthesis directive 选择的模块保留层次，该文件会层次化地给出分模块的资源使用；当 keep_hierarchy 为 auto 时，td 会自动选择较大的模块保留层次，该文件会层次化地给出分模块的资源使用。

Hierarchy Gate Summary：



当 keep_hierarchy 为 flatten 时，该文件包含了源文件中的所有输入输出端口以及各逻辑门的使用情况。

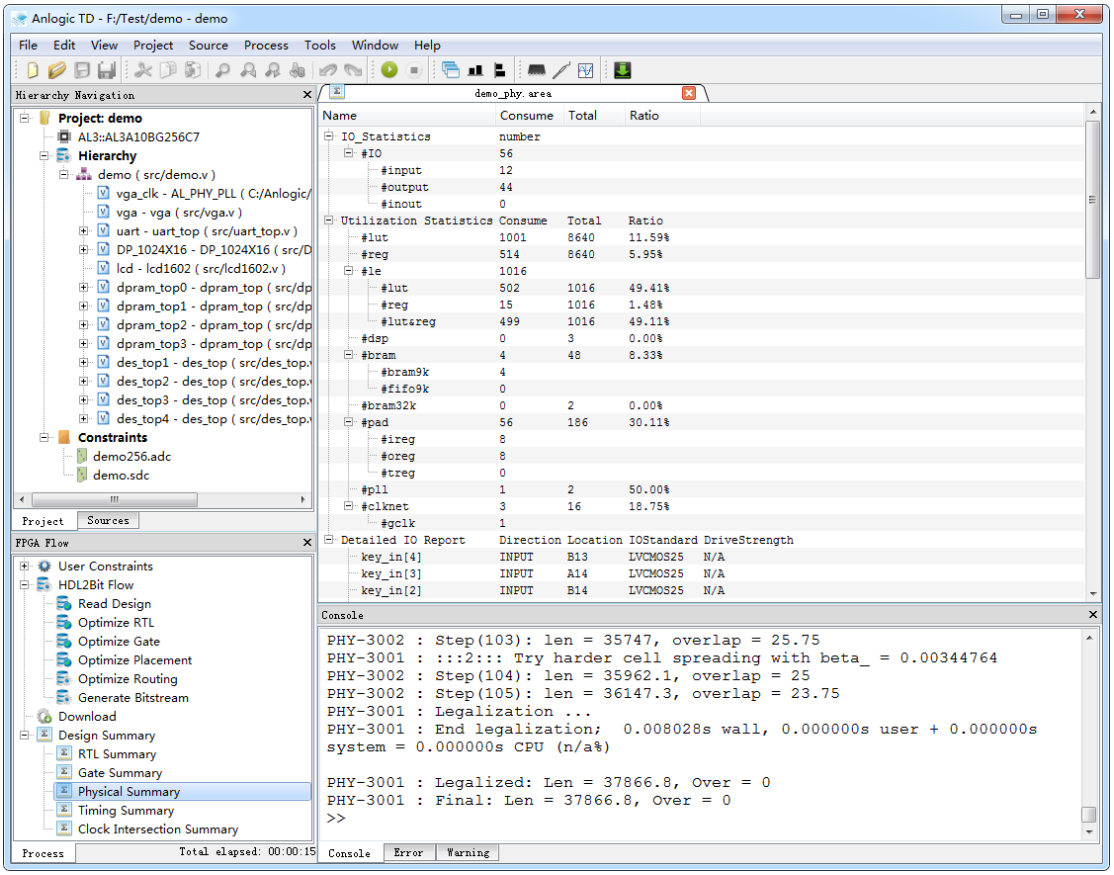
Flatten Gate Summary：



5.8.3 Physical Summary

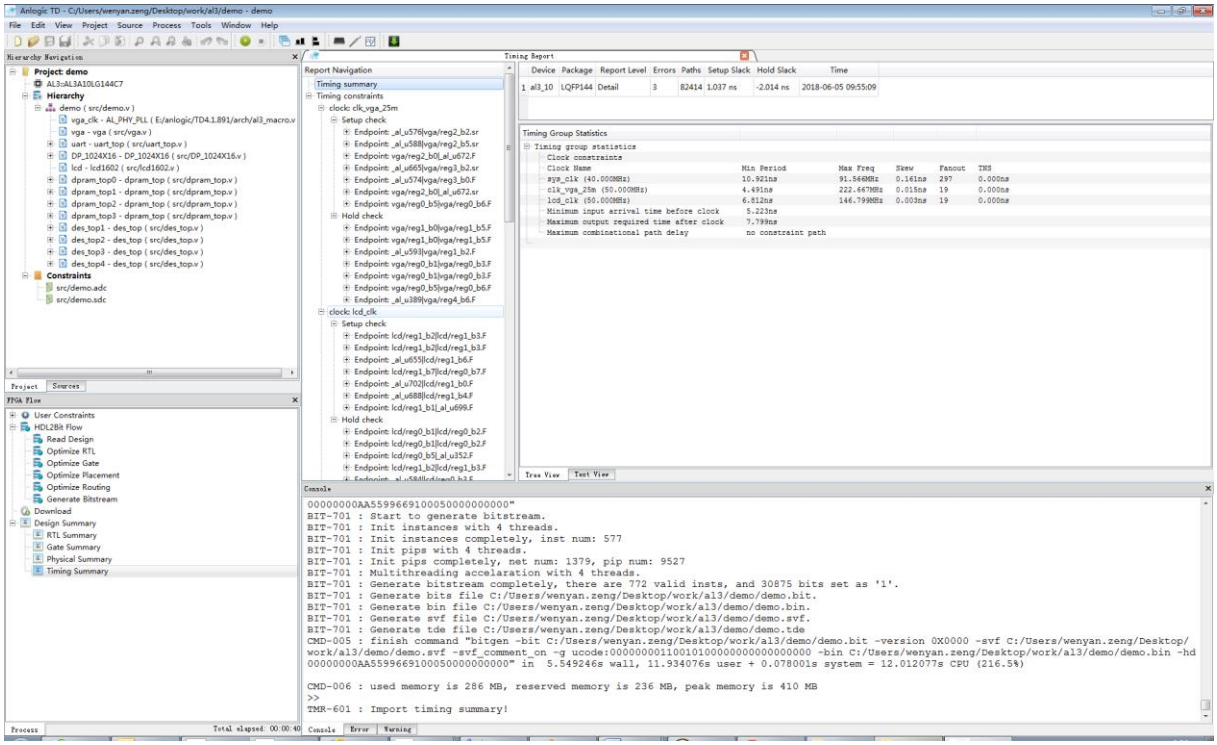
当 HDL2Bit Flow 运行完 Optimize Routing，可展开 **Design Summary**，双击查看 **Physical Summary**

该文件列出了 IO 端口和基本逻辑单元的使用情况和 IO 管脚分布情况。

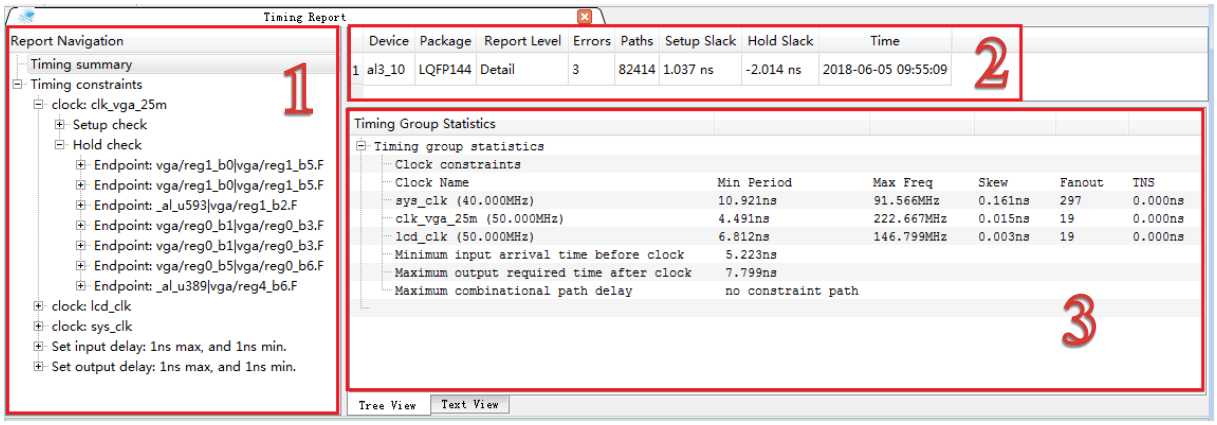


5.8.4 Timing Summary

当 HDL2Bit Flow 运行完成后，双击 Design Summary -> Timing Summary 即可打开时序分析报告（Timing Report），主界面如下：



Timing Report 的界面主要包括以下部分：



1. Report Navigation

此处为时序报告的目录树，将层次化的依次列出：时序约束类型，时序检查类型，EndPoint 名称，timing path 名称等信息。

2. Timing Summary

此处将列出时序报告对应的芯片，封装，以及简略的时序统计信息。

3. Timing Group Statistics

Clock constraints：列出了 SDC 中定义的各个 clock 能到达的最小周期，最大频率，时钟树的 skew，扇出数以及 TNS 数据；

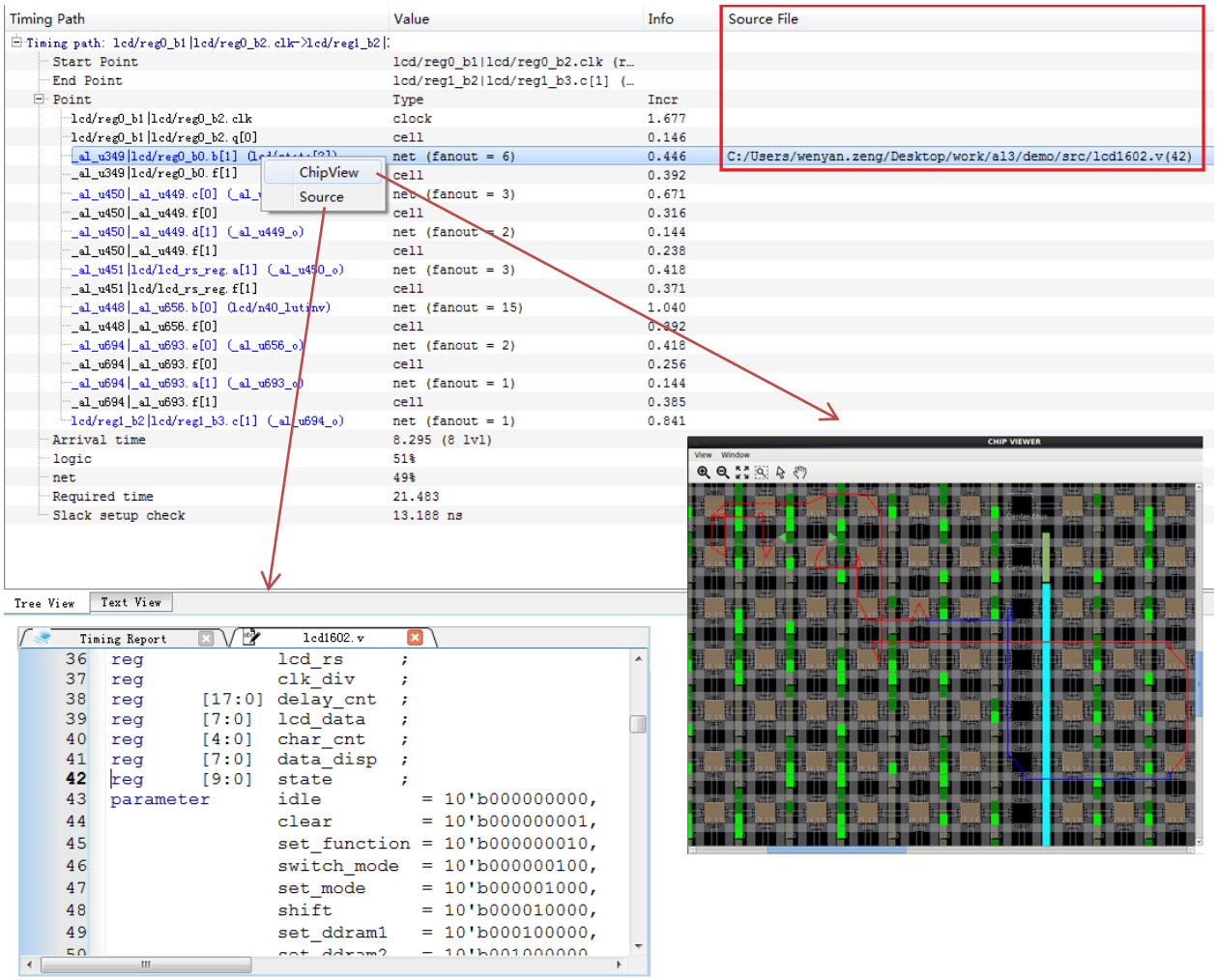
Minimum input arrival time before clock：FPGA 芯片内部 input->reg path 的最大延时。

Maximum output required time after clock：FPGA 芯片内部 reg->output path 的最大延时。

Maximum combinational path delay：input->output 直通的组合逻辑路径最大延时。

在 Timing constraints 中展开 clock 下的 Setup Check 或者 Hold Check，将根据 proerties 设置显示相应数目的 timing path，双击某一条 timing path，则会在 Timing Report 右侧栏显示该时序路径的详细信息，其中主要条目的含义如下：

Start Point	指时序路径的起点，一般为寄存器的 clock 端或原始输入端
End Point	指时序路径的终点，一般为时序单元的输入数据管脚或原始输出端
Point	依次列出了该时序路径经过的结点名
Type	指明该段时序路径经过的是单元(cell)还是线网(net)
Incr	该段时序路径的延时增量
Source File	指明该 net 在 Source File 中定义的地方(需打开 net_info 选项)
Arrival time	表示该时序路径的到达时间： Arrival time = $T_{launch\ edge}$ + $T_{launch\ clock\ delay}$ + path delay 在 setup check 的情况下为最大可能延时，而 hold check 的情况下则是最小可能延时，并且触发时钟的到达时间也已包含在内
Logic	表示逻辑资源在整条时序路径中所占用的延时比例
Net	表示互连资源在整条时序路径中所占用的延时比例
Required time	表示该项时序约束的要求时间： 最大路径约束（setup check）：Required time = $T_{capture\ edge}$ + $T_{capture\ clock\ delay}$ – T_{setup} – $T_{clock\ uncertainty}$ 最小路径约束（hold check）：Required time = $T_{capture\ edge}$ + T_{hold} + $T_{clock\ uncertainty}$
Slack	表示时序路径的松弛余量，小于 0 则代表有时序风险： 最大路径约束（setup check）：Slack = Required time – Arrival time 最小路径约束（hold check）：Slack = Arrival time - Required time



Timing path 的命名规则为：

“/”表示层级关系；

“|”表示并联关系；

“.”表示从属关系。

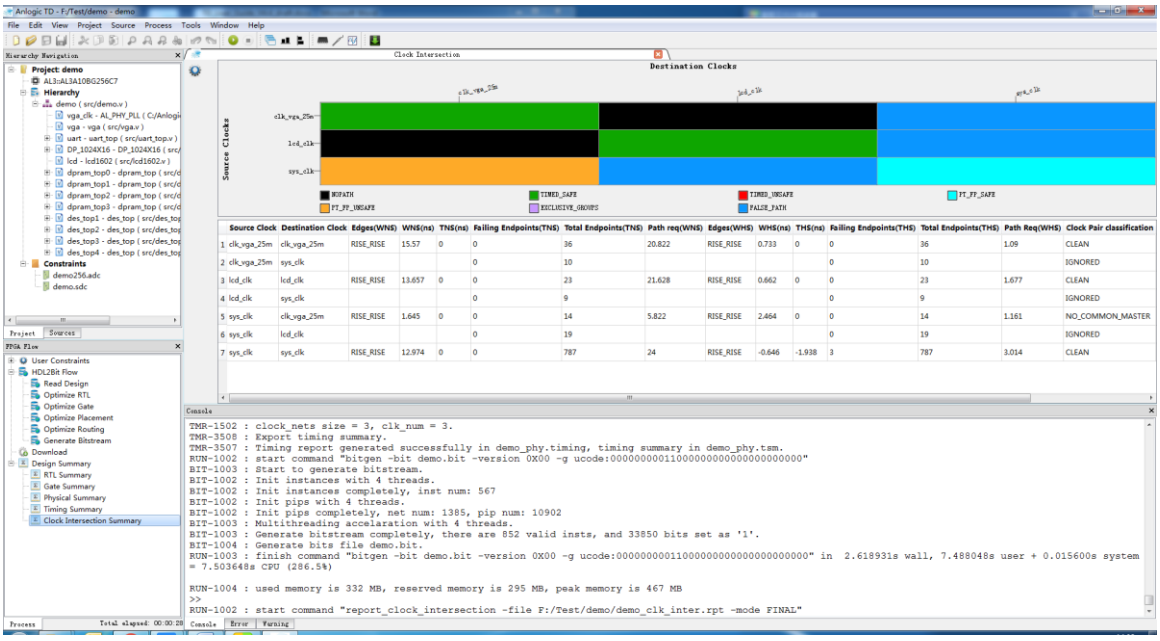
下表介绍了时序路径经过的各节点含义：

Point	Type	Incr	Comment
lcd/reg0_b1|lcd/reg0_b2.clk	clock	1.677	表示 lcd 模块内的 reg0_b1 和 reg0_b2 被合并到了同一个 slice，该路径起点为 slice 的 clk 端。 clock 代表时钟树的延时为 1.677ns
lcd/reg0_b1|lcd/reg0_b2.q[0]	cell	0.146	为 cell 内部的延时，即该 slice 的 clk 端到输出 q[0]的延时为 0.146ns。
_al_u349|lcd/reg0_b0.b[1] (lcd/state[2])	Net fanout=6	0.446	为 net 延时，即前一个 slice 的 q 端到下一个 slice 的 b 端之间的线网延时为 0.446ns； 括号里为 net 名称：net lcd/state[2]。 fanout=6，表示该信号一共驱动了 6 个管脚。
…	…	…	

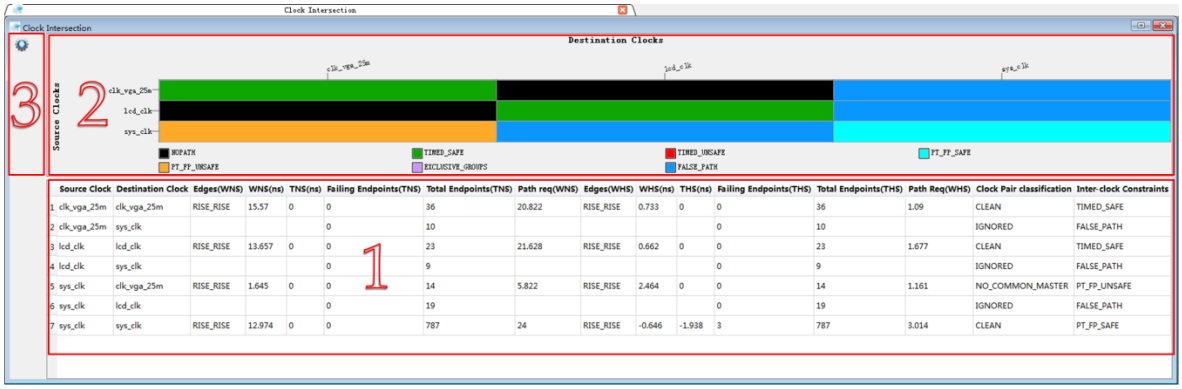
对于给出了文件路径的 net 可以右键该 net，点击 Source 则可跳转到源文件相对应的地方，点击 ChipView 则可在 ChipView 中显示该路径的具体走线，蓝色线段则为该 net 在整个 path 中的部分。

5.8.5 Clock Intersection Summary

当 HDL2Bit Flow 运行完成后，双击 Design Summary -> Clock Intersection Summary 即可打开查看 Clock Intersection，主界面如下：



Clock Intersection 的界面主要包括以下部分：



1. Clock Intersection 数据表

此处为时钟交互的数据表格，表格每一行的数据均是时钟“单向”的，即从一个时钟到另一个时钟。如果两个不同时钟（如 clk_vga_25m 和 sys_clk）之间互有信号发送和捕获，在此表格中需要分两行描述，分别为 clk_vga_25m → sys_clk 和 sys_clk → clk_vga_25m。

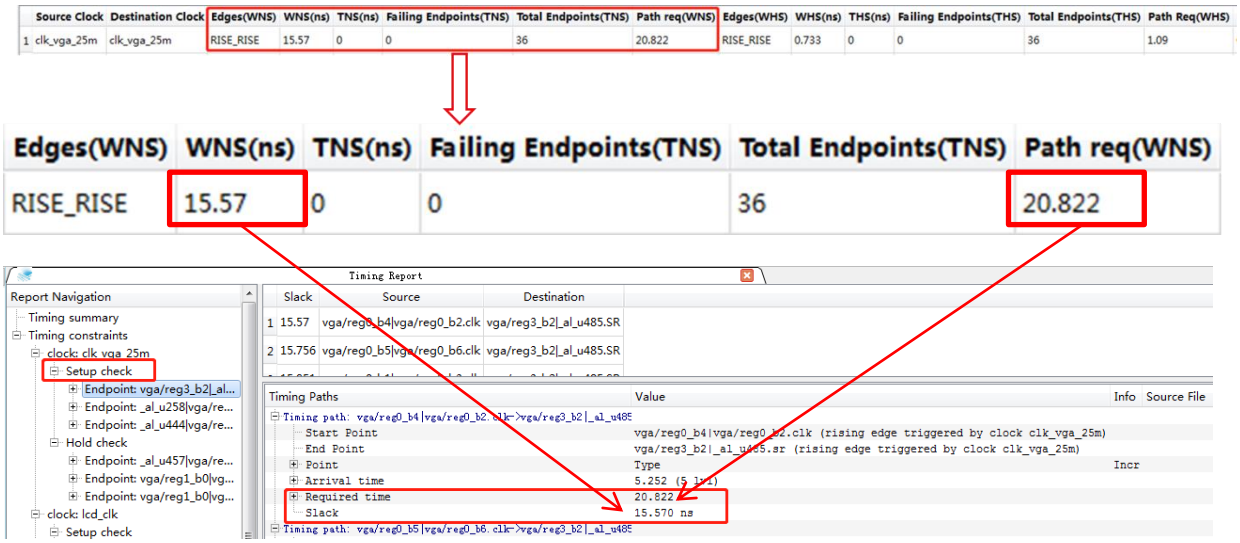
	Source Clock	Destination Clock
1	clk_vga_25m	clk_vga_25m
2	clk_vga_25m	sys_clk
3	lcd_clk	lcd_clk
4	lcd_clk	sys_clk
5	sys_clk	clk_vga_25m
6	sys_clk	lcd_clk
7	sys_clk	sys_clk

其中，source clock 和 destination clock 定义如下：

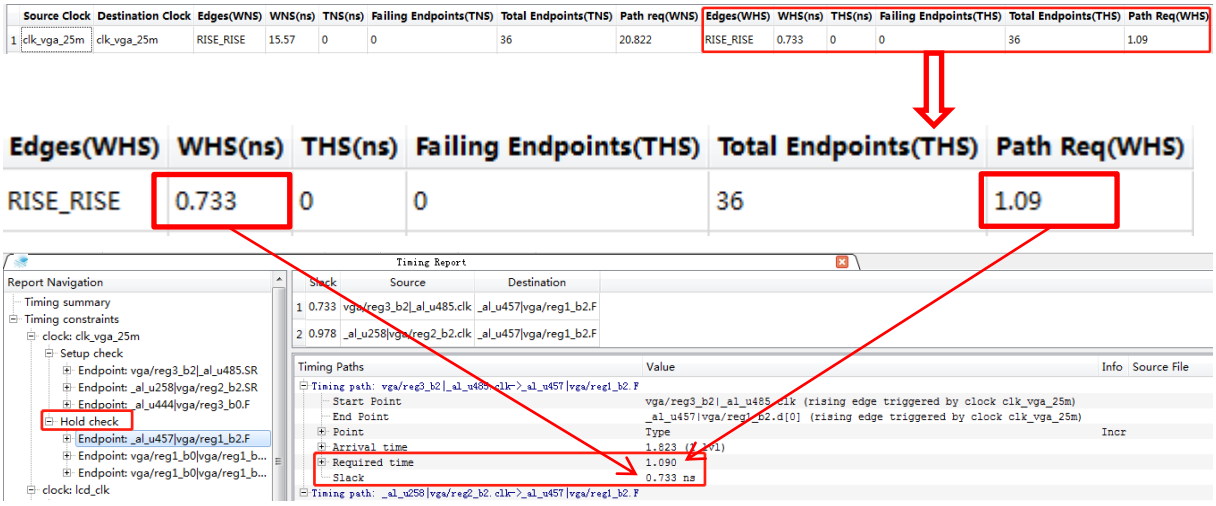
Source Clock　　data path 起点 DFF 的控制时钟
Destination Clock　　data path 终点 DFF 的控制时钟

表中显示了时序路径的详细信息，与 Timing Report 中的信息相对应，其中主要条目的含义如下：

Edges(WNS/WHS)	此单向时钟关系中最 critical 路径的 launch clock、capture clock 的触发沿。RISE_RISE 表示 launch/capture clock 均是上升沿触发，类似的还有 RISE_FALL、FALL_RISE、FALL_FALL
WNS/WHS(ns)	Worst Negative Slack，在此单向时钟关系的所有时序路径中，最 critical 的那条路径的 slack，以 ns 为单位，不一定为负
TNS/THS(ns)	Total Negative Slack，此单向时钟关系的所有时序路径中 slack 为负的总和，以 ns 为单位，肯定不为正。如果不存在时序违规路径，此值为 0
Failing Endpoints(TNS/THS)	此单向时钟关系的所有时序路径中出现 slack 为负的 endpoint 的总数
Total Endpoints(TNS/THS)	design 中存在的此单向时钟关系的所有 endpoint 总数
Path req(WNS/WHS)	在此单向时钟关系的所有时序路径中，最 critical 的那条路径的 require time，以 ns 为单位



此处描述的分析类型为 setup 或者 recovery，以及 timing report 中的对应信息。



此处描述的分析类型为 hold 或者 removal，以及 timing report 中的对应信息。

Clock Pair classification	Inter-clock Constraints
CLEAN	TIMED_SAFE

Clock Pair classification：静态综合时序约束的信息对两个时钟进行分类。可能出现的标签有：

IGNORED	使用 set_clock_groups 或者 set_false_path 指定此单向时钟关系中的所有路径不考虑
VIRTURAL	表示 source 或者 destination clock 中至少有一个是 virtual clock，多体现于 input/output path 的时序路径中
DERIVED	表示 source 或者 destination clock 中至少有一个是 derived clock，derived_clocks 生成的时钟
NO_COMMON_MASTER	表示两个时钟来自于两个 primary clock，包括不同 primary clock 之间的比较
NO_EXPAND	表示在 1000ns 范围内两个时钟周期找不到时钟沿重合的点，完全是异步时钟
CLEAN	表示两个时钟同属一个 primary clock 定义的时钟域且有限时间跨度内同步

静态分析按上述顺序依次检查定性，当满足一个定义时检查终止。

Inter-clock Constraints：在时序分析过程中动态地给单向时钟关系记录并在分析结束后完成总结定性。可能出现的标签有：

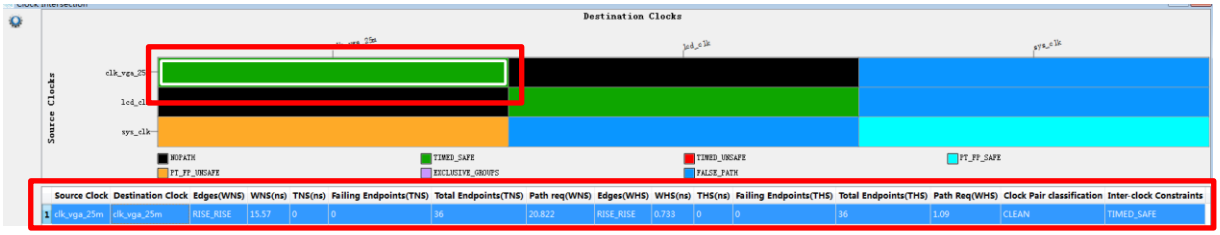
NOPATH	不存在此单向时钟关系的时序路径
FALSE_PATH	此单向时钟关系中的所有时序路径均为 false path
EXCLUSIVE_GROUPS	此单向时钟关系中的所有时序路径均被 set_clock_groups 约束要求忽略
TIMED_SAFE	source 和 destination 时钟同属一个时钟域且周期简单同步
TIMED_UNSAFE	source 和 destination 时钟属不同时钟域或者同属一个时钟域但周期不同步
PT_FP_SAFE	source 和 destination 时钟同属一个时钟域且周期简单同步，有一部分时序路径为 false path
PT_FP_UNSAFE	source 和 destination 时钟属不同时钟域或者同属一个时钟域但周期不同步，有一部分时序路径为 false path

2. N*N 方格图

此处是根据 Clock Intersection 数据表格中的内容画出的一个 N*N 方格图：图中纵向为 Source clocks，横向为 Destination Clocks，不同的 Inter-clock Constraints 对应不同的颜色，默认 default 颜色如下：

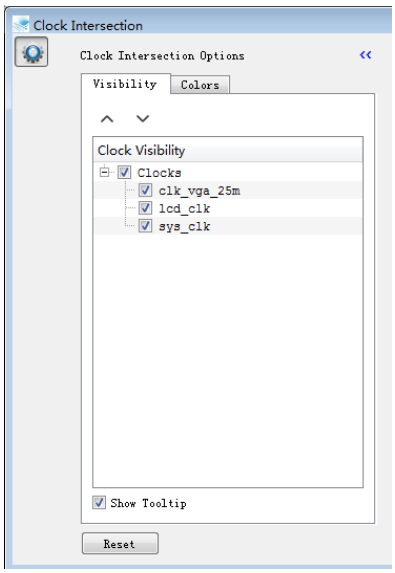
NOPATH
TIMED_SAFE
TIMED_UNSAFE
PT_FP_SAFE
PT_FP_UNSAFE
EXCLUSIVE_GROUPS
FALSE_PATH

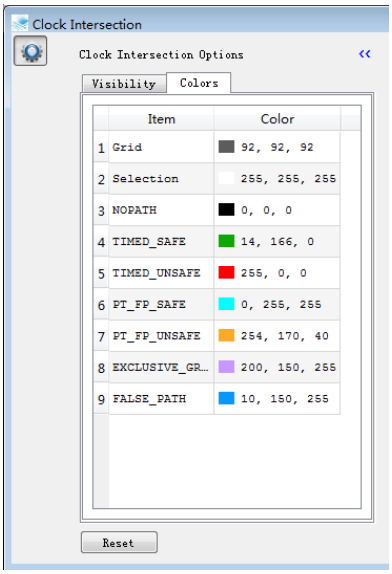
选择方格中的某一格会在数据表中显示对应的一行；选择数据表中的某一行会在对应的方格中用白色边框标示。



3. Clock Intersection Options

点击图中按钮处即可打开 Clock Intersecton Options：Visibility 一栏可以通过对任一 clk 前的方框进行勾选或取消勾选来决定显示或隐藏该 clk；Colors 一栏可以对任一 Inter-clock Constraints 的颜色进行修改。





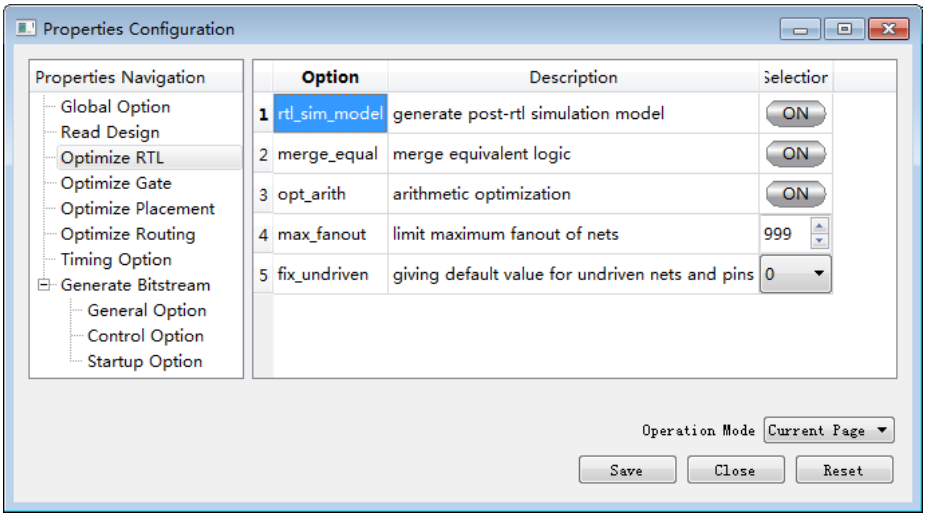
6 功能仿真

TD 支持用户使用第三方工具（如 Synopsys VCS、 Mentor Graphics Modelsim 等）来进行功能验证和时序验证。TD 提供仿真所需的功能和时序模型。

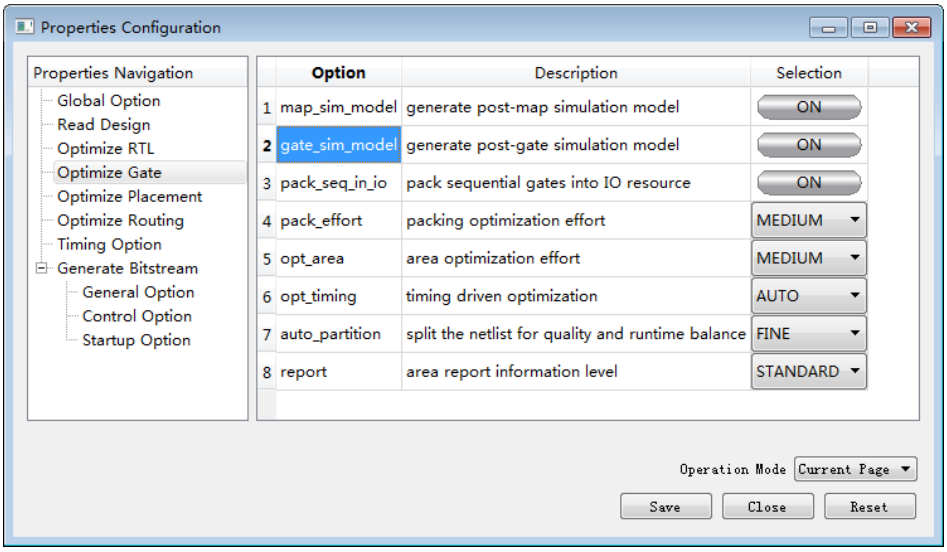
该章节主要介绍在 TD 软件中生成供 Modelsim 仿真所需文件的流程。

1. 在运行 HDL2Bit Flow 前，先设置相关参数。

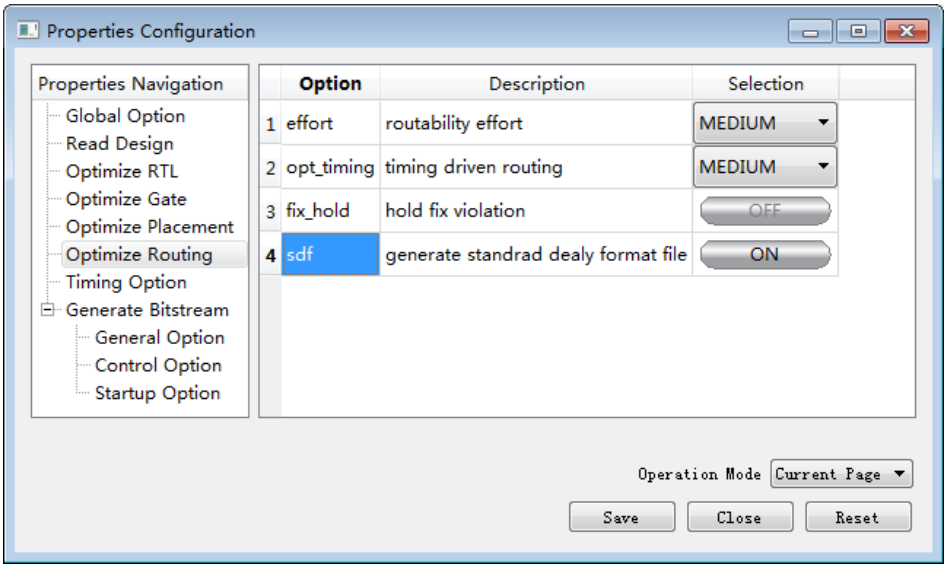
Process → Properties → Optimize RTL：set rtl_sim_model ON。



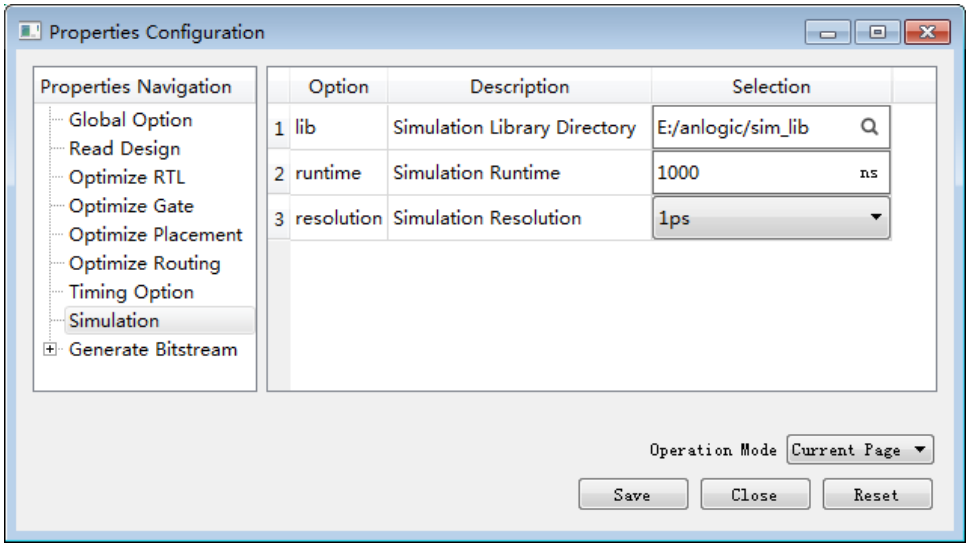
Process → Properties → Optimize Gate： set gate_sim_model ON。



Process → Properties → Optimize Routing：set sdf ON。



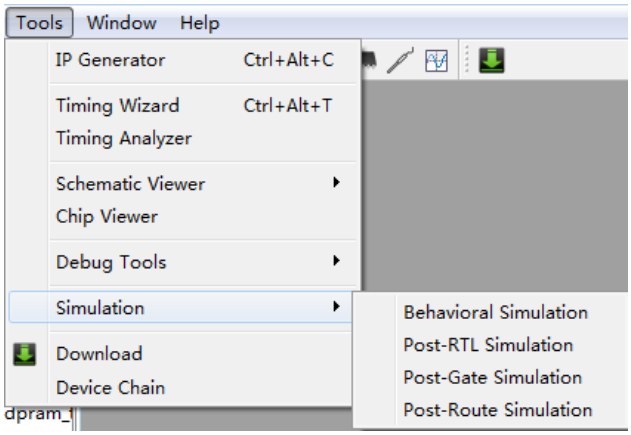
2. 设置 Modelsim 仿真相关参数



Property	Comments	Default
lib	指定仿真的库文件	没有默认值，需手动指定
runtime	指定仿真运行的时间	1000 ns
resolution	指定仿真的时间精度	1　ps

3. 运行 HDL2Bit Flow

4. 运行 **Tools → Simulation**



当 HDL2Bit Flow 运行至 Read Design 这一步时，可执行 Behavioral Simulation；

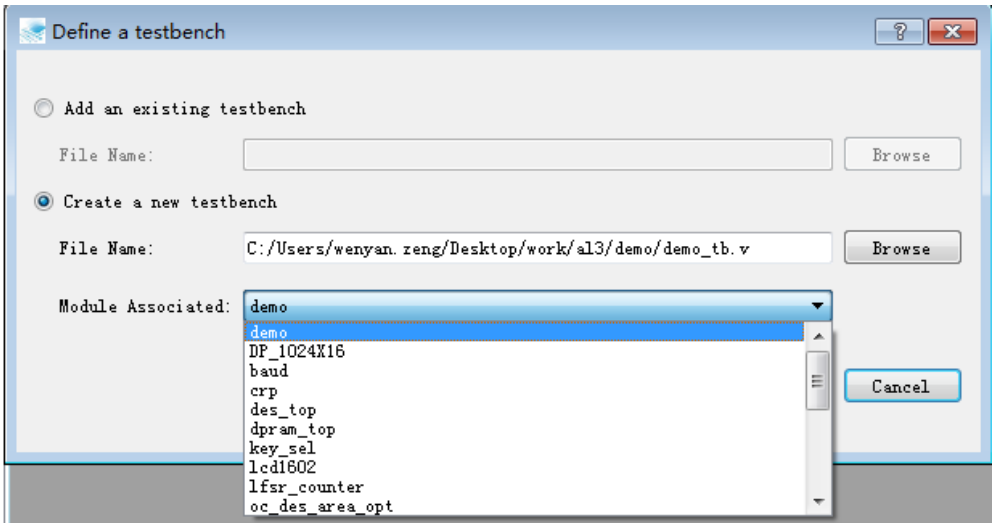
当 HDL2Bit Flow 运行至 Optimize RTL 这一步时，可执行 Post-RTL Simulation；

当 HDL2Bit Flow 运行至 Optimize Gate 这一步时，可执行 Post-Gate Simulation；

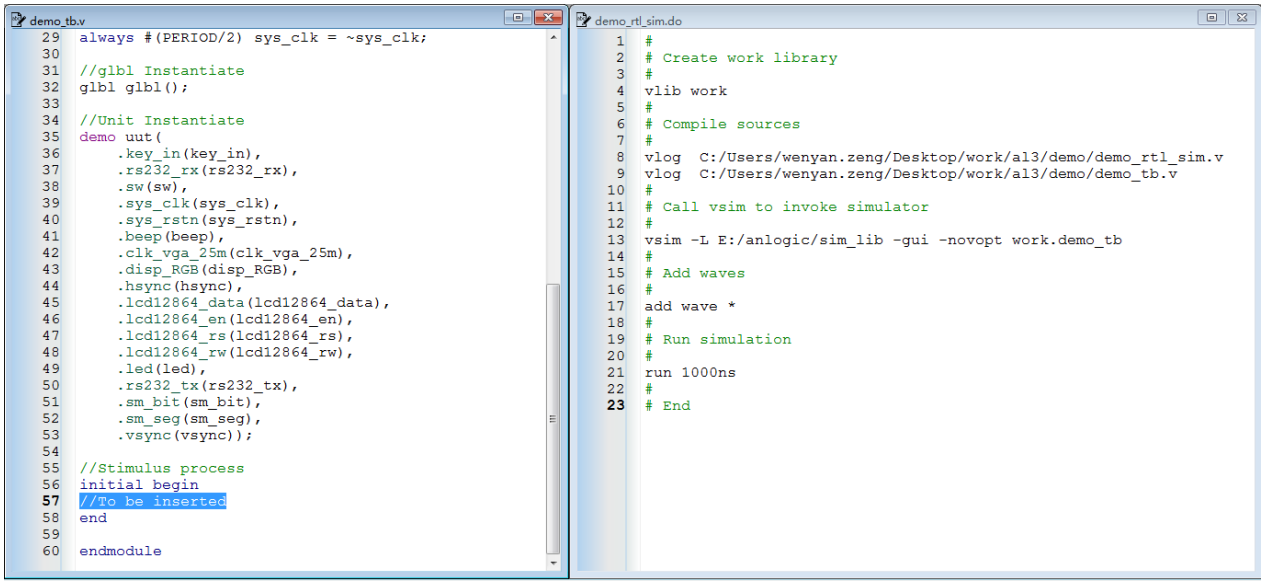
当 HDL2Bit Flow 运行至 Optimize Routing 这一步时，可执行 Post-Route Simulation。

5. 定义 testbench 文件

如点击 Post-RTL Simulation，则会弹出如下对话框，可以添加一个已经存在的 testbench 文件，也可以新建一个 testbench 文件。在新建的时候，需要指定对应的 module。



点击 OK 后，将会在工程目录下生成 prj_tb.v 和 prj_name_rtl_sim.do 并在 TD 界面打开这些文件。注意，prj_tb.v 中并没有给激励，在做仿真前，需手动填写。



demo_tb.v

```
29  always #(PERIOD/2) sys_clk = ~sys_clk;
30
31  //glbl Instantiate
32  glbl glbl();
33
34  //Unit Instantiate
35  demo uut(
36      .key_in(key_in),
37      .rs232_rx(rs232_rx),
38      .sw(sw),
39      .sys_clk(sys_clk),
40      .sys_rstn(sys_rstn),
41      .beep(beep),
42      .clk_vga_25m(clk_vga_25m),
43      .disp_RGB(disp_RGB),
44      .hsync(hsync),
45      .lcd12864_data(lcd12864_data),
46      .lcd12864_en(lcd12864_en),
47      .lcd12864_rs(lcd12864_rs),
48      .lcd12864_rw(lcd12864_rw),
49      .led(led),
50      .rs232_tx(rs232_tx),
51      .sm_bit(sm_bit),
52      .sm_seg(sm_seg),
53      .vsync(vsync));
54
55  //Stimulus process
56  initial begin
57  //To be inserted
58  end
59
60  endmodule
```

demo_rtl_sim.do

```
1  #
2  # Create work library
3  #
4  vlib work
5  #
6  # Compile sources
7  #
8  vlog  C:/Users/wenyan.zeng/Desktop/work/al3/demo/demo_rtl_sim.v
9  vlog  C:/Users/wenyan.zeng/Desktop/work/al3/demo/demo_tb.v
10 #
11 # Call vsim to invoke simulator
12 #
13 vsim -L E:/anlogic/sim_lib -gui -novopt work.demo_tb
14 #
15 # Add waves
16 #
17 add wave *
18 #
19 # Run simulation
20 #
21 run 1000ns
22 #
23 # End
```

在 Modelsim 中的具体仿真流程可参考该手册的 9.3 Modelsim 仿真流程。

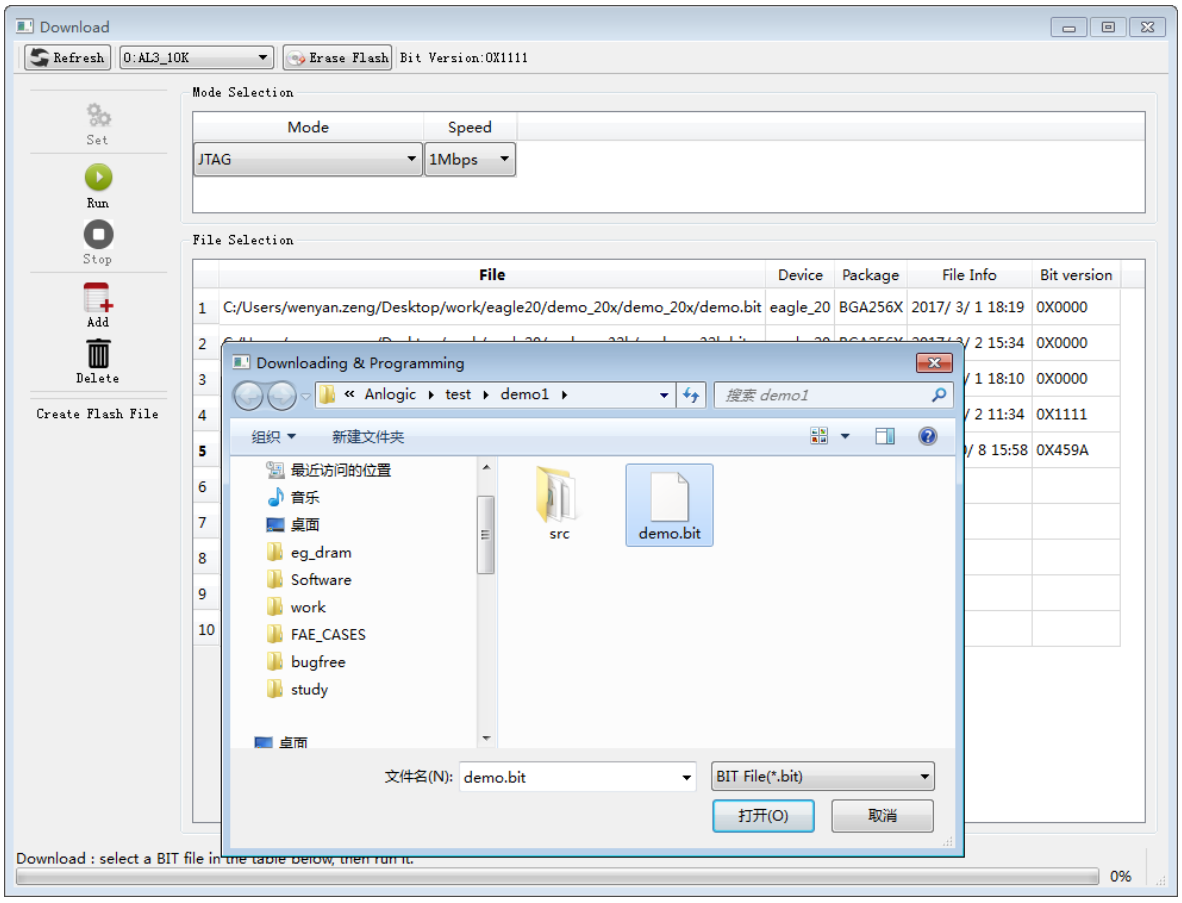
7 下载

7.1 下载流程简介

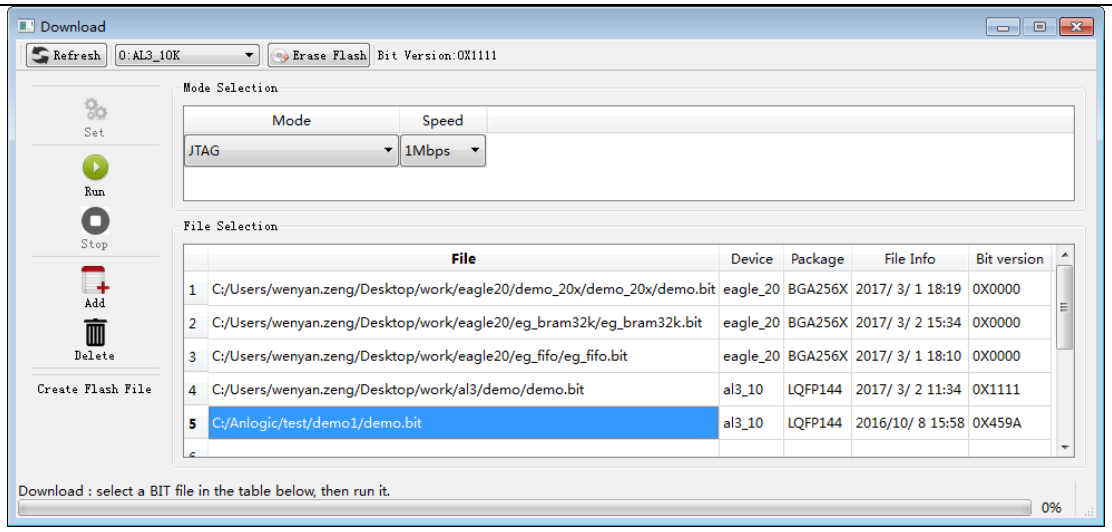
在成功生成位流文件后，可以将它们载入到 FPGA 芯片的配置存储器或 SPI Flash 存储器中。

1. 在 FPGA Flow 面板中，双击 **Download**

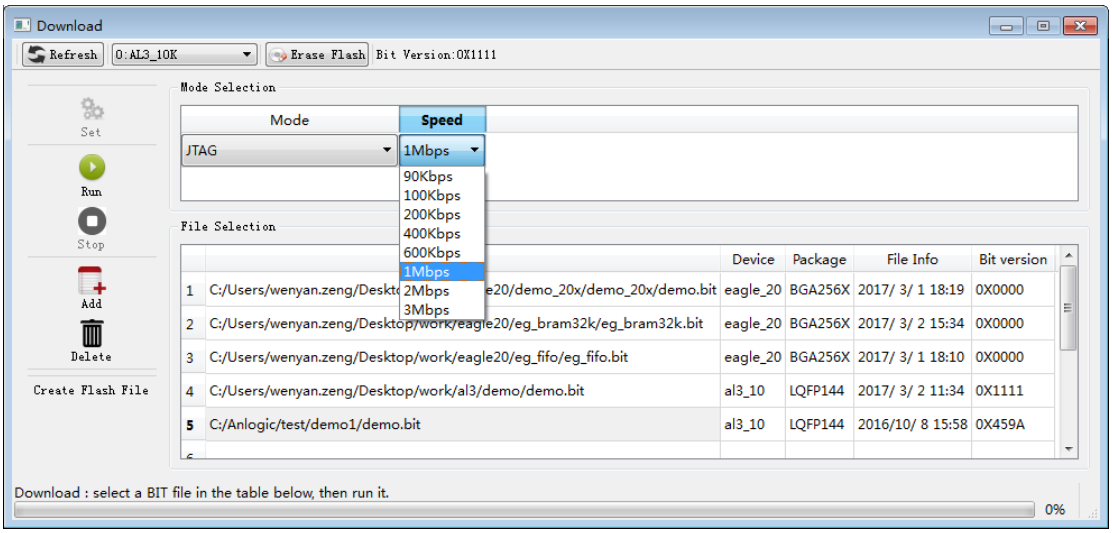
2. 通过 **Add** 添加需要下载的位流文件。



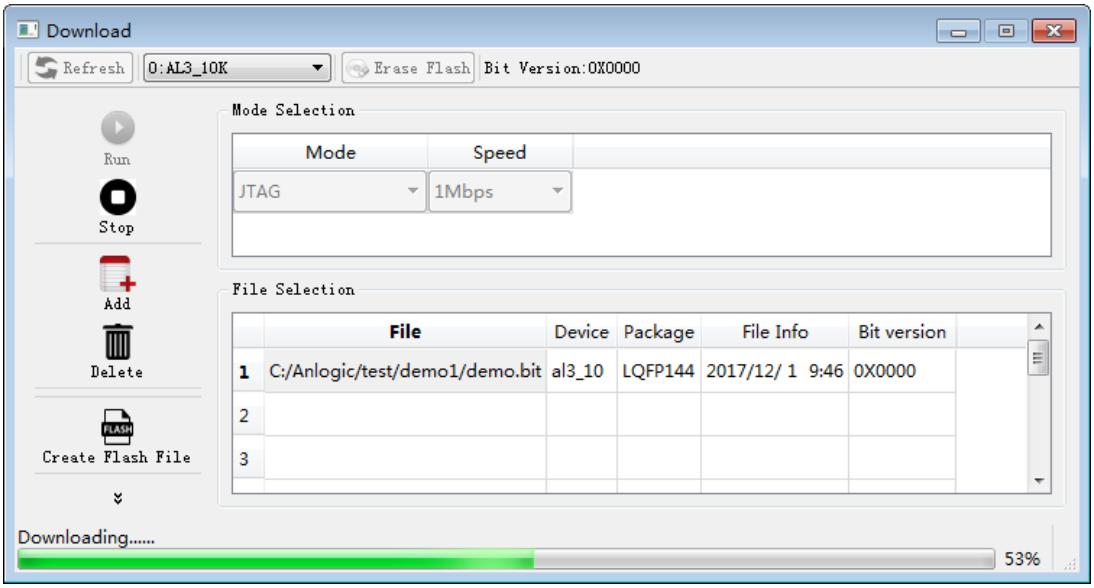
3. 选择相应的位流文件，点击 **Run** 进行下载。



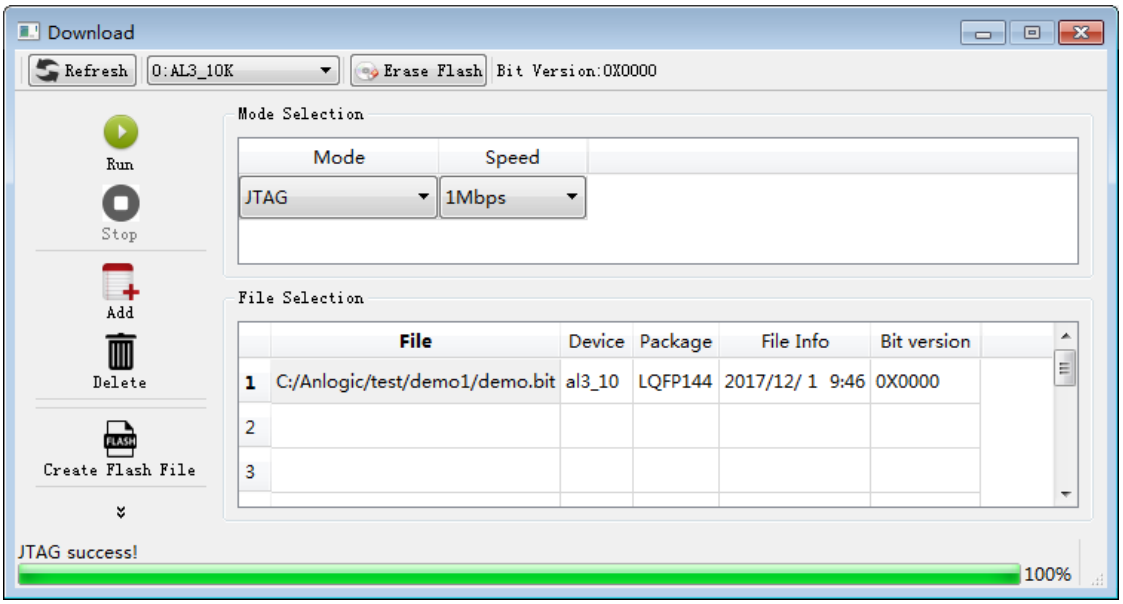
下载的速度分为九个等级，90Kbps 最慢，3Mbps 最快，默认为 1Mbps。



下载过程中可通过进度条查看下载进度。



下载完成后，将返回下载成功的提示。



TD 无法识别芯片的情况：

1. “No hardware”：用户在下载前，没有正确安装 USB 驱动，USB 下载驱动安装说明请参考附录 9.4。下载时，各接口未正确连接，请检查各接口处是否有松动，然后点击 **Refresh** 按钮进行刷新。
2. “USB Cable is connected”：下载时，没有识别到 FPGA 芯片或 Flash 芯片，请检查电路板电源是否打开，然后点击 **Refresh** 按钮进行刷新。

位流文件校验：

为确保客户文件及设备的安全，TD 软件在下载过程中，会进行芯片 ID 匹配检查，以及针对位流文件完整性与正确性的 CRC 校验：

1. 当位流文件中记录的芯片 ID 与下载目标芯片不符时，TD 软件会报错并终止下载；
2. 当位流文件不完整，或者内容有被篡改时，将无法通过 CRC 校验，无法继续下载。

7.2 位流文件类型

TD 软件中支持下载的位流文件、生成方式和下载操作如下：

1. **bit**：bit 文件包含完整的芯片配置及位流信息。

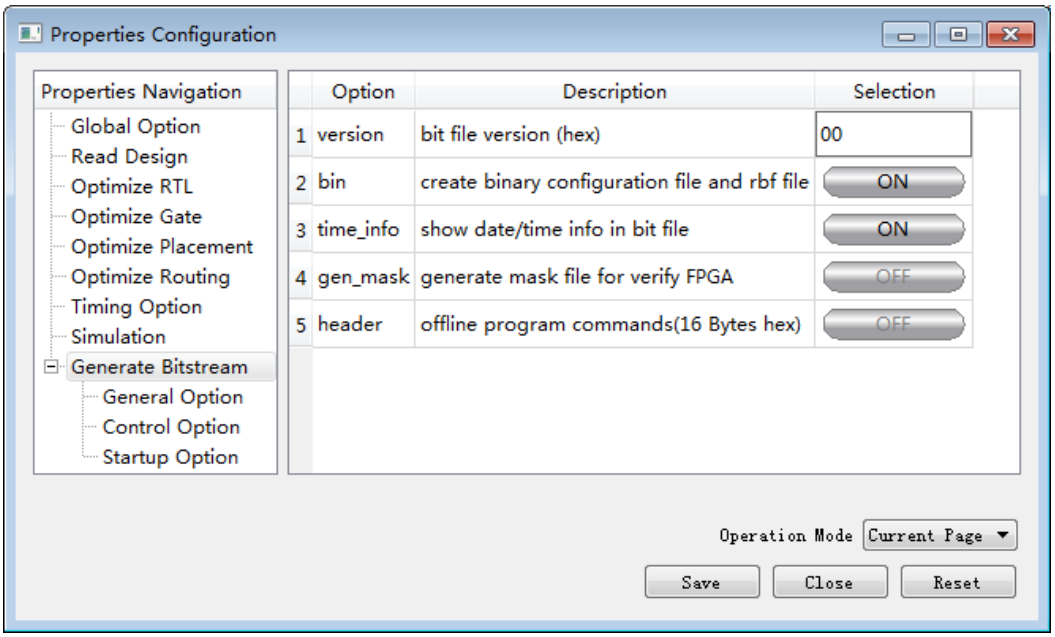
   在 TD 界面中运行 Generate Bitstream 默认生成的即为 bit 文件。

   bit 文件可用于 TD 支持的任何一种下载模式。

2. **bin**：仅包含位流信息的纯二进制文件。

   在 **Process→Properties→Generate Bitstream→General Option** 将 bin 选项的值设为 ON 并保存，运行完 Generate Bitstream 后将在工程目录中生成相应的 bin 文件。

   bin 文件可供离线下载器下载，支持的下载模式为 Direct Flash Write、Program SPIBIN。

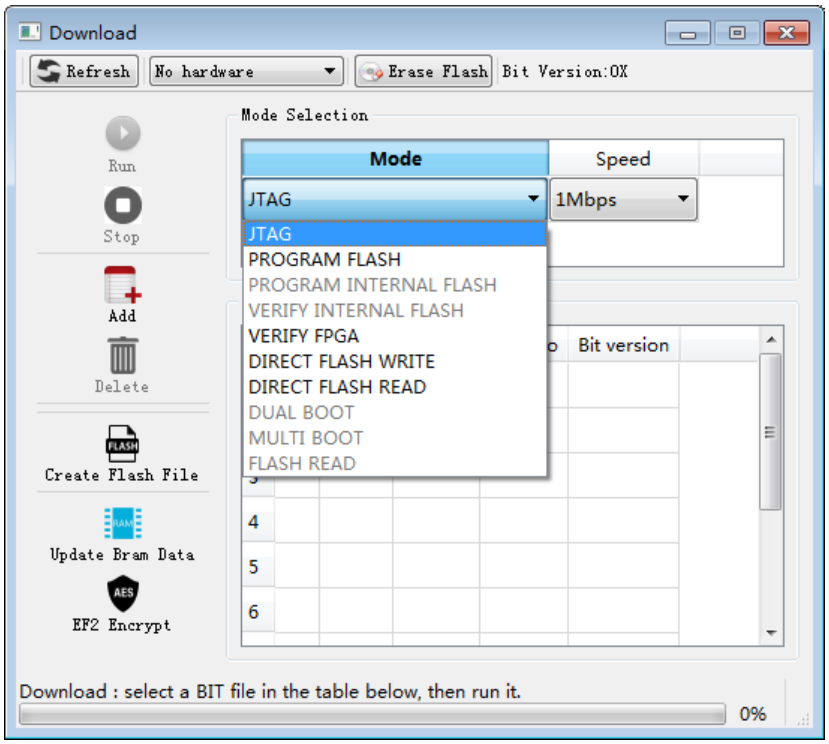


3. **svf**：串行向量文件。用于屏蔽内部细节而提供的统一标准结构。

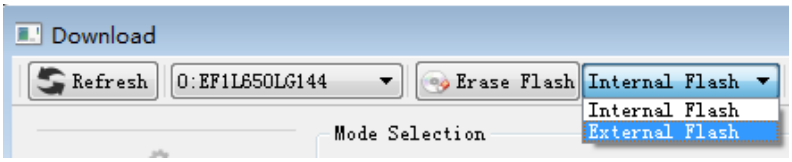
   svf 文件的生成与下载将在 Device Chain 中做详细介绍。

7.3 下载模式

TD 提供以下几种下载模式供用户选择：**JTAG、PROGRAM FLASH、PROGRAM INTERNAL FLASH、VERIFY INTERNAL FLASH、VERIFY FPGA、DIRECT FLASH WRITE、DIRECT FLASH READ、DUAL BOOT、MULTI BOOT、FLASH READ。**



1. **JTAG** 模式：下载的 bit 文件不会被保存到 flash 中，配置位信息被直接存在 FPGA 芯片中控制编程开关，电路板断电后配置位信息就完全丢失。
2. **PROGRAM FLASH** 模式：bit 文件将被保存至外置的 Flash 芯片中，电路板掉电重启后 FPGA 芯片自动读取保存在 Flash 芯片中的位流信息。若想擦除 flash 中的位流信息可点击 **Erase Flash** 按钮，擦除时间取决于 Flash 芯片的器件参数。对于 ELF 系列的器件，该功能用于外部 FLASH 的下载。在擦除外部 FLASH 时，需选择 **Erase External Flash**。

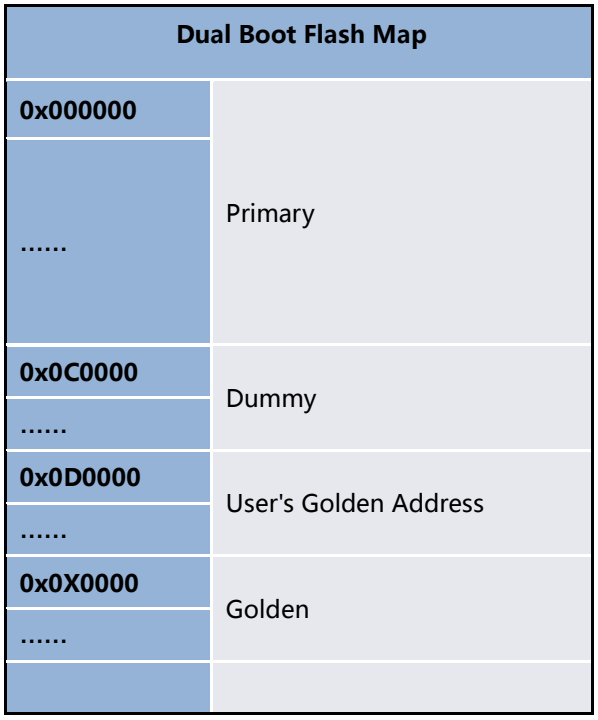


3. **PROGRAM INTERNAL FLASH** 模式：仅支持 ELF 系列的器件，用于 internal flash 的下载。在擦除内部 FLASH 时，需选择 **Erase Internal Flash**。
4. **VERIFY INTERNAL FLASH** 模式：仅支持 ELF 系列的器件，用于比较 internal flash 中的配置文件与用户当前选中的 bit 文件中的信息是否一致。
5. **VERIFY FPGA** 模式：用于比较 FPGA 芯片中的配置位信息和用户当前选中的 bit 文件中的信息是否一致，最好配合遮罩文件(.bmk)一同使用，保证位流文件和遮罩文件在同一个文件夹内。
6. **DIRECT FLASH WRITE** 模式：不经过 FPGA，直接将数据写入 FLASH 指定地址区域中，硬件上需要下载器直接与 FLASH 的信号线相连。用于离线下载器的下载，仅支持下载 bin 文件。
7. **DIRECT FLASH READ** 模式：不经过 FPGA，直接从 FLASH 中读出指定区域的数据，并存在指定文件中，该模式同样需要下载器直接与 FLASH 的信号线相连。

7.3.1 Dual Boot

对于 Eagle、EF2 系列的 FPGA，在对外部 flash 进行程序加载时，支持 Dual Boot（双启动模式）和 Multi Boot（多启动模式）。

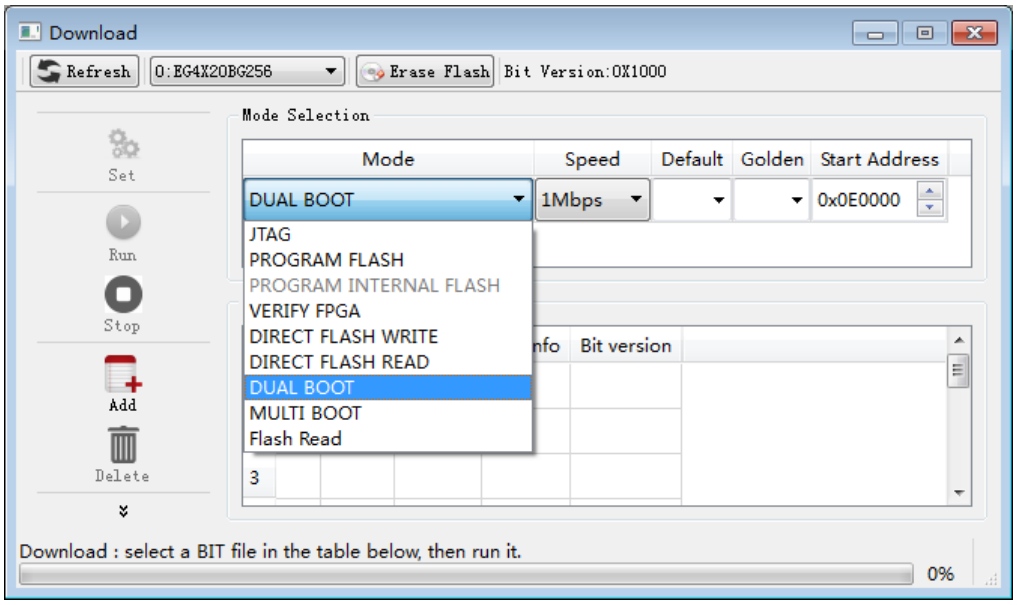
双启动模式是指在 SPI FLASH 中存放了两套 FPGA 位流，上电后 FPGA 首先加载 Primary 位流，如果 Primary 位流出错导致加载失败，则会根据 Golden Address（跳转地址）去加载 Golden 位流。双启动模式下，数据空间分布如下图所示：



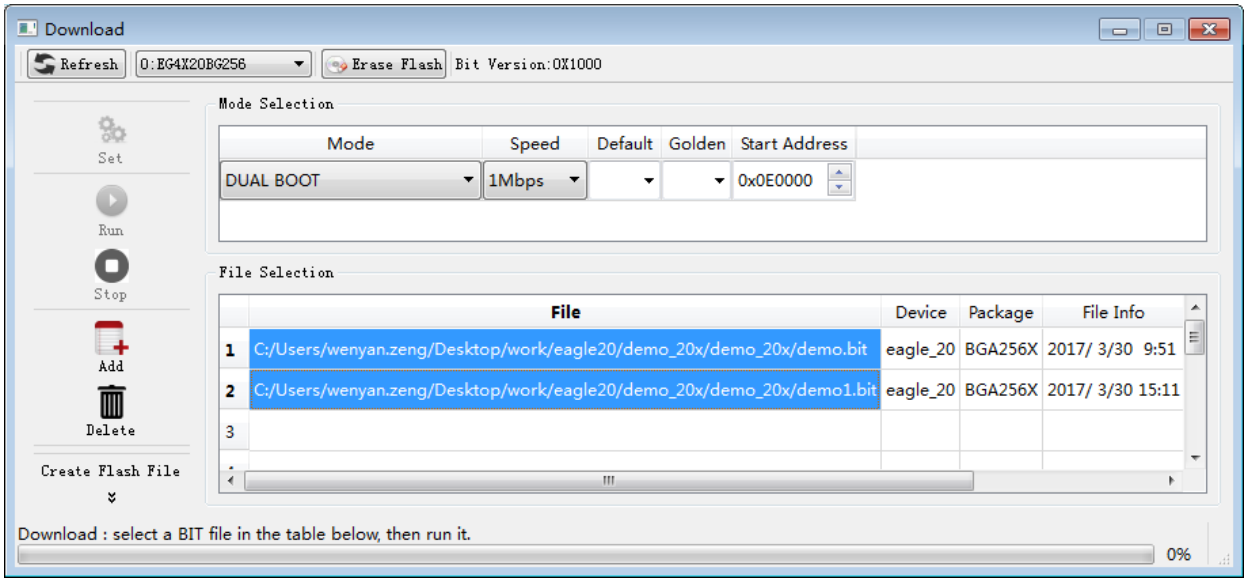
Eagle 系列 FPGA 在默认情况下就支持双启动模式，即在上电后会默认从 FLASH 的 0 地址加载程序，如果 0 地址开始的位流被破坏导致 FPGA 程序加载失败，则 FPGA 会到 0X0D0000 地址读取跳转地址，然后从指定的跳转地址去加载位流。

使用双启动模式的步骤为：

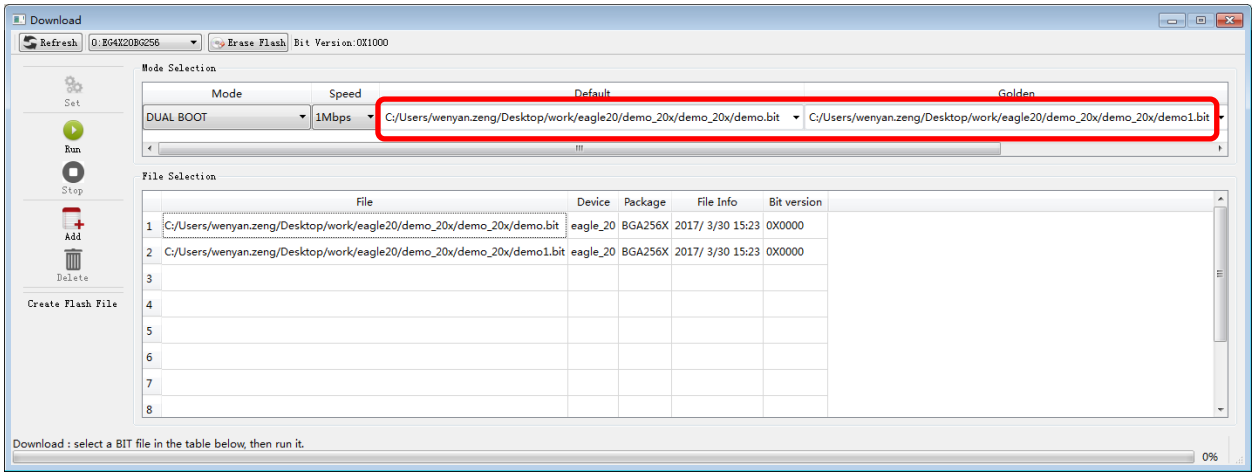
1. 在 **Download** 界面中，选择下载模式为 **Dual Boot**;



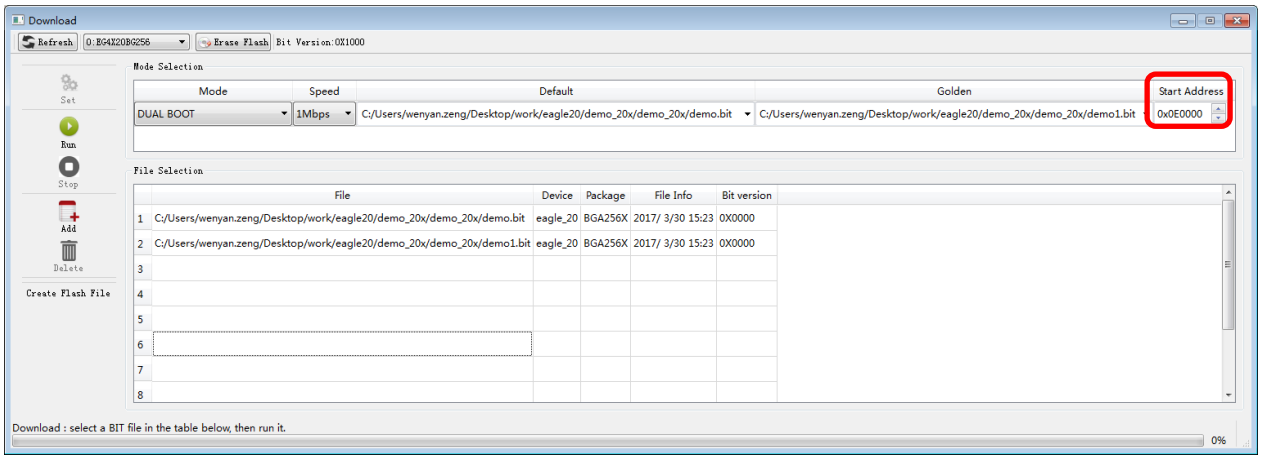
2. 通过 **Add** 按钮添加需要下载到 FLASH 中的两个位流文件;



3. 设置 **Default** 位流（Primary 地址段）和 **Golden** 位流（Golden 地址段）；



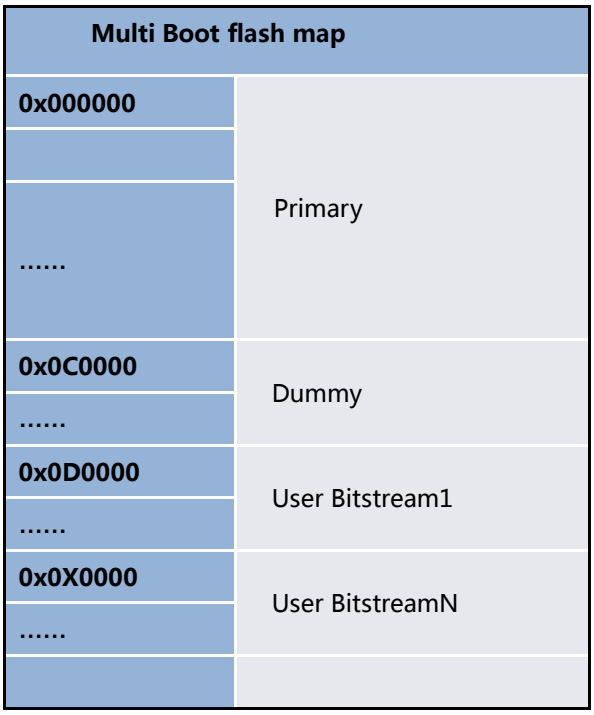
4. 设置 Golden 位流存放的起始地址。注意，Golden 存放地址只能大于 0X0D0000，默认为：0X0E0000；



5. 点击 **Run** 下载位流。

7.3.2 Multi Boot

多启动模式是指用户可以在 SPI FLASH 中存放两套或者多套 FPGA 位流，上电后 FPGA 首先加载 Primary 位流，然后在 Primary 位流的 FPGA 代码中可以控制 FPGA 从指定地址加载位流。多启动模式下，数据空间分布如下图所示：

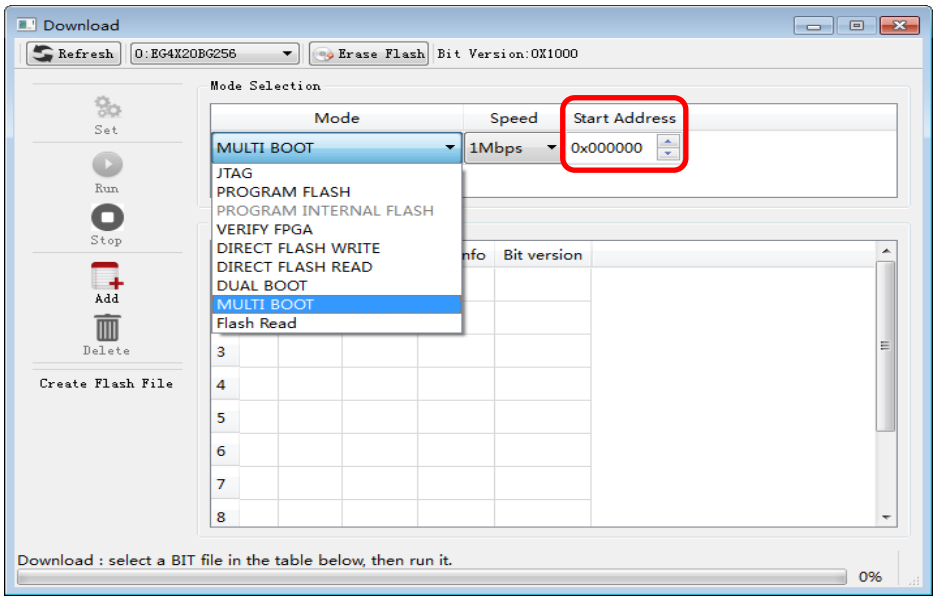


使用多启动模式前，用户需要在代码中调用以下 IP 单元：

EG_LOGIC_MBOOT #("DYNAMIC",8'h00) mboot(rebootn, dynamic_addr);

其中，dynamic_addr 为 8 位 FLASH 地址，是 24 位 FLASH 地址的高 8 位，在设置好地址后，通过给 rebootn 一个低脉冲，即可实现从 dynamic_addr 指定地址重新加载 FPGA 程序。

在 TD **Download** 界面中，需设置下载模式为 **MULTI BOOT**，并指定用于跳转的 bit 文件的 **Start Address**，该地址需和 dynamic_addr 相同。

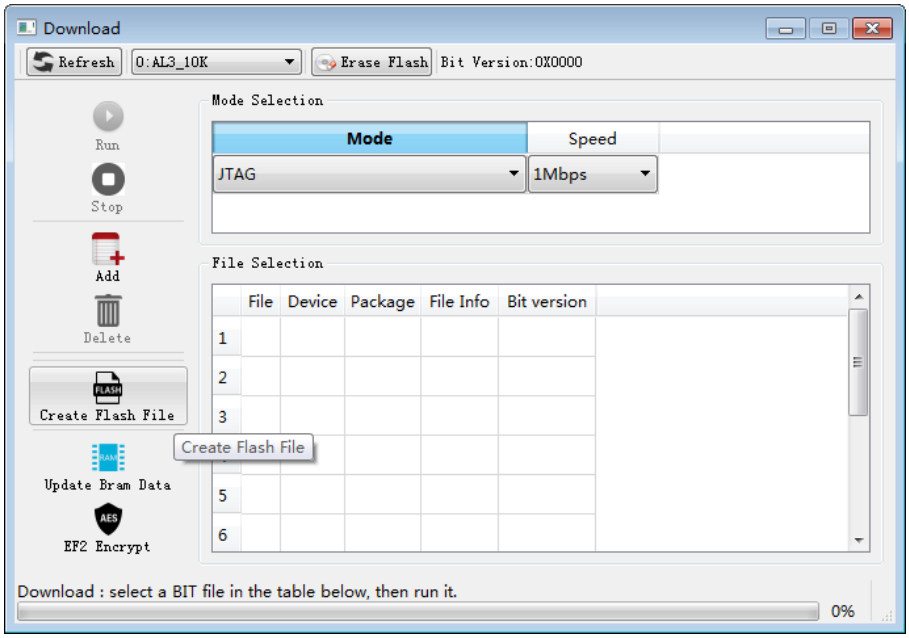

Download
Refresh
0:EG4X20BG256
Erase Flash
Bit Version:0X1000
Set
Run
Stop
Add
Delete
Create Flash File
Mode Selection
Mode
Speed
Start Address
MULTI BOOT
1Mbps
0x000000
JTAG
PROGRAM FLASH
PROGRAM INTERNAL FLASH
VERIFY FPGA
DIRECT FLASH WRITE
DIRECT FLASH READ
DUAL BOOT
MULTI BOOT
Flash Read
Bit version
3
4
5
6
7
8
Download : select a BIT file in the table below, then run it.
0%

7.4 扩展功能

7.4.1 Create Flash File

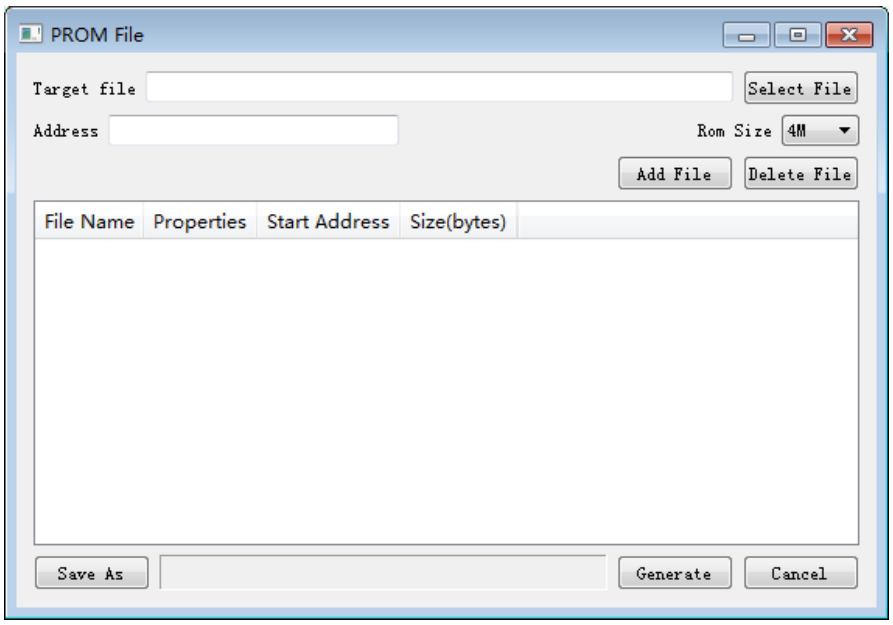
TD 软件提供 Create Flash File 功能，方便用户在目标位流文件中添加自定义的内容，扩展已有位流文件的功能，而需要重新修改源代码，节省大量时间。具体操作如下：

1. 打开 **Download** 界面，点击“**Create Flash File**”；

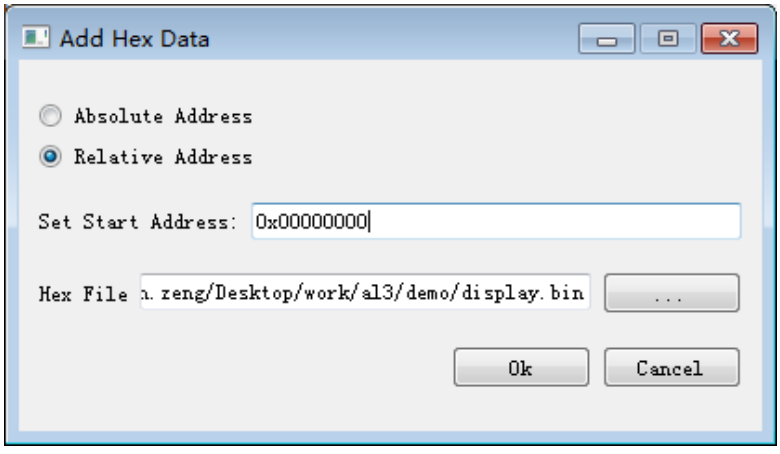


2. 点击“**Select File**”添加 **Target File**，目标文件可以为 bit 文件，也可以为 bin 文件。添加目标文件后，会相应的显示该文件的大小，即 **Address**。

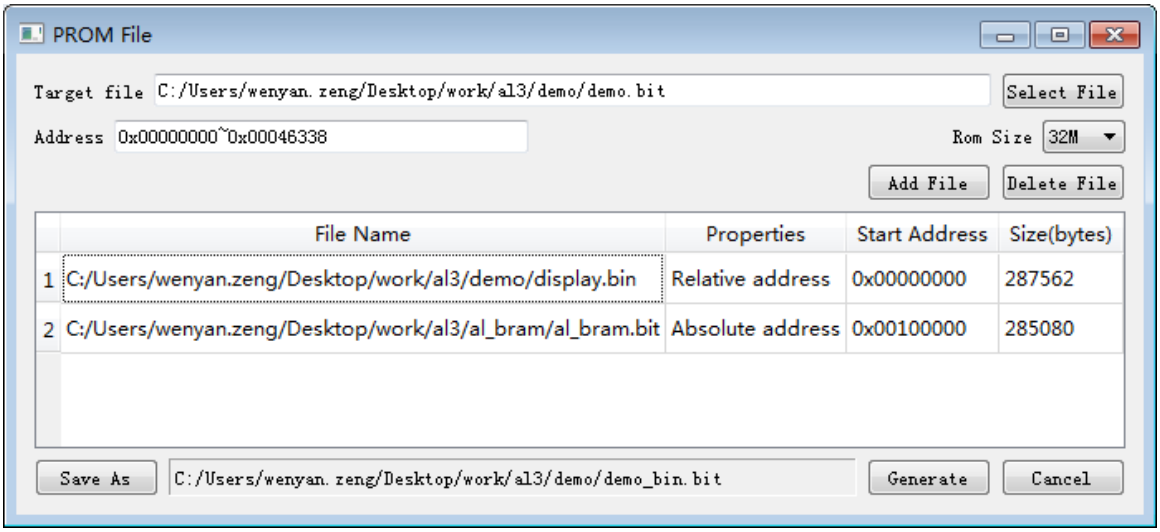
   Rom Size 指目标 Flash 的容量，即最后生成的文件不能超过该容量，否则在 Generate 时会失败。

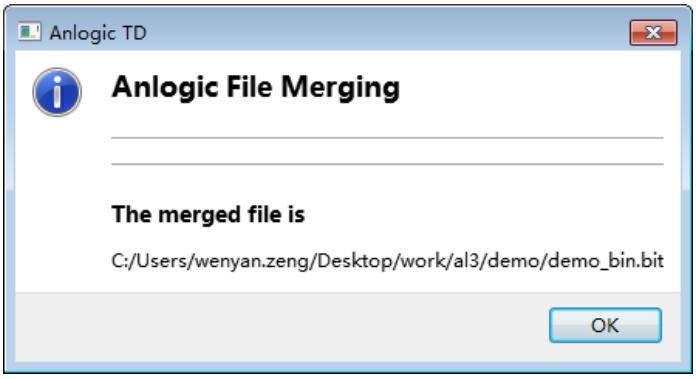


3. 点击“**Add File**”添加文件，称该文件为合并文件，合并文件可以为 bin 文件、hex 文件或 bit 文件。若选择 **Absolute Address**，则需设置起始地址大于 Target File 的大小，否则添加文件后，会覆盖掉 Target File 的内容；若选择 Relative Address，指相对于 Target File 大小，在其后进行添加，如设置起始地址为 0x00，则紧跟在 Target File 后添加文件。



4. 可同时添加多个合并文件，只需根据各文件的大小，合理设置起始地址。
5. 选择生成文件的路径，点击 **Generate**，将会给出如下提示，否则会给出相应错误提示。

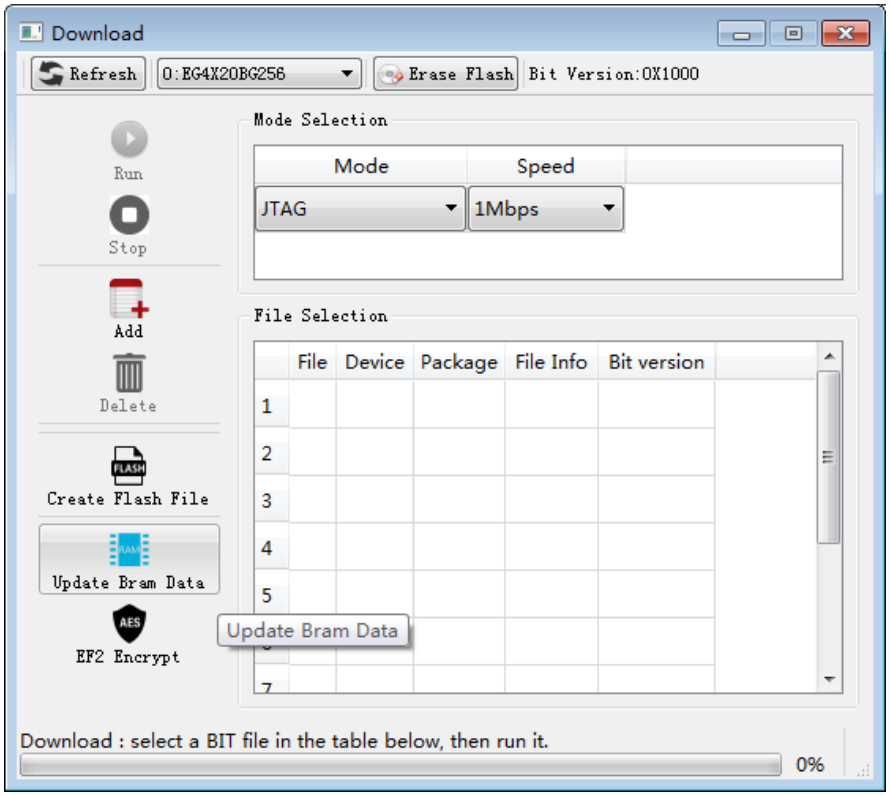

PROM File
Target file
C:/Users/wenyan.zeng/Desktop/work/al3/demo/demo.bit
Select File
Address
0x00000000~0x00046338
Rom Size
32M
Add File
Delete File
File Name
Properties
Start Address
Size(bytes)
1
C:/Users/wenyan.zeng/Desktop/work/al3/demo/display.bin
Relative address
0x00000000
287562
2
C:/Users/wenyan.zeng/Desktop/work/al3/al_bram/al_bram.bit
Absolute address
0x00100000
285080
Save As
C:/Users/wenyan.zeng/Desktop/work/al3/demo/demo_bin.bit
Generate
Cancel

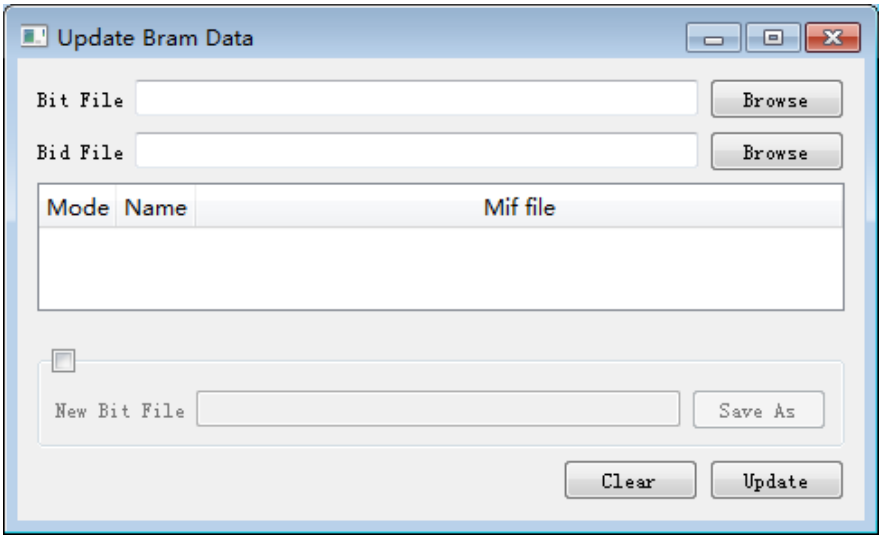

Anlogic TD
Anlogic File Merging
The merged file is
C:/Users/wenyan.zeng/Desktop/work/al3/demo/demo_bin.bit
OK

7.4.2 Update BRAM Data

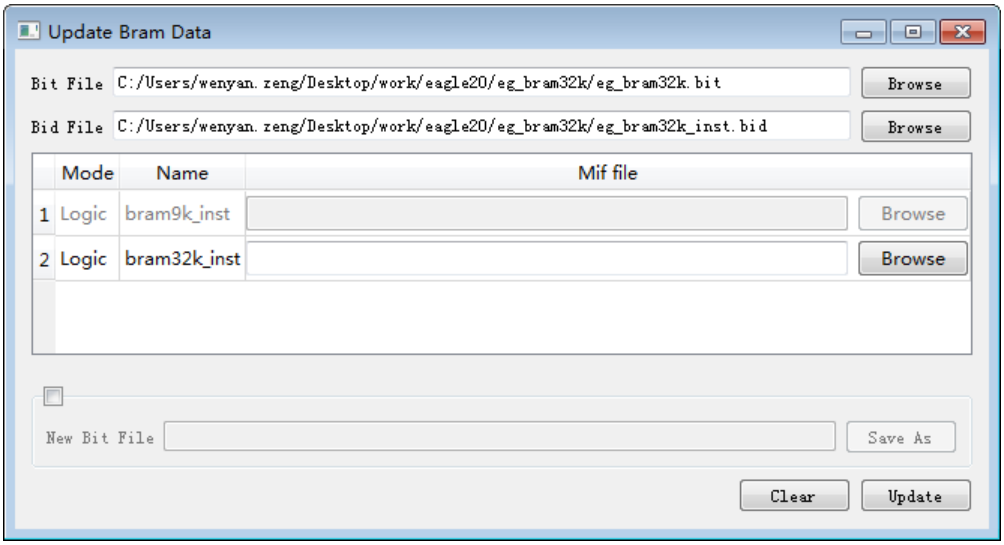
TD 提供 Update BRAM Data 工具，若只需更新设计中 BRAM 的初始值时，无需重新编译工程，直接修改位流文件中 BRAM 的数据段，即可生成新的位流文件，节省大量时间。具体操作如下：

1. 打开 **Download** 界面，点击“**Update Bram Data**”；

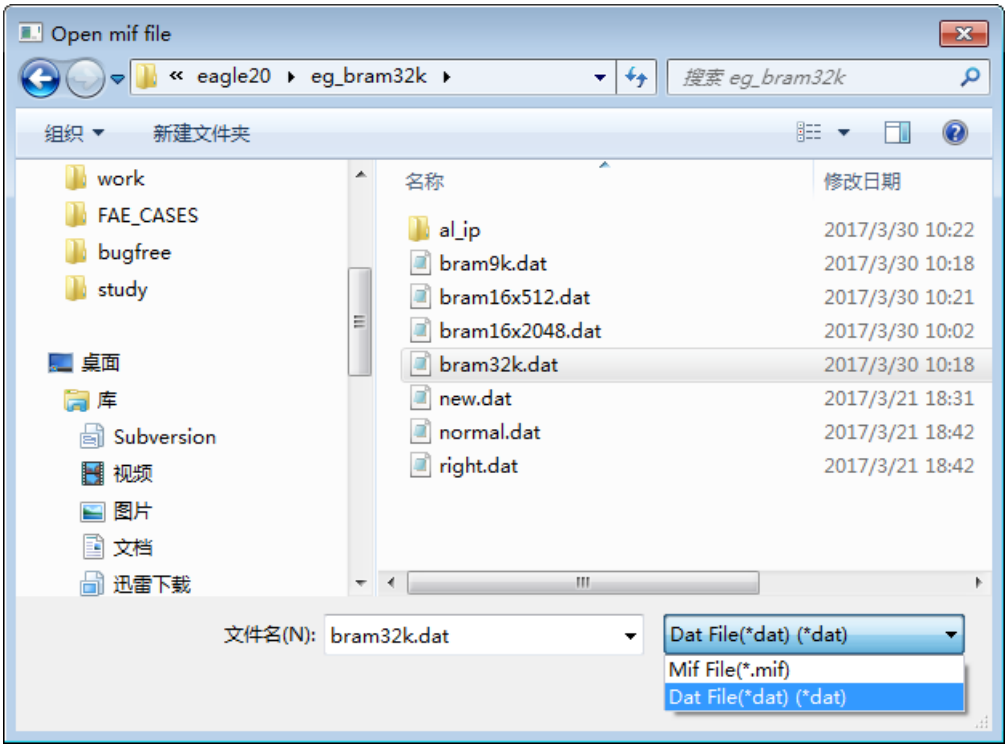




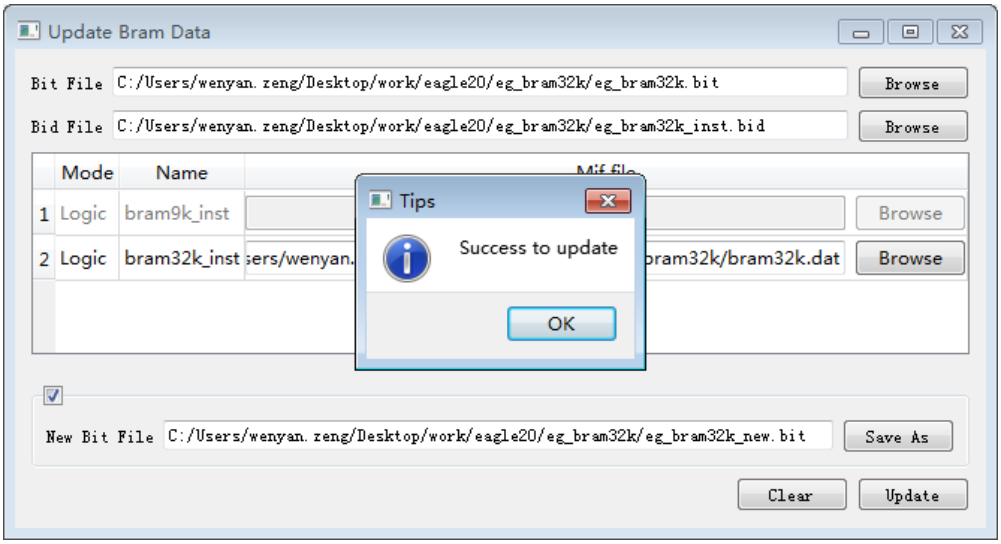
2. 选择需要更新的 **Bit File** 以及描述 BRAM 的 **Bid File**，则会相应的显示该位流中 Logic BRAM。只有在设计初期已添加了初始化文件的 BRAM 才能在这里进行更新；没有添加初始化文件的 BRAM 则灰色显示，如下图中的 bram9k_inst；



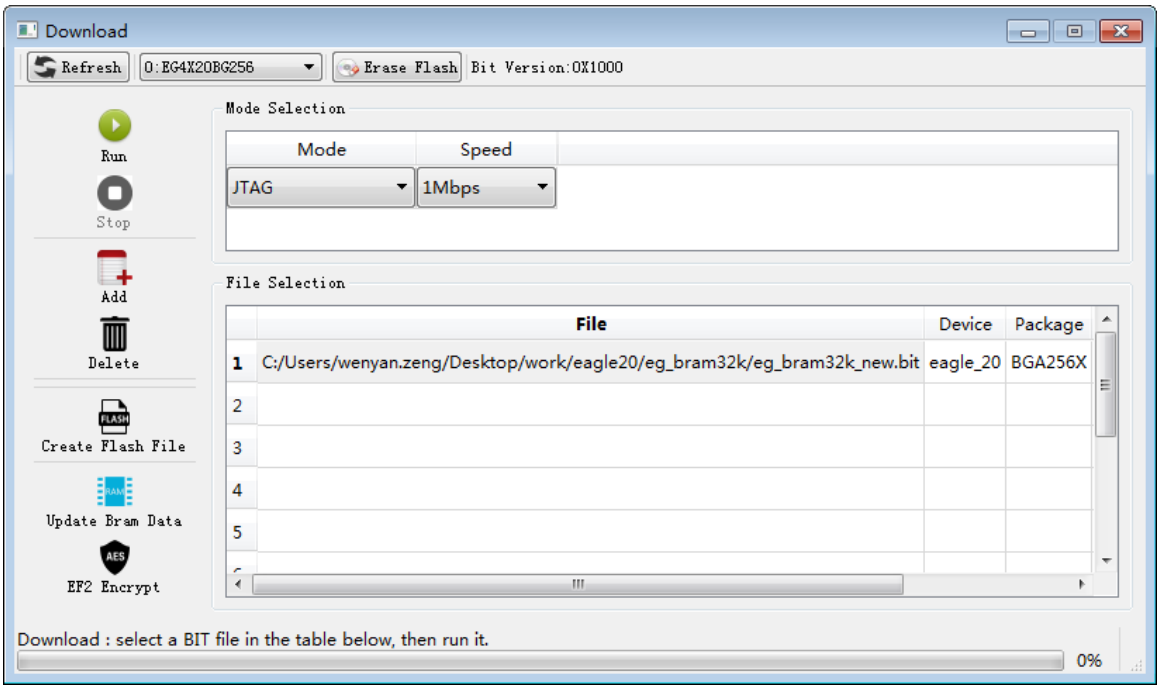
3. 点击“**Browse**”添加新的 BRAM Data，可添加的文件为.dat 文件和.mif 文件。这里，新的 BRAM Data 的大小需和设计中 BRAM 的大小一致，否则会给出警告，并达不到预期的功能；



4. 可点击 **Update** 直接更新当前的 bit 文件，也可通过 **Save As** 生成新的 bit 文件；



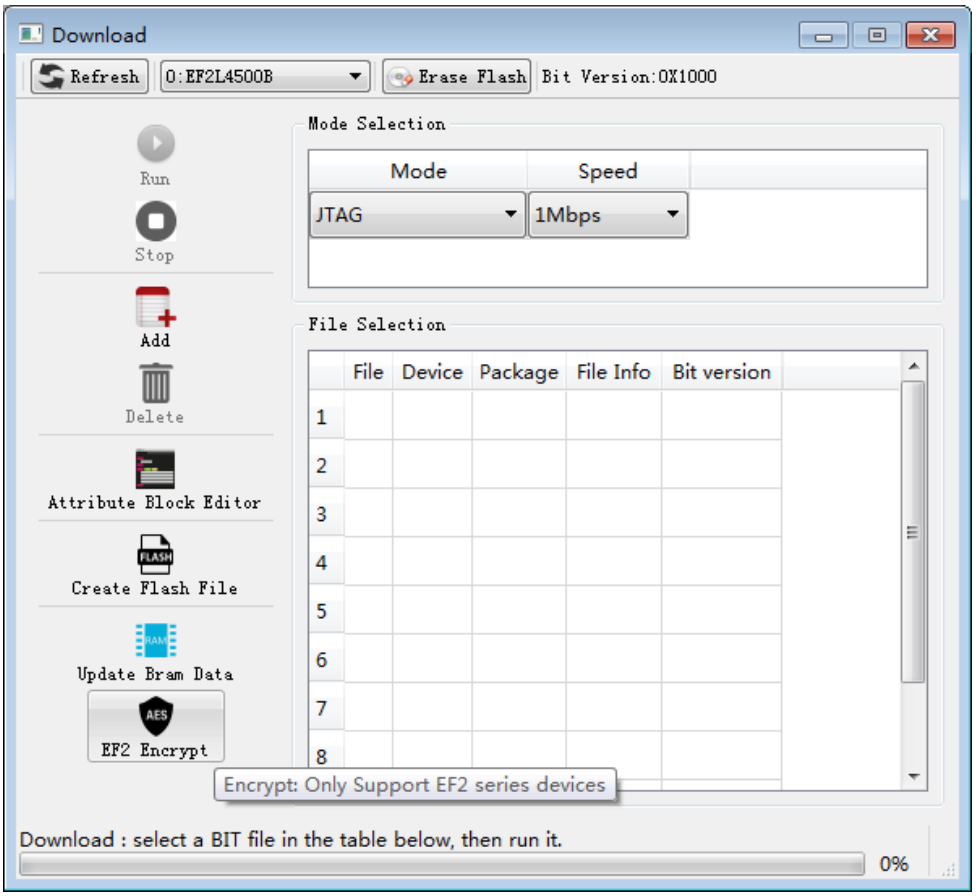
5. 点击 **OK**，完成更新，并在 **Download** 界面添加新的 bit 文件进行下载。

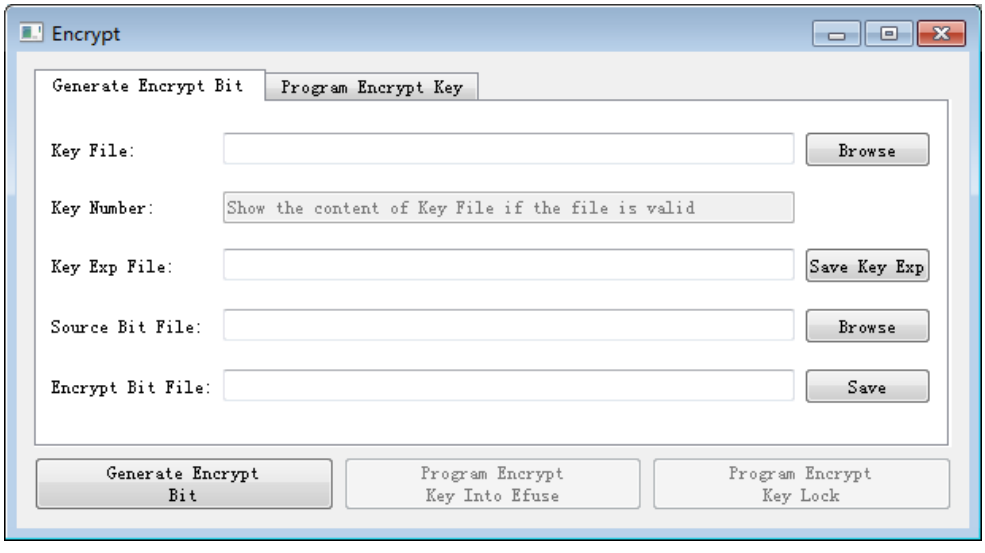


7.4.3 EF2 Encrypt

对于 EF2 系列器件，TD 软件支持对位流文件进行 128Bit AES 加密。加密后的文件必须由用户提供的秘钥才能进行解密。

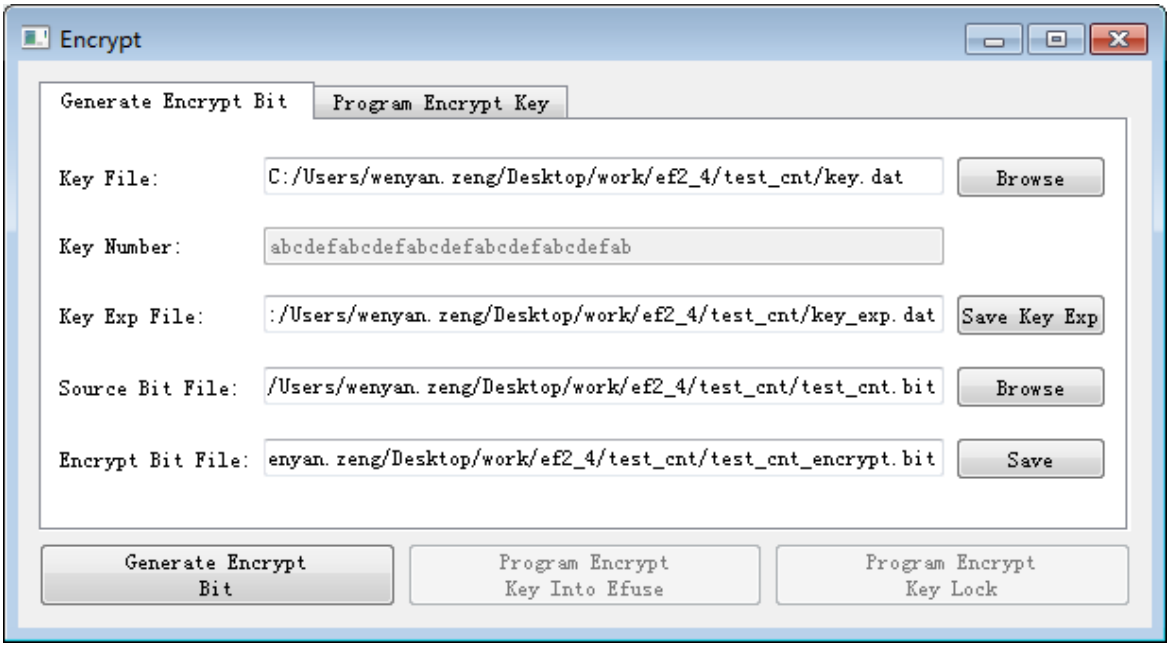
打开 Download 界面，点击 EF2 Encrypt 按钮，出现如下界面：

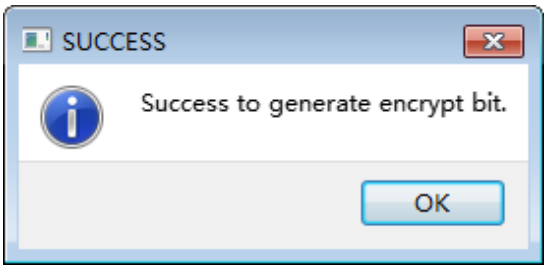




加密步骤如下:

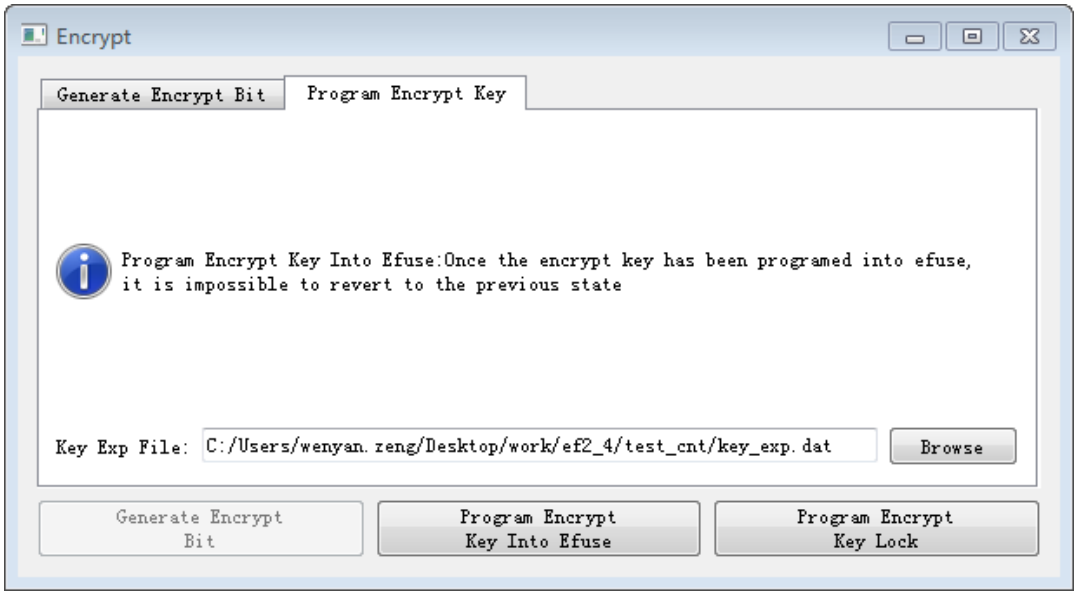
1. 在 Key File 中选定用户秘钥文件。用户秘钥是 32 个 16 进制数。选定秘钥后将在 Key Number 中进行显示。
2. 在 Key Exp File 中选择加密后秘钥文件保存的位置。
3. 在 Source Bit File 中选择需要加密的 EF2 位流配置文件(*.bit 文件)。
4. 在 Encrypt Bit File 中选择加密后的位流配置文件的保存位置。
5. 点击 Generate Encrypt Bit 按钮，生成加密文件。若加密成功会给出提示，否则加密失败会在 TD 界面给出相应的错误。

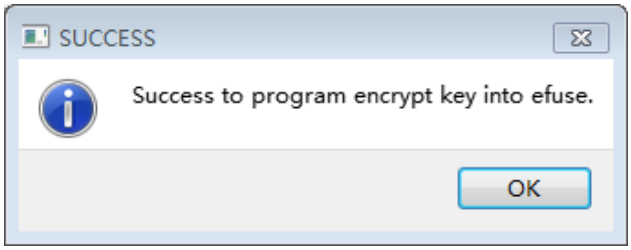


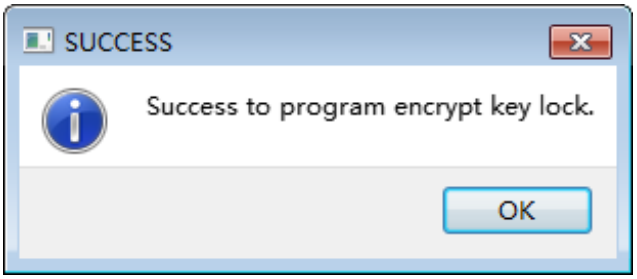


生成完加密文件后需要将加密秘钥文件的内容烧写到 EF2 芯片内部，烧写流程如下：

1. 在 Encrypt 界面中，选择 Program Encrypt Key 界面。
2. 在 Key Exp File 中选择刚生成的加密秘钥文件 key_exp.dat。
3. 点击 Program Encrypt Key Inot Efuse 按钮，烧写加密秘钥。
4. 烧写完加密秘钥后，点击 Program Encrypt Key Lock 按钮，加密秘钥将被硬件锁定，将不能从 EF2 器件中读取加密秘钥。







7.5 离线下载器

7.5.1 离线下载器的介绍

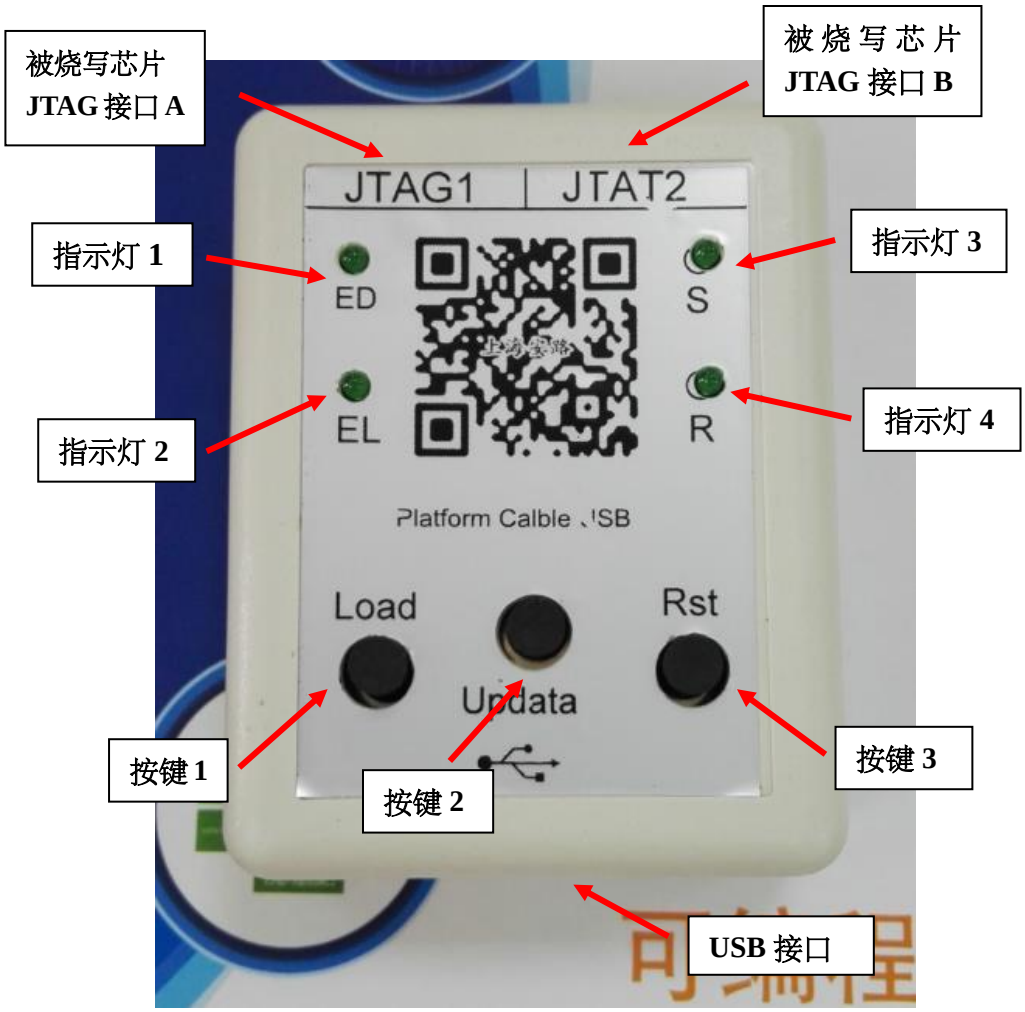
离线下载器同时支持在线 JTAG 程序下载（兼容传统下载器），在线 FLASH 直接读写，离线 FLASH 程序烧写三种模式。其中，离线 FLASH 程序烧写模式又分为以下三种模式：

第一，支持通过 JTAG 烧录 SPI FLASH

第二，支持直接烧录 SPI FLASH

第三，支持用户自定义烧写协议

离线下载器的硬件介绍如下：



离线下载器工作状态介绍如下：

上电后离线下载器处于在线 JTAG 下载模式，此时该下载器就像一个普通下载器，可以通过被烧写芯片 JTAG 接口与目标板上 FPGA 的 JTAG 连接，实现对目标板 FPGA 的程序下载，包括在线 JTAG 调试和目标板 FLASH 在线下载。 此种状态下指示灯 1，指示灯 2，指示灯 3 均熄灭，指示灯 4 闪烁。

当按下按键 2，此时离线下载器进入 FLASH 直接读写模式，此时 PC 软件可以读写离线下载器上用于离线模式时使用的源 FLASH 的内容。在此模式下指示灯 3 闪烁，指示灯 4 则保持闪烁。

当按下按键 1，此时离线下载器进入离线下载模式，将不再受 PC 软件控制，离线下载器将自动读取源 FLASH 的 16 个字节头，然后根据字节头进入三种离线下载模式的其中一种，然后进行 FLASH 复制。 在此模式下指示灯 4 常亮，指示灯 1 亮代表检测到目标 FLASH 的 ID 有错，有可能没检测到目标 FLASH；指示灯 2 亮代表检测到复制 FLASH 数据时比对错误；指示灯 3 亮代表正确的完成一次 FLASH 复制，此时可以换取下一块芯片，再按一次按键 1，继续下一次烧写。

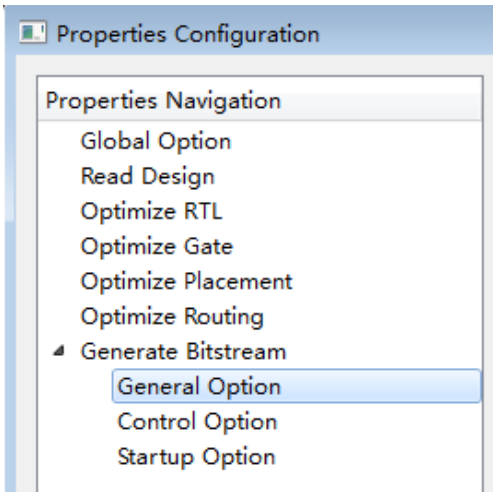
任何模式下，通过按下按键 3 均可使离线下载器回到上电初始模式，即在线 JTAG 下载模式。

7.5.2 离线下载器的使用步骤

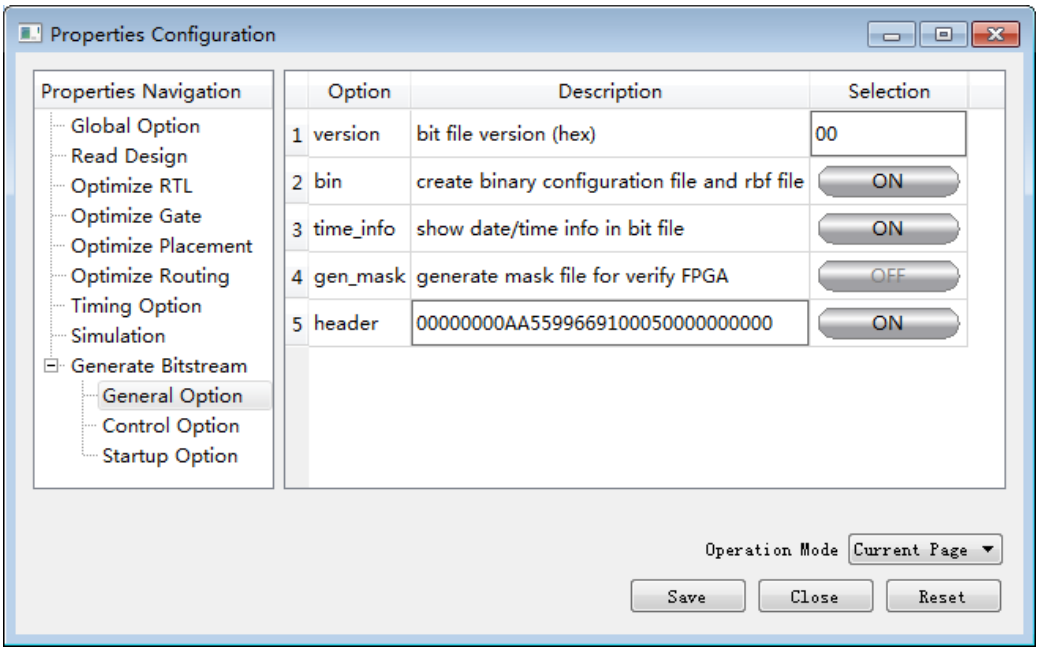
1. 生成可下载至源 FLASH 的 BIN 文件

该 BIN 文件的特点是在目标 FPGA 运行的 bit 文件基础上添加了 16 个字节的头，在编译工程之前通过对软件进行设置，可添加这 16 个字节头，具体步骤如下：

步骤一，打开软件属性设置窗口 **Properties→General Option**



步骤二，将 **bin** 选项的值设为 **ON**，并在 **header** 栏添加 16 个字节的包头。



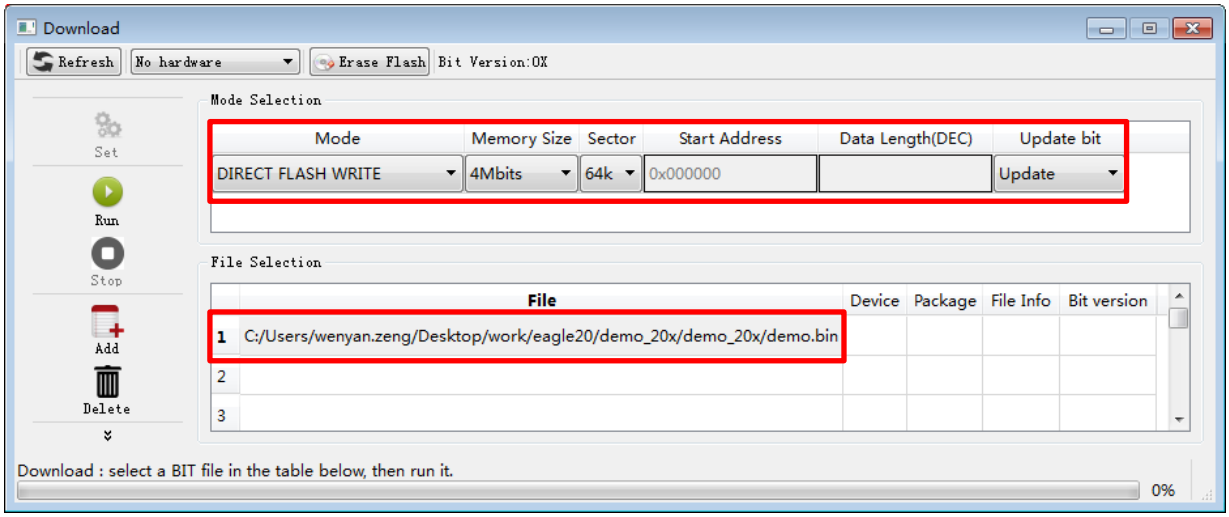
包头格式的定义如下表：

Byte	**示例（16 进制）**	**含义**
0~3	0000_0000	填充位
4~7	AA55_9966	识别码
8	11	命令字： Bit7~6 选择擦除命令： 01： sector erase 4K ~50ms 10: block erase 64K~0.2s 11: chip erase full chip ~1.5s Bit5~4 选择是否回读校验： 01：回读 10：不读 Bit3 选择是自动连接测试： 0：不自动测试 1：空闲状态自动测试连接 Bit2:0 选择烧写模式： 001： 通过 AL3 烧录 SPI FLASH 010： 直接烧录 SPI FLASH 011： 用户自定义
9~10	800	烧录长度，page 数量，最小 1 page ，**低字节在前**
11~12	400	烧录起始地址，page 数量，0 地址对应 0，**低字节在前**
13~15	00_0000	填充位

按照该表格填写好包头后，保存设置，重新编译目标工程，会在工程路径生成包含设置包头的 BIN 下载文件。

2. 将生成的 BIN 文件下载至离线下载器的源 FLASH

编译完成后，打开软件的 Download 界面，分别设置好下图红色框内的内容。注意，Start Address 设置为 0；Data Length 为 BIN 文件的大小，十进制表示，单位为 Byte；Sector 为被烧写芯片的 Sector 大小，通常为 64K。选中刚才生成的 BIN 文件，按下离线下载器的中间按钮（按键 2），然后点击 **Run**，即开始对离线下载器上的源 FLASH 进行烧写。



3. 对目标 FPGA 的 FLASH 进行下载

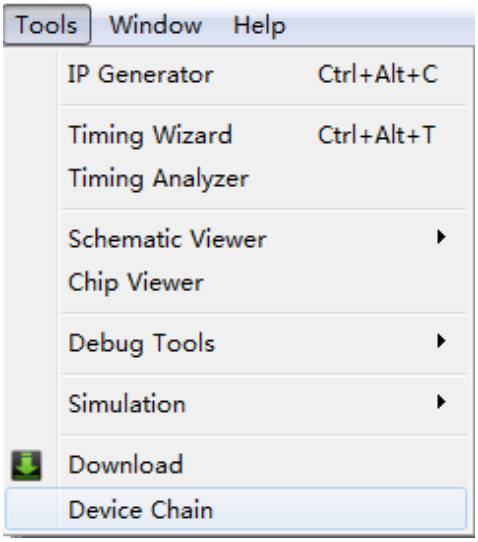
将离线下载器与目标 FPGA 的 JTAG（或者相应端口）连接好，按下离线下载器上的按键 1 即可开始对目标 FPGA 的 FLASH 进行程序下载。成功下载后，指示灯 3 点亮，蜂鸣器响 2s。下载完后，重复按按键 1，可反复对目标 FPGA 的 FLASH 进行下载。如果下载过程中指示灯 1 亮，表示目标 FLASH ID 错误；指示灯 2 亮，表示数据比较错误。若下载过程存在错误，蜂鸣器会一直响，直到按下复位按钮（按键 3）才停止。

7.6 Device Chain

为满足用户对多个 CPLD 或 FPGA 的 JTAG 级联加载的需求，TD 提供 Device Chain 功能，用于生成并下载 SVF 文件，可选择一个芯片进行下载，也可对 SVF 文件进行合并再一次性下载，避免用户多次执行下载操作。

具体操作如下:

1. 打开 **Device Chain : Tools → Device Chain**;



出现如下图所示的 Device Chain 界面，界面左侧为工具栏，Files Option 区域为文件列表，Device Chain 区域为芯片级联显示区域。

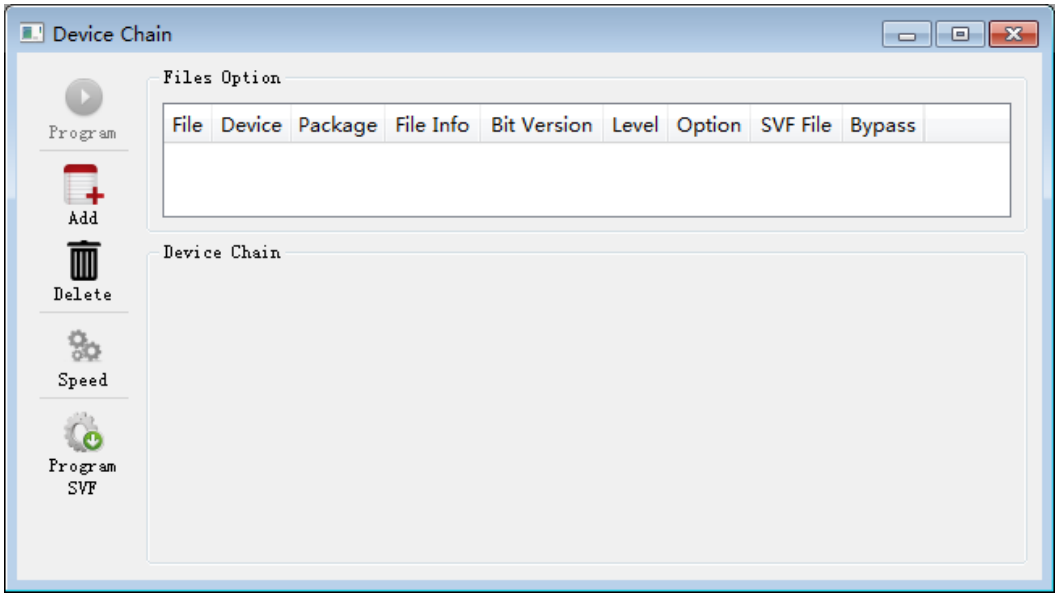
Program : JTAG 加载按钮;

Add : 添加文件按钮;

Delete : 删除文件按钮;

Speed : JTAG 加载时的速度选择按钮;

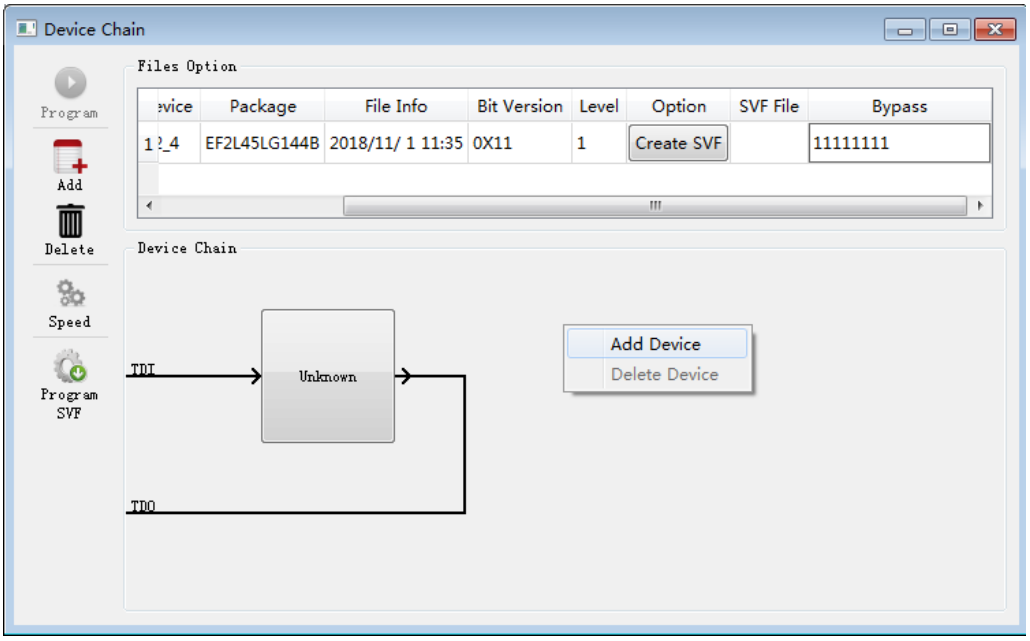
Program SVF : 加载合并后的 SVF 文件。



2. 添加文件与设备

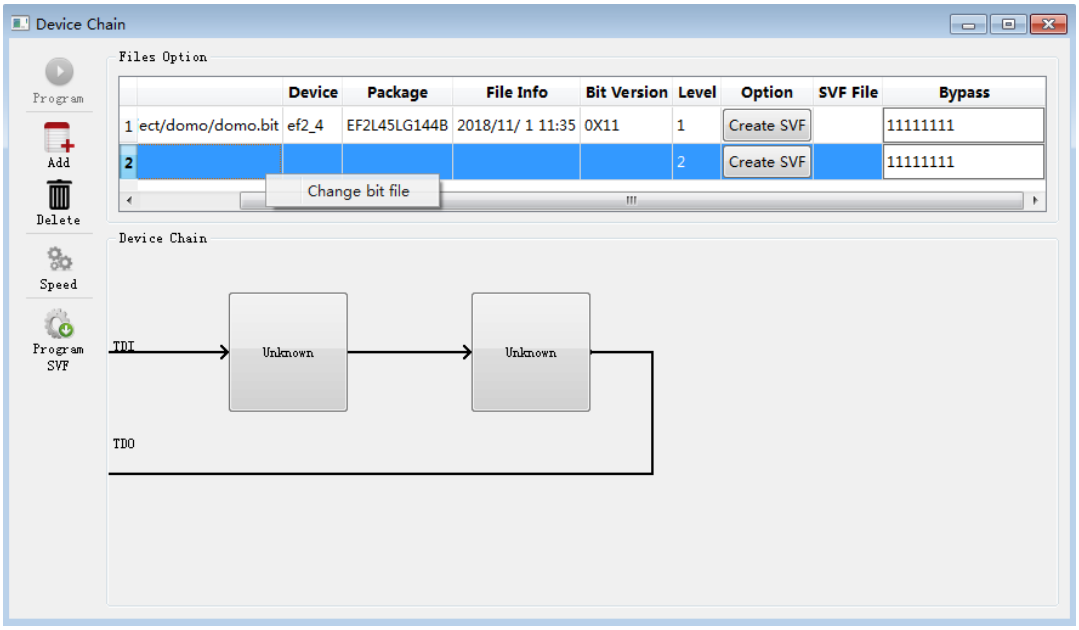
点击界面左边工具栏的“Add”图标添加 bit 文件或 bin 文件，如果为三级级联，需要添加三个文件，默认第一个添加的文件为 Level1。相应的，每添加一个文件会增加一个 Device 级联。

同样，可以先在 Device Chain 区域，右键，选择 “Add Device”进行添加。第一个添加的 Device 为 Level1。



如果需要更改某级的文件，先选中文件列表中该级对应的行，然后选择“Change

bit file”进行文件更改操作。

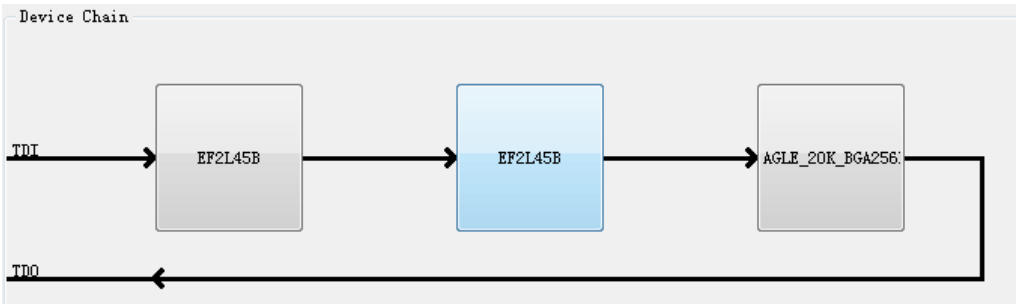


若要删除某一级，可选择该级的文件点击左侧菜单栏的 “Delete” 按钮，也可选中该级对应的 Device，右键选择 “Delete Device”。

在文件列表最后一列为每级芯片对应的 Bypass 值,目前都给出默认值为 11111111，不同厂家的 Bypass 的值可能不一样。

3. 芯片 ID 识别

添加完文件或 Device 后，所有器件均显示为 Unkown，需要手动识别已连接的器件。如果 Device 已正确连接，只需点击 Unkown 图标，则会在相应按钮上显示出具体器件类型，如下图所示，级联了两个 EF2 的器件和一个 Eagle_20 的器件。

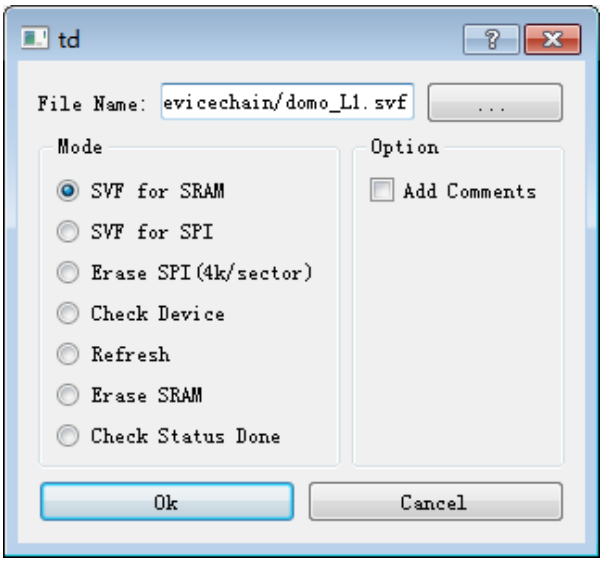


4. 生成 SVF 文件

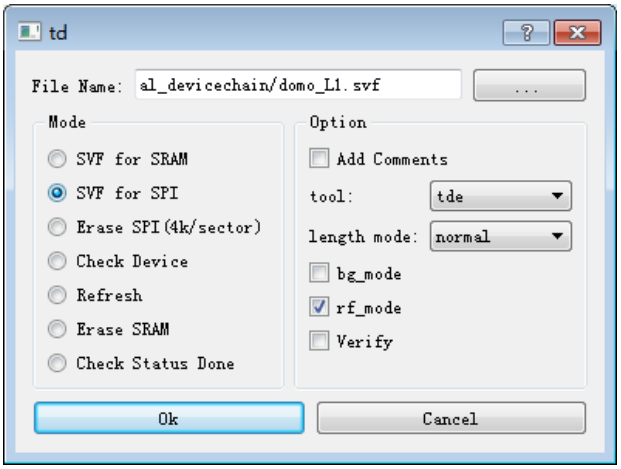
点击文件列表中的“Create SVF”按钮，在弹出的对话框中进行参数设置生成相应层级的 SVF 文件。生成 SVF 文件时共有 7 种模式：

1) **SVF for SRAM**：默认模式，可以支持 n 级 SRAM 的下载；

Add Comments 选项为在 SVF 文件中添加注释信息。



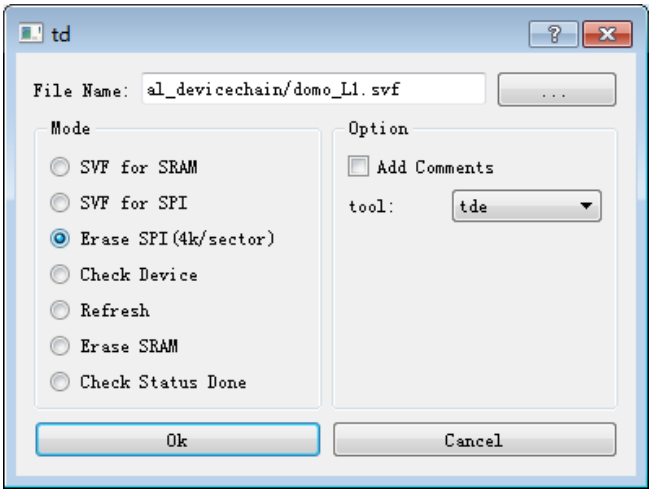
2) **SVF for SPI**：支持 1、2、3 级写入，n 级回读校验（verify）；



- tool：可选 tde 和 jtag_studio，默认为 tde。当选择为 jtag_studio 时，length_mode，bg_mode 和 rf_mode 不可选，均为默认选项。
- length mode：可选为 normal 和 short，默认为 normal；
- bg_mode：后台模式，默认不勾选；

- rf_mode：默认勾选，program spi 后自动刷新；
- Verify：SVF 的校验功能。Bit 位流烧录结束后，把目标区域信息读取出来和文件进行对比，判断是否有 bit 写入错误。

3) **Erase SPI** (4K/sector)：SPI 擦除，tool 可选为 tde 和 jtag studio，默认为 tde；



4) **Check Device**：检查 bit 文件对应的 device id 和 器件 id 是否一致；

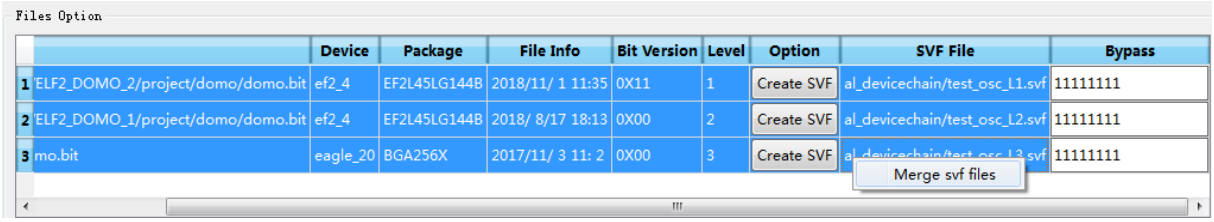
5) **Refresh**：刷新芯片状态；

6) **Erase SRAM**：擦除 SRAM；

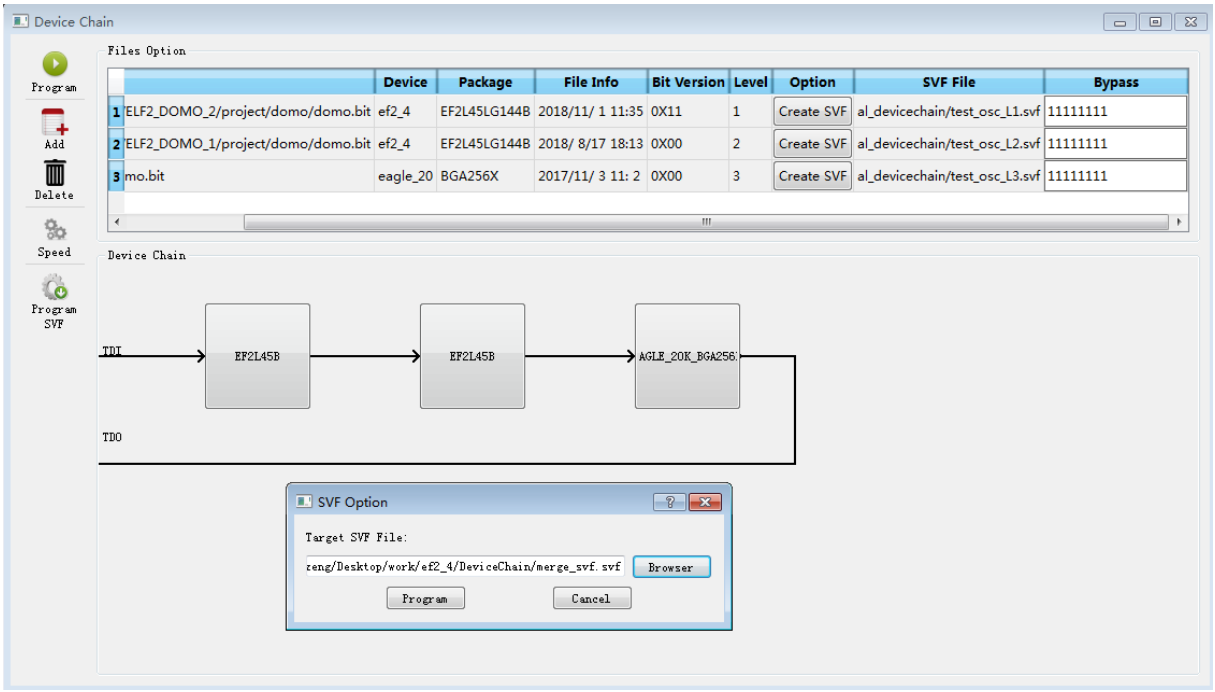
7) **Check Status Done**：通过 JTAG 读出 status 寄存器中 done 的状态，检查 program spi 是否成功。

5. SVF 合并

为了避免级联时，用户多次进行下载操作，增加 SVF 文件合并功能，将所生成的 SVF 文件合并成一个文件，供一次性下载。如图所示，选中要合并的 SVF 所在的那几行，右键选择“Merge SVF files”。



合并后的 SVF 文件，使用 Program SVF 进行下载。



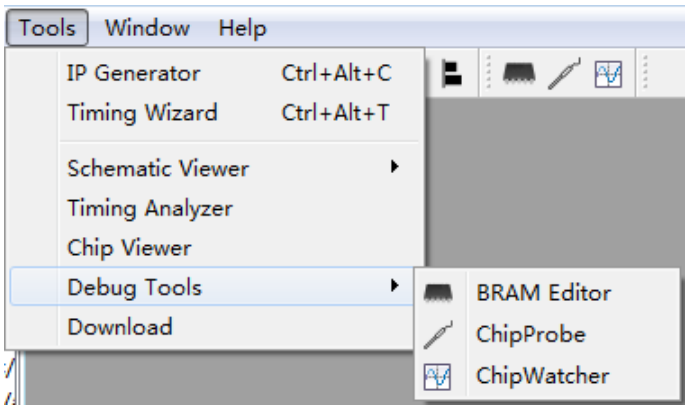
6. Program

选中要下载的 SVF 那一行，点击左边工具栏的 Program 按钮进行下载。SVF for SRAM、SVF for SPI 和 Check Device 模式都会检查 bit 文件对应的 device id 和器件 id 的一致性，如果不一致则会 TD Console 窗口中给出 error。

下载之前，还可点击工具栏的 Speed 按钮对下载速度进行调节，默认值为 6（1Mbps）。

8 工具集

TD 软件中有许多工具帮助用户更好的分析工程，主要有：**Schematic Viewer**、**ChipViewer**，以及调试工具集中的三个工具：**BramEditor**、**ChipProbe**、**ChipWatcher**。



Schematic Viewer 为综合优化的每一中间过程生成相应的逻辑电路图，提高在设计过程中的交互性，帮助设计人员理解电路，缩短电路设计的开发周期。**Chip Viewer** 可提供物理实现后的详细信息，包括资源使用率，详细布局，全局布线以及关键时序路径。

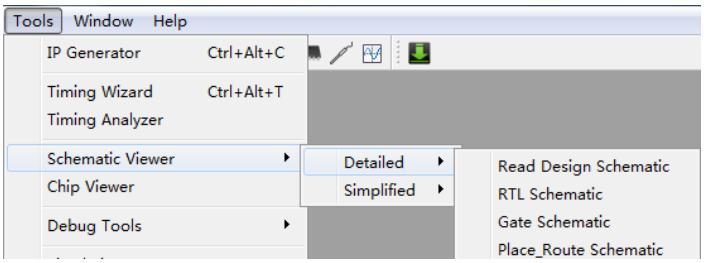
在不改变设计的情况下，**ChipWatcher** 无需借助外部设备，即可查看内部信号在指定条件下的变化情况，是一款在线调试工具。而 **ChipProbe** 则通过把内部需要监控的信号指定到闲置的 IO 端口，通过示波器等外部工具进行查看。**Bram Editor** 从芯片中的 RAM 读取数据，并可对这些数据进行修改，修改后写进芯片，即可看到改动效果。

8.1 Schematic Viewer

在 TD 软件中完成每一个过程后，都可在 **Schematic Viewer** 中查看相应的逻辑电路图。在“**Read Design**”、“**Optimize RTL**”完成后，查看到的逻辑电路图是对语法分析，逻辑优化后的效果，此时的电路结构与工程器件是独立的，在电路图中看到的是加法器、乘法器、比较器、与门、或门等元器件。而在“**Optimize Gate**”、“**Optimize Placement**”、

“**Optimize Routing**”完成后，查看到的逻辑电路是基于工程所选 FPGA 器件结构优化后的效果，在电路图中看到的是查找表、寄存器、BRAM、PLL 等元器件。

展开 **Tools → Schematic Viewer**，可看到有两种模式可以选择：**Detailed** 和 **Simplified**。



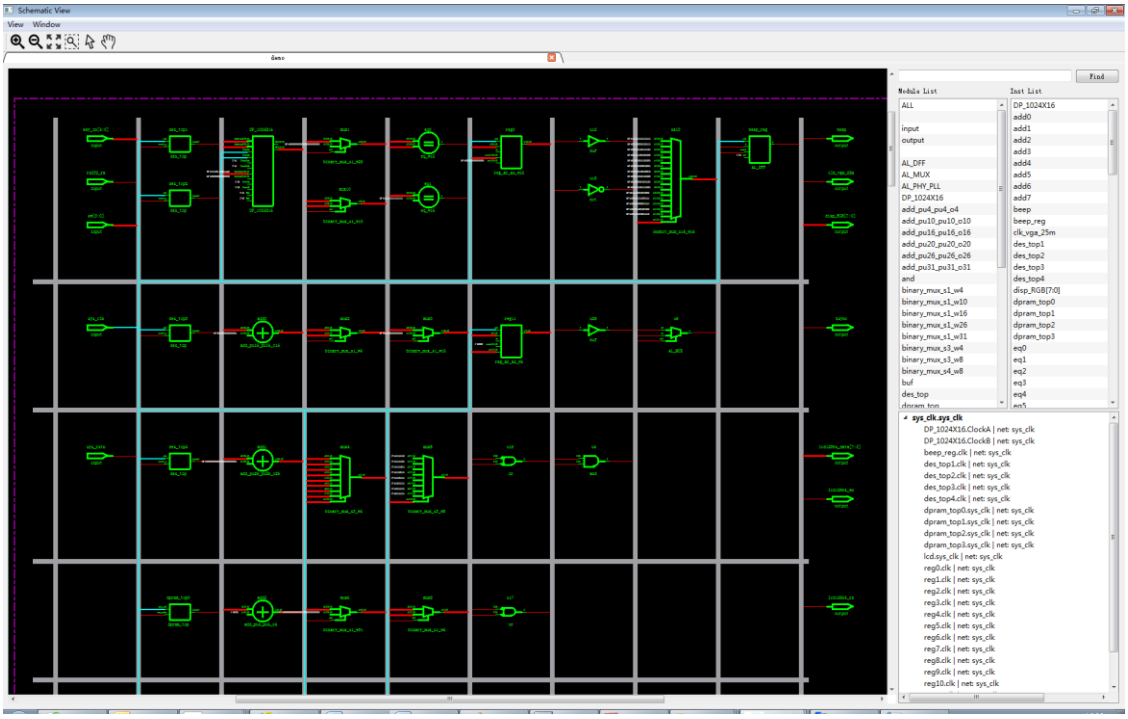
当 HDL2Bit Flow 运行至 Read Design 这一步时，可执行 Read Design Schematic；

当 HDL2Bit Flow 运行至 Optimize RTL 这一步时，可执行 RTL Schematic；

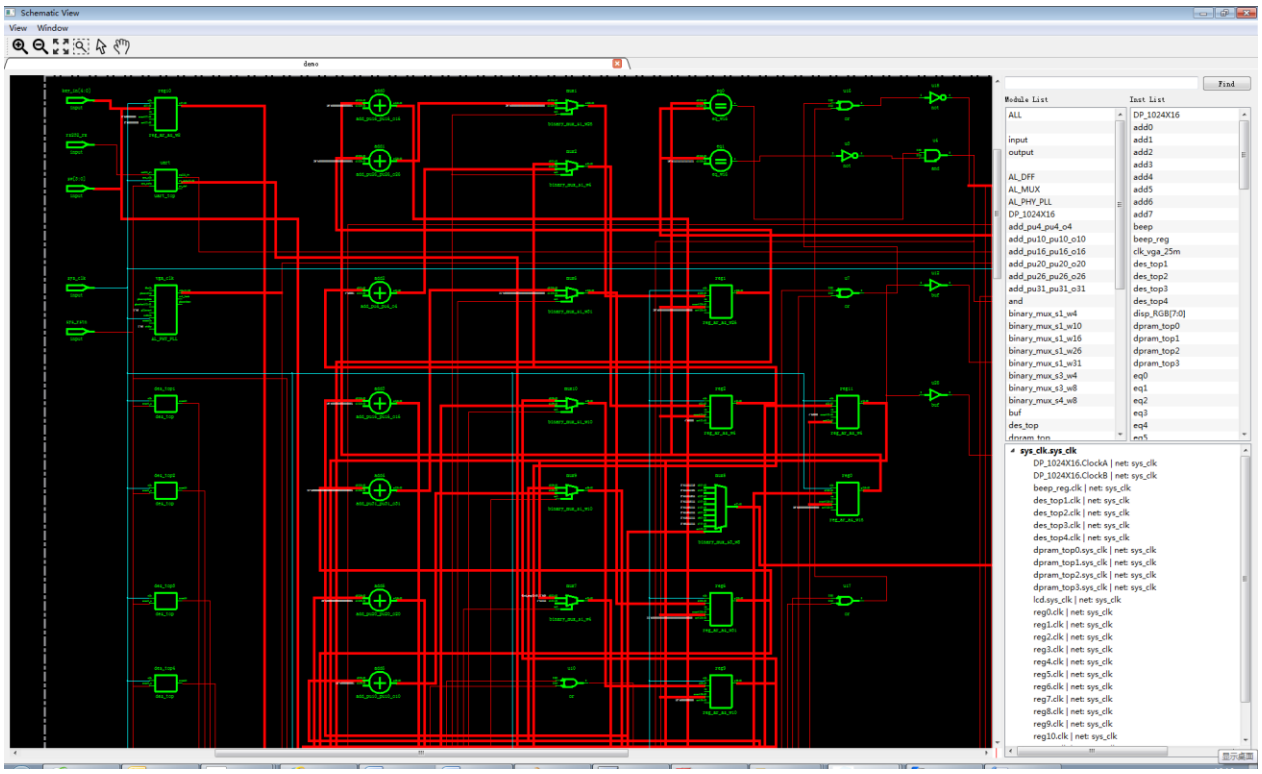
当 HDL2Bit Flow 运行至 Optimize Gate 这一步时，可执行 Gate Schematic；

当 HDL2Bit Flow 运行至 Optimize Routing 这一步时，可执行 Place_Route Schematic。

在 Simplified 模式中，显示的电路图会更精简，没有复杂的电路连线，所有的线路均经由总线传输，此模式适合资源利用较多的工程，如下图所示。



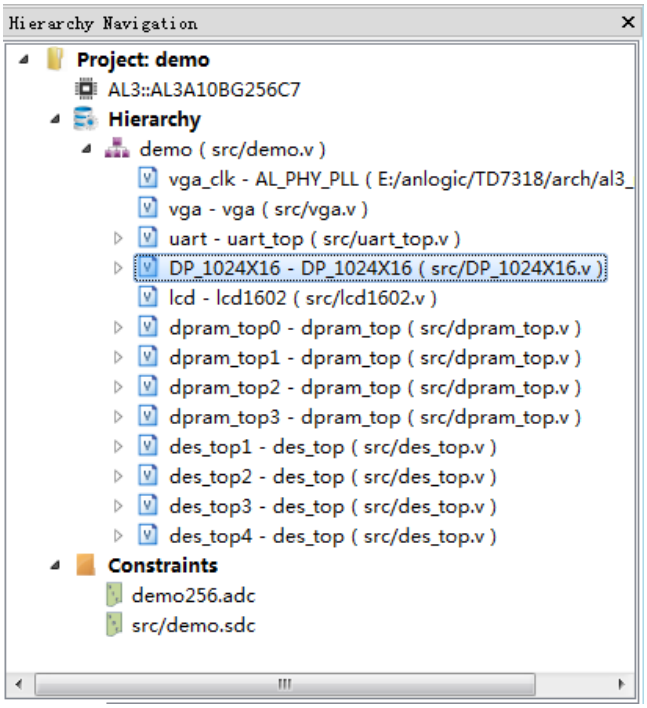
在 Detailed 模式中，可以看到经过 TD 综合优化后详细的电路信息，此模式适合资源利用较少的电路，可更好的理解电路设计，如下图所示。



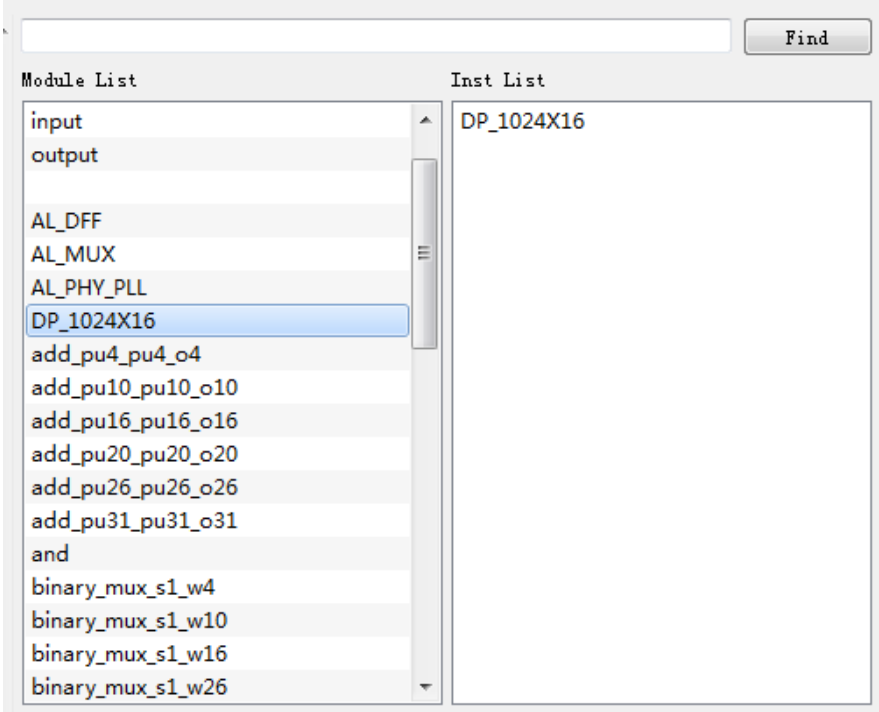
图形界面中的操作方式：

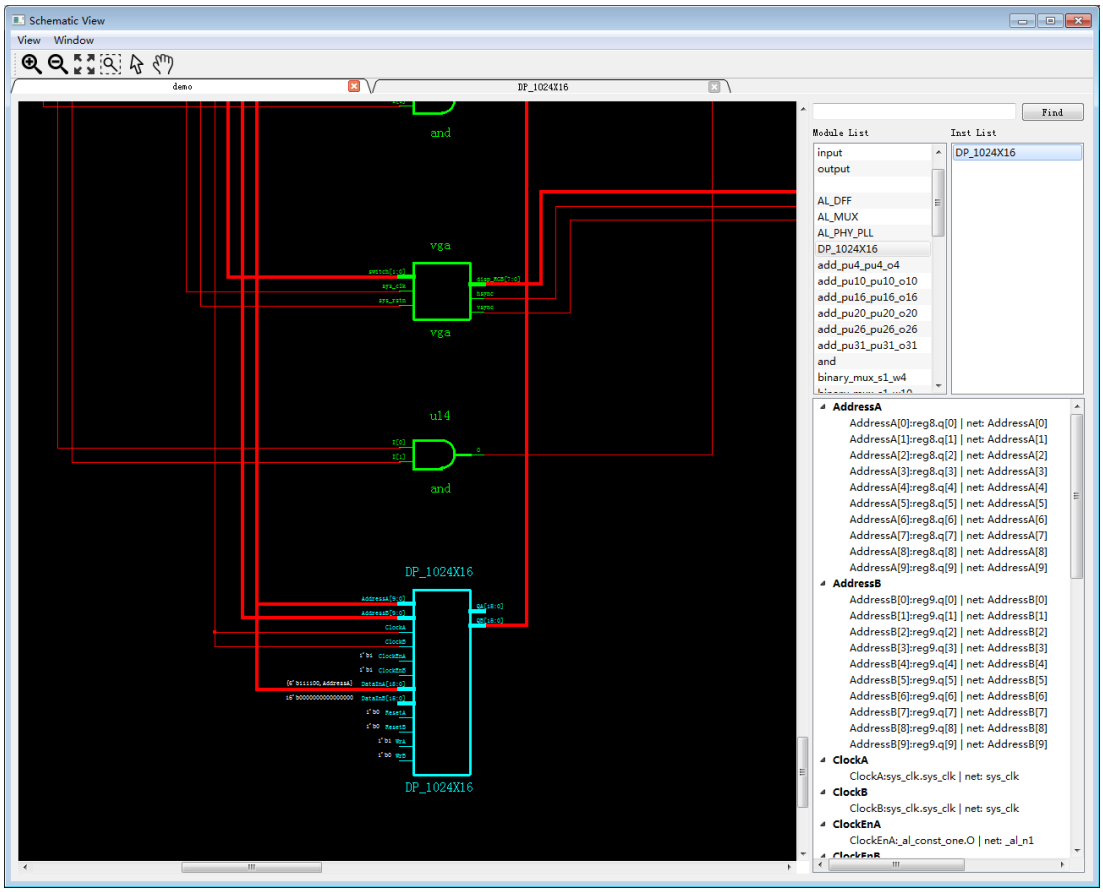
1. 高亮显示：鼠标点击连线、电路元件或者通过双击 Inst、Net；
2. 缩放：点击工具栏中的按钮进行放大，点击按钮进行缩小，或者通过 Ctrl+滚轮的方式进行缩放；
3. 整体视图：点击工具栏中的按钮，可在主视窗中显示整个电路图；
4. 拖动：点击工具栏中的按钮，可对主视窗进行上下左右的拖动；
5. 局部放大/缩小：点击工具栏中的按钮，在主视窗中按下鼠标左键，向右下拖动可对局部进行放大，向左上拖动可对局部进行缩小；
6. 选择模式：点击工具栏中的按钮，可切换回正常的鼠标功能。

在“Read Design”后，还保留了用户设计的层次结构，所以在 **Schematic Viewer** 中可以通过双击查看设计中的子模块，如下图中的子模块 DP_1024x16。

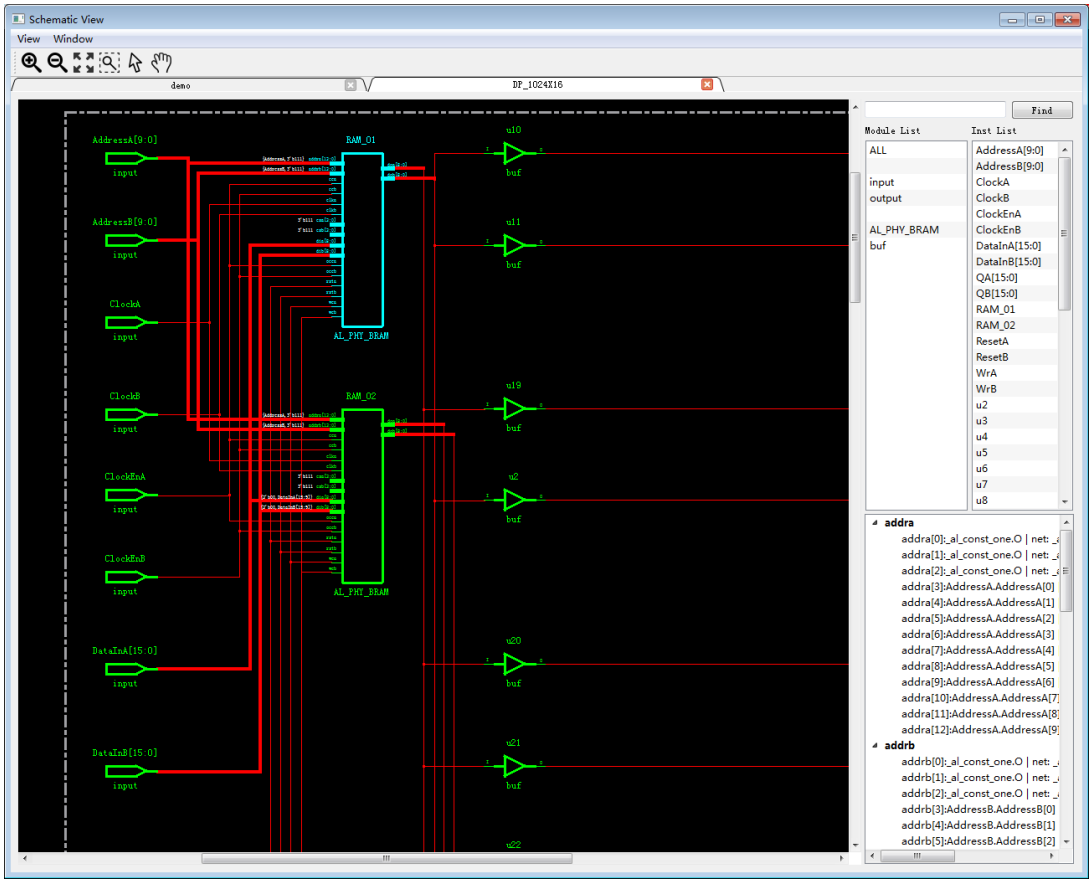


在 **Module List** 中查找 DP_1024x16 并双击，则在 **Inst List** 中会列出该模块对应的 Instance，双击该 Instance 则可在主视窗中跳转到它所在的位置。

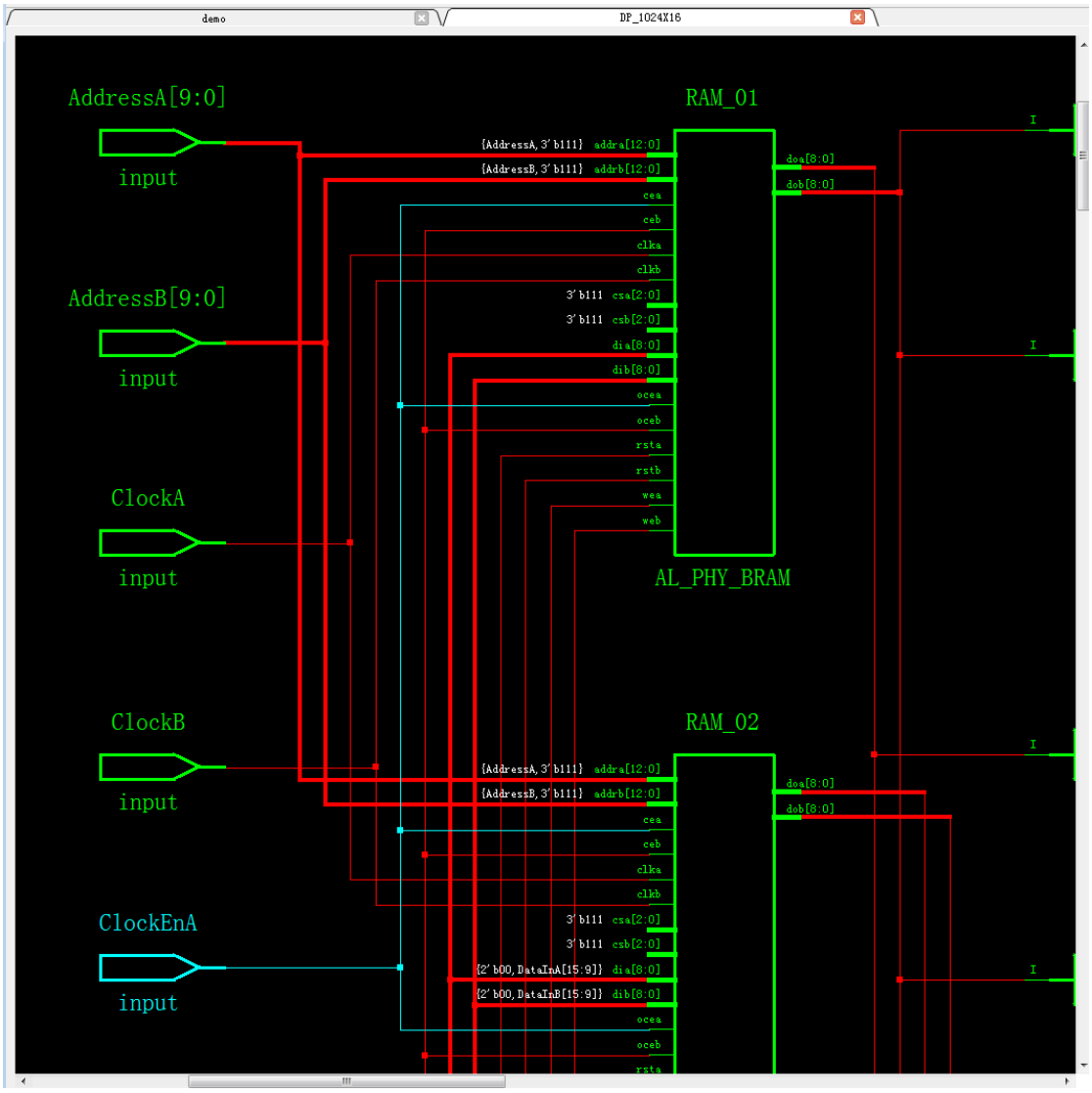




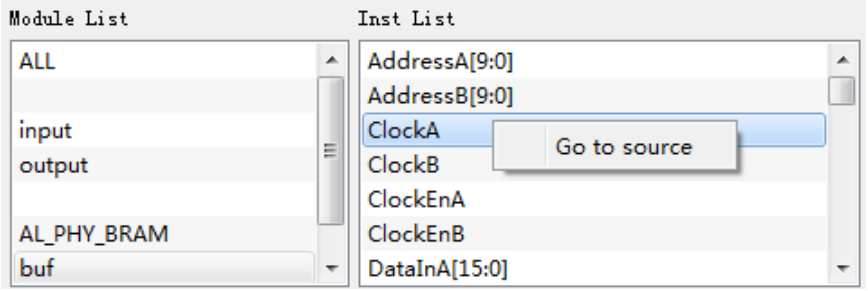
在主视窗中双击 DP_1024x16 图形，则可打开该模块对应的电路图。



在主视图中选中某个模块，会在右下角的窗口显示其所有相关的 net，双击某条 net 可以高亮显示该 net 所有的连接关系。在主视窗中，加粗的连线为 bus，非加粗的为 net，没有连线的 port 为常数，并可查看到该 port 的值。



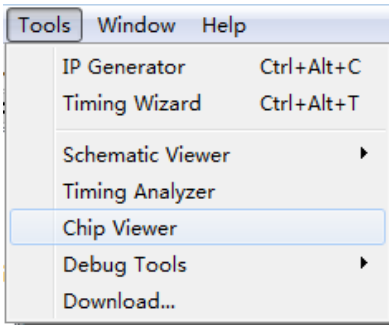
在 Inst List 中，还可右键单击某 Instance，选择“Go to Souce”，则可跳转至该 Instance 所在源代码中的位置。



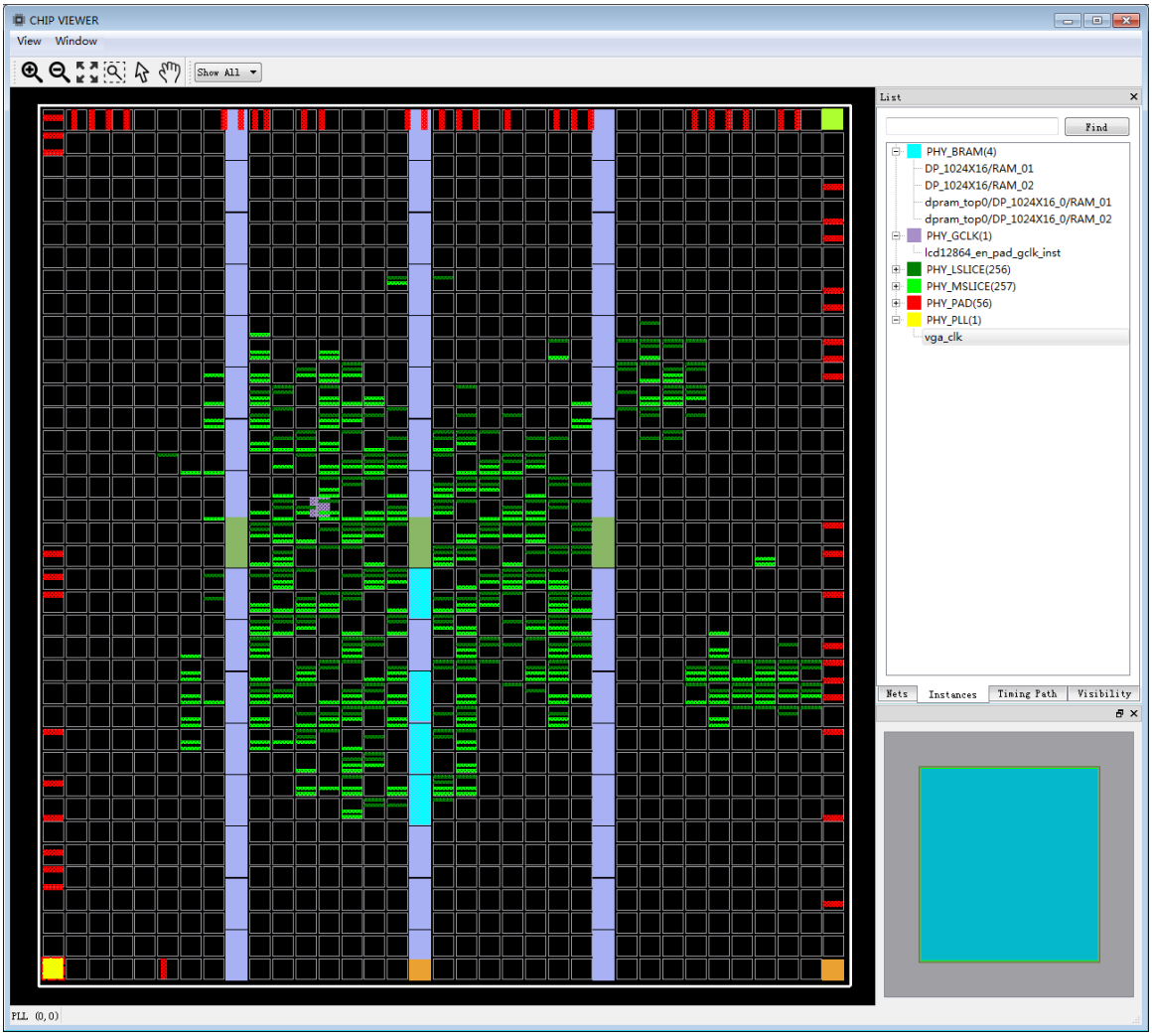
8.2 Chip Viewer

布局布线完成后，用户可通过 **ChipViewer** 来查看物理实现完成后的详细信息：包括 Cells Utilization、Global Routing、Detail Routing、All Nets、Timing Path。

单击 **Tools → Chip Viewer** 打开，

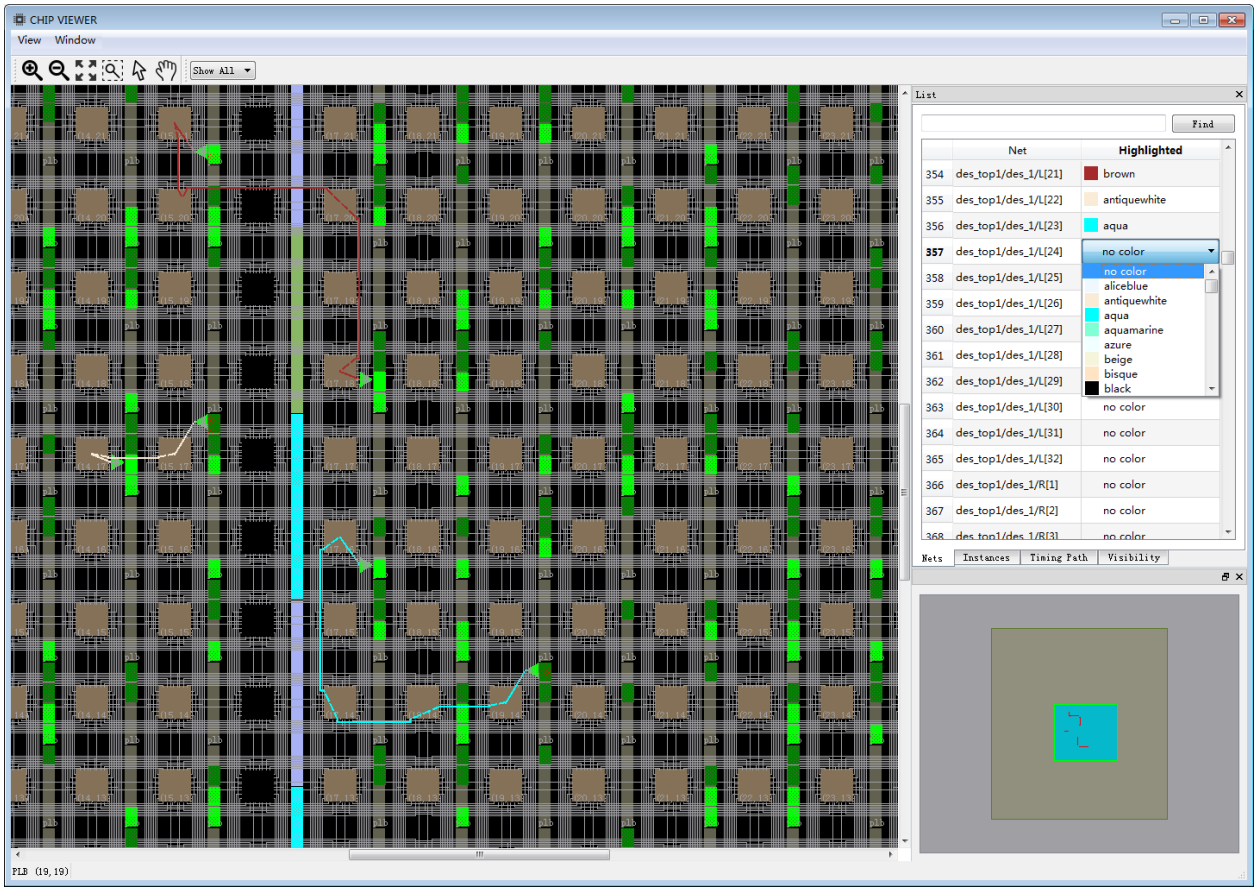


Chip Viewer 的顶层图例如下所示：

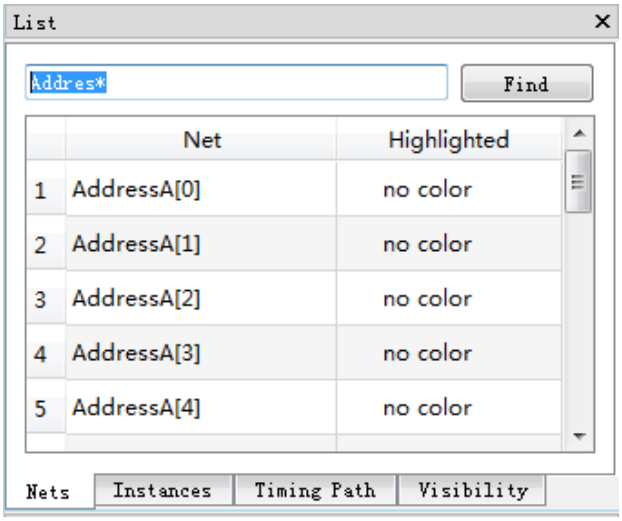


图形界面中的操作方式与 **Schematic Viewer** 相同。

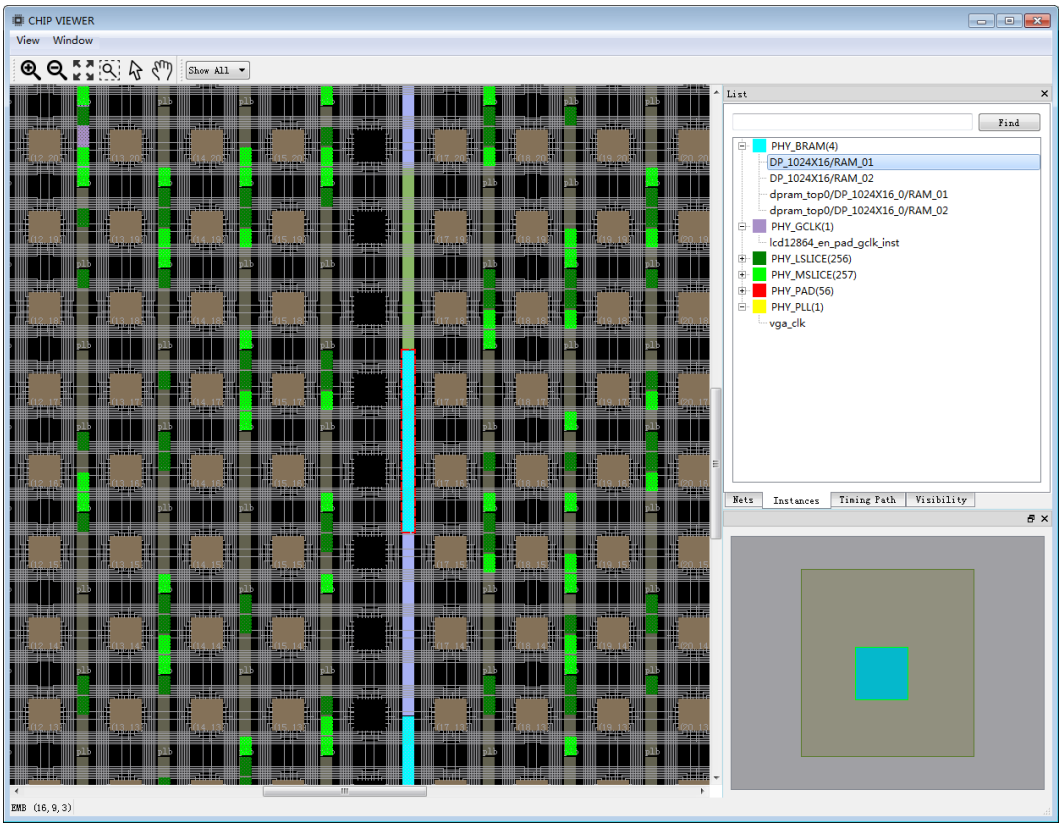
Nets 一栏中可查看到 design 中所有的 net，双击该 net 即可在左侧窗口显示具体的走线情况，在 **Highlighted** 一列中可选择不同的颜色来保持该 net 高亮显示。对于一条 net，三角形箭头由 plb 朝外的为 source 端，三角形箭头朝向 plb 的为 sink 端。

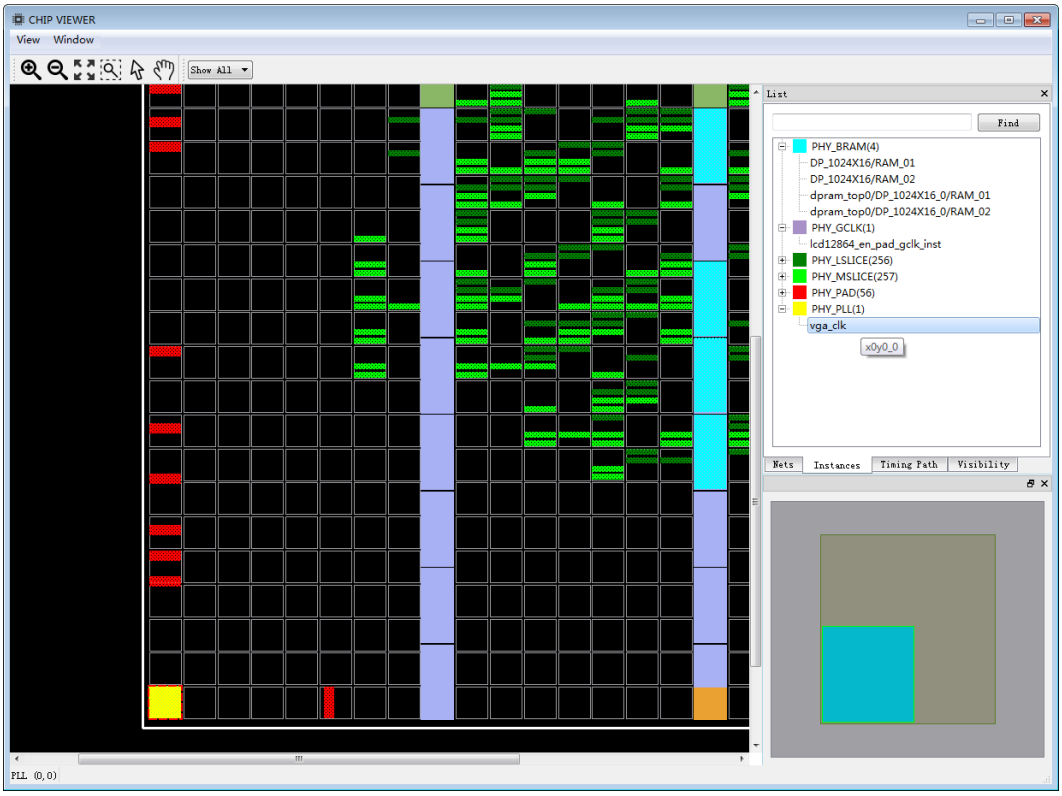


在 Find 方框可对 net 进行搜索。

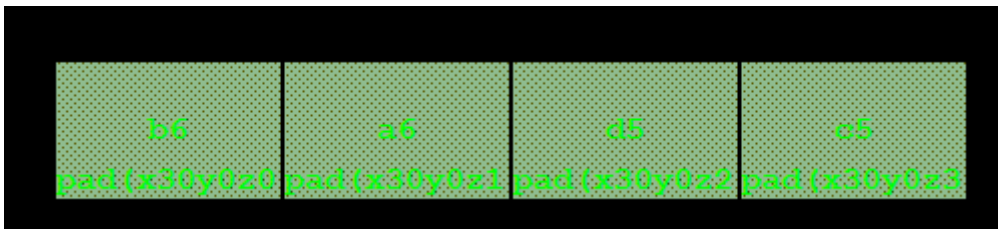


在 **Instance** 一栏可查看 design 中所有物理模块，双击该 instance 可在左侧窗口显示其布局位置。单击该模块，将会在左下角显示其物理坐标。

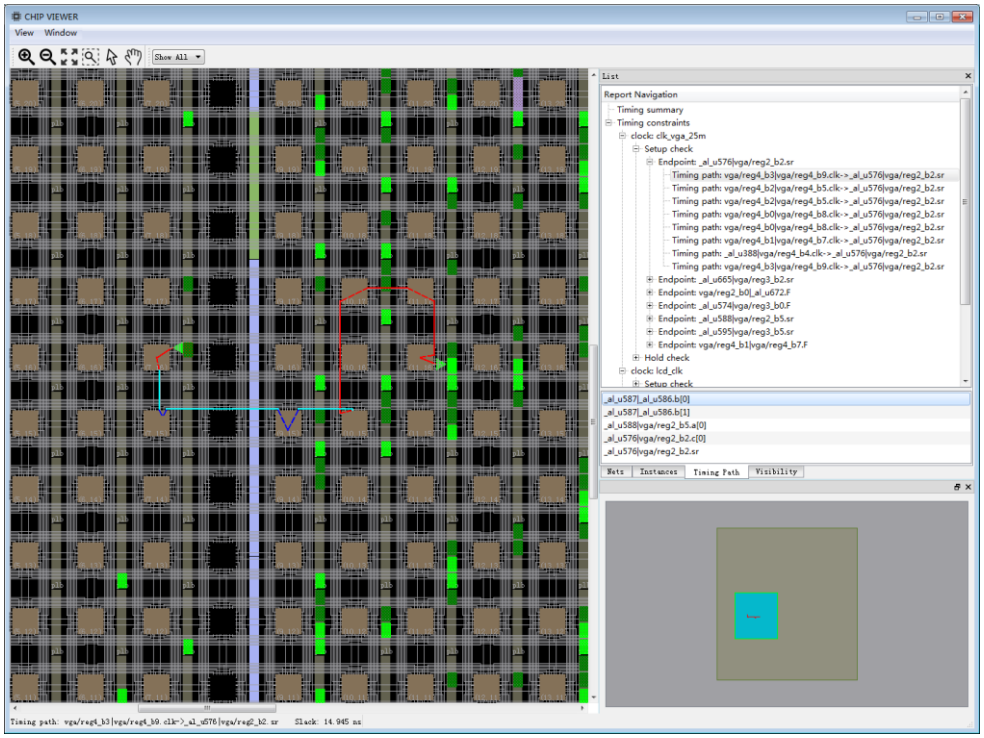




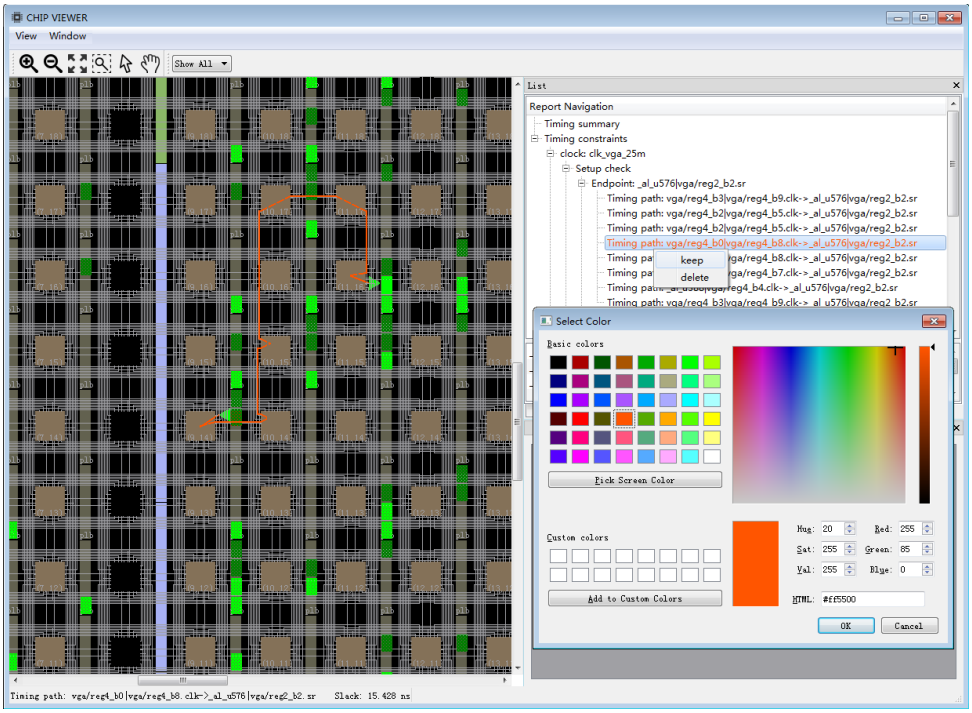
将视图放大，可看到每一个 pad 对应该封装内部的 location，方便用户进行约束，如下图中对应的 location 分别为：B6，A6，D5，C5。



在 **Timing Path** 一栏可查看到 timing report 中列出的所有时序路径，双击该 timing path 可在左侧窗口显示其详细走线情况。双击 path 中的某一条 net，可进行高亮显示。



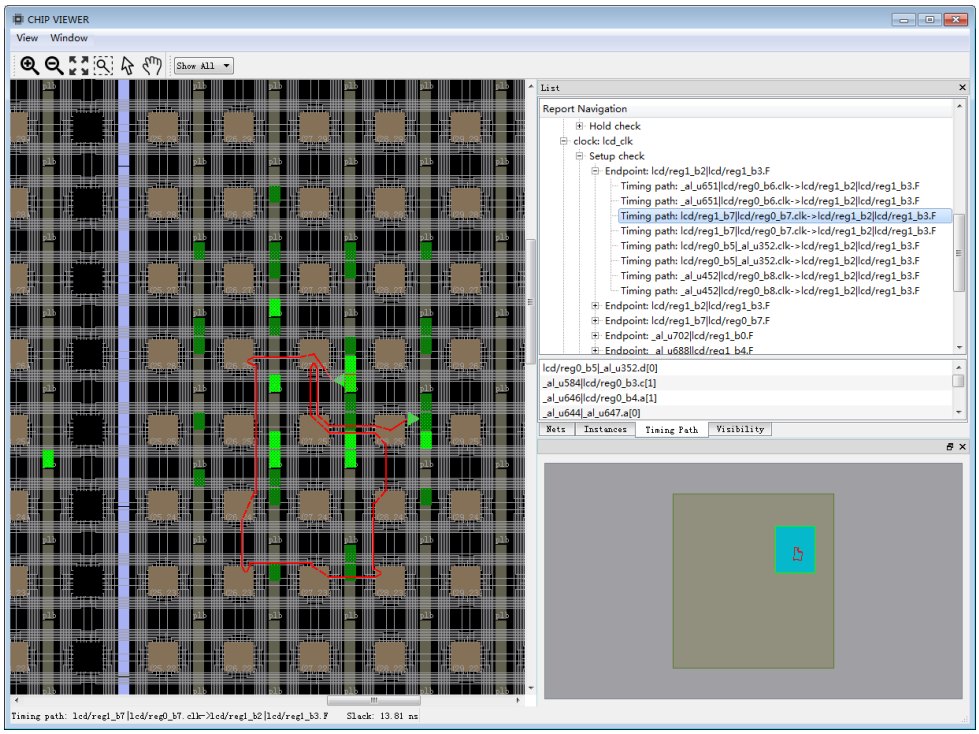
同样，选择一条 timing path，右键单击，选择 keep，并选择一种颜色，对该 path 进行高亮显示，并且保持该 path 的持续显示，即在 Chip Viewer 中可以同时显示多条 timing path 或 net。



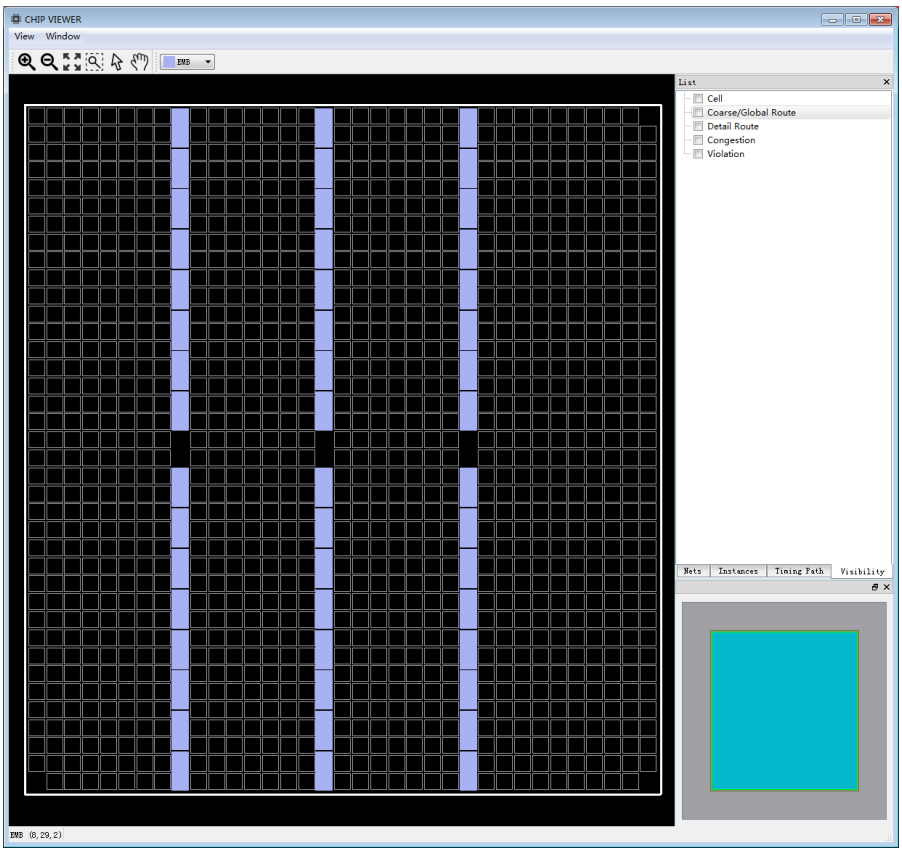
在 Visibility 一栏可选择显示基本单元，全局布线或详细布线。

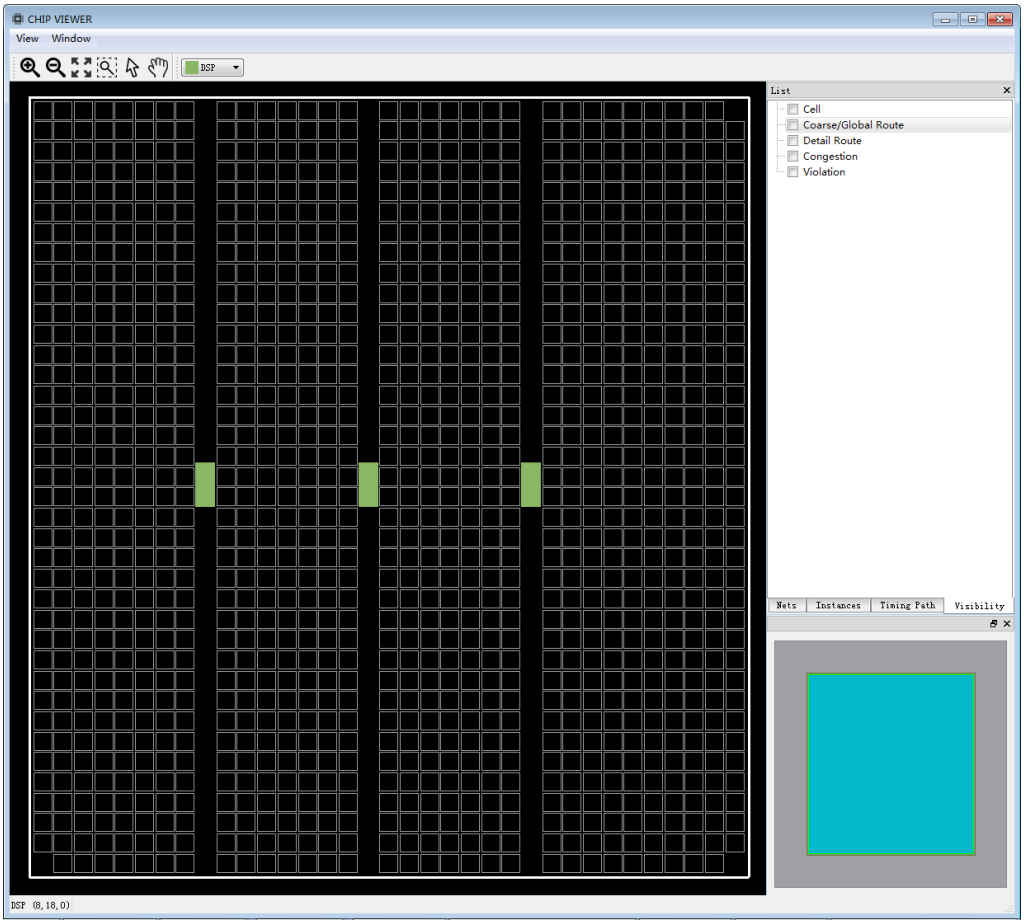


右下角显示的为 Chip Viewer 中的鸟瞰图。

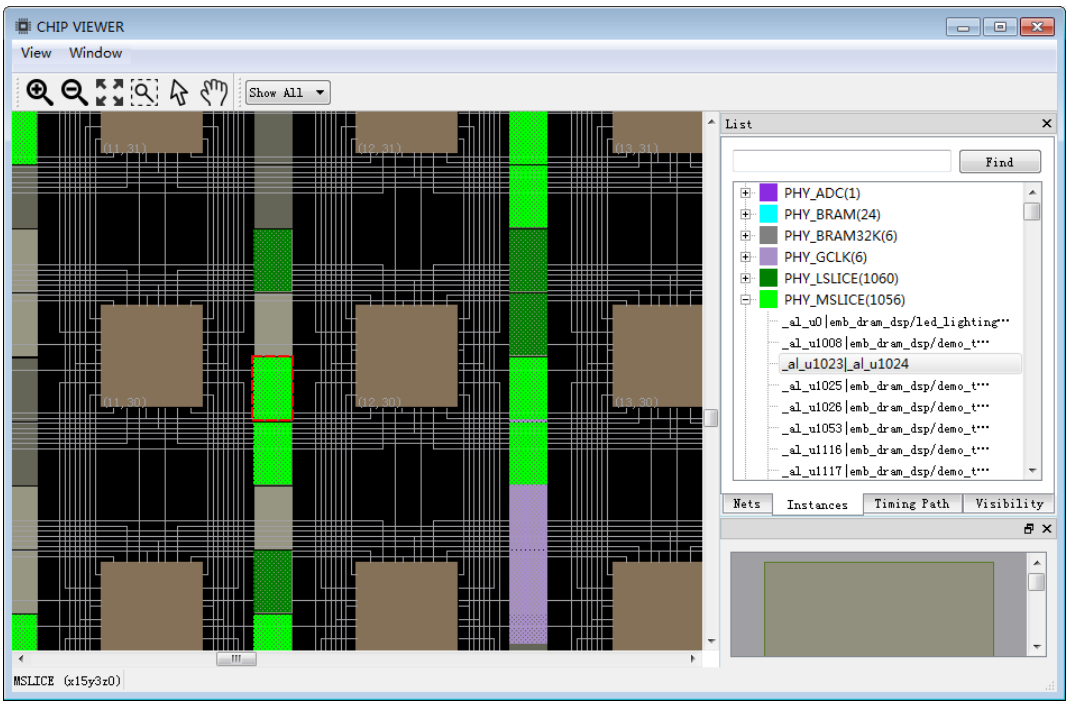


在 Chip Viewer 的菜单栏中，可以选择显示该 device 中所有物理资源的位置，如：PLL，EMB，EMB32K，DSP。单击某个模块，可在左下角显示其物理坐标。



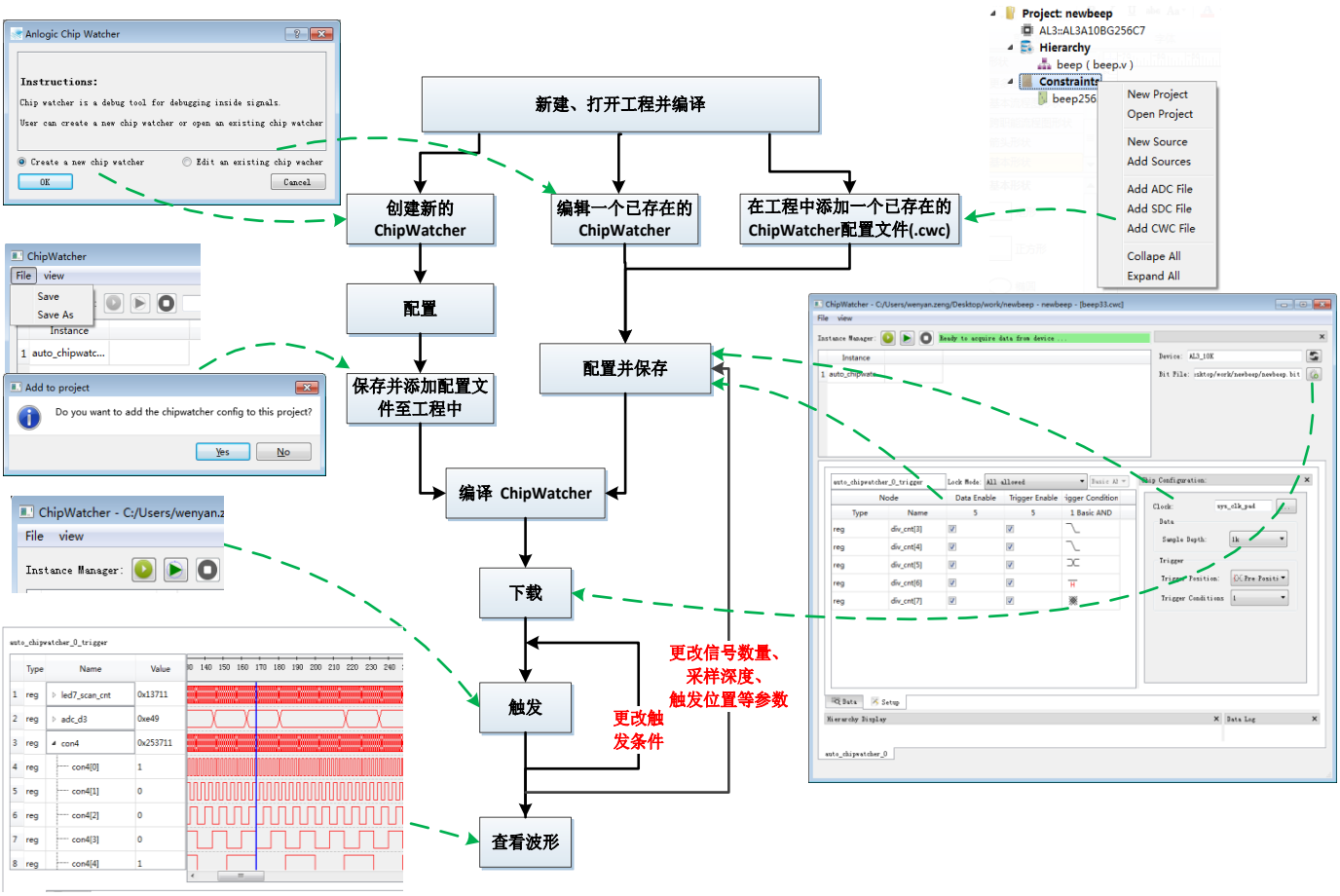


一个 plb 中包含 4 个 slice，mslice 的 z 坐标为 0,1；lslice 的 z 坐标为 2,3。如下图所示的 mslice 的坐标为（x15y3z0）。



8.3 ChipWatcher

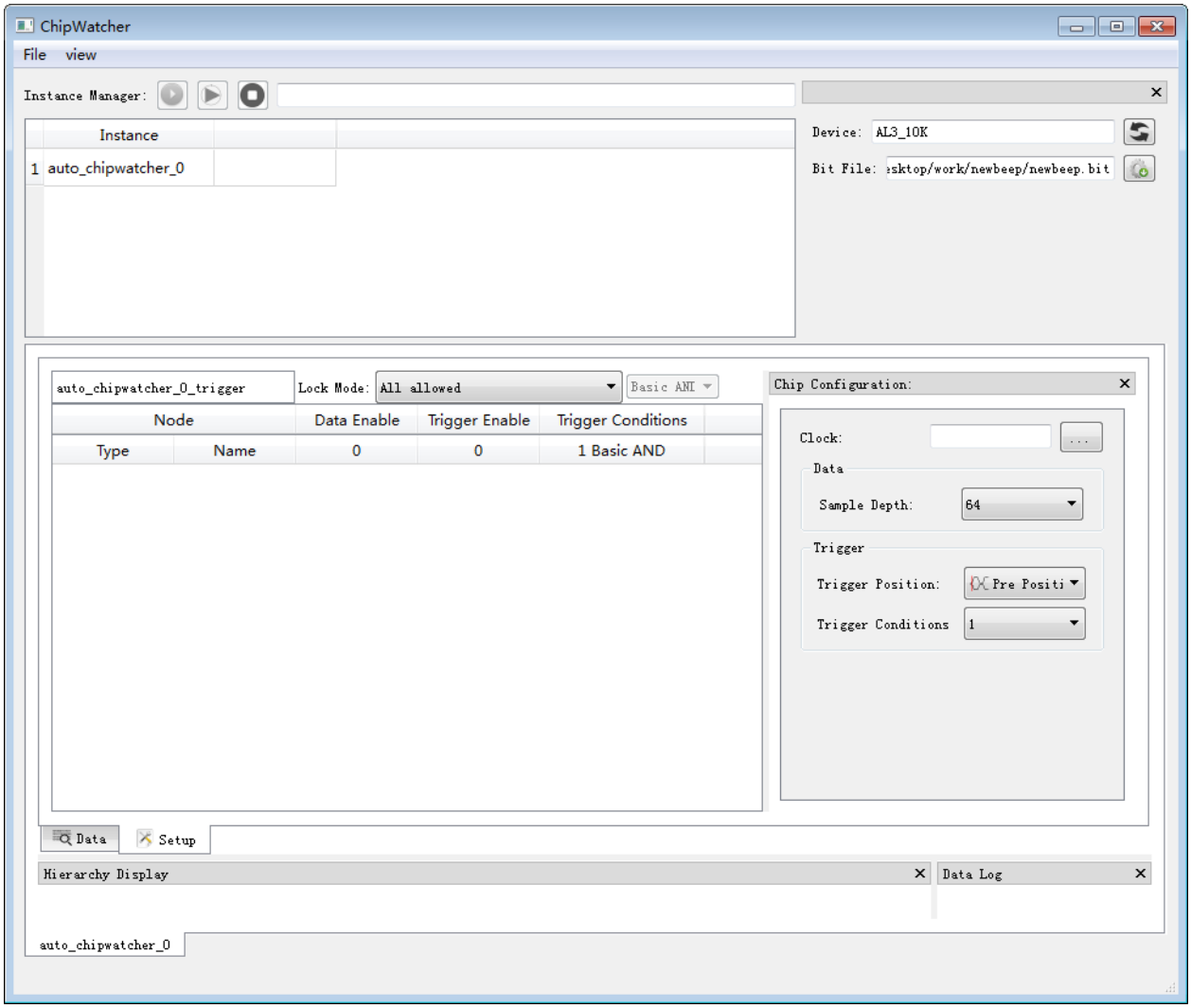
通过 **ChipWatcher**，用户无需借助外部设备即可在线监测电路内部信号的变化情况。在 **ChipWatcher** 中，用户可同时添加多个信号，在设置信号的采样时钟、采样深度、触发条件及触发位置后，经过重新编译、下载和触发，即可查看到指定条件下的信号变化情况。**ChipWatcher** 的工作流程如下图所示。



1. 运行完 HDL2Bit 流程后，有三种方式启动 ChipWatcher：

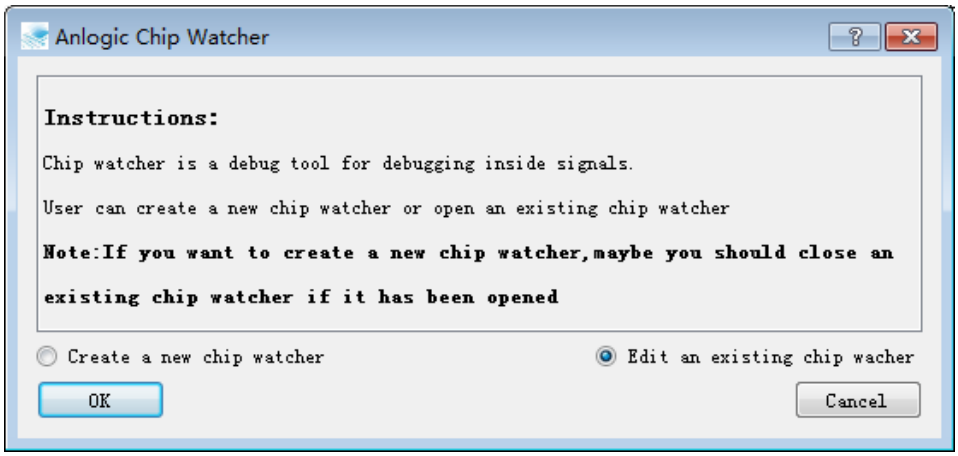
   ①．创建新的 ChipWatcher

   展开 **Tools → Debug Tools** ，选择 **ChipWatcher**，根据提示选择“**Create a new chip watcher**”，点击“**OK**”，出现如下 ChipWatcher 的主界面：



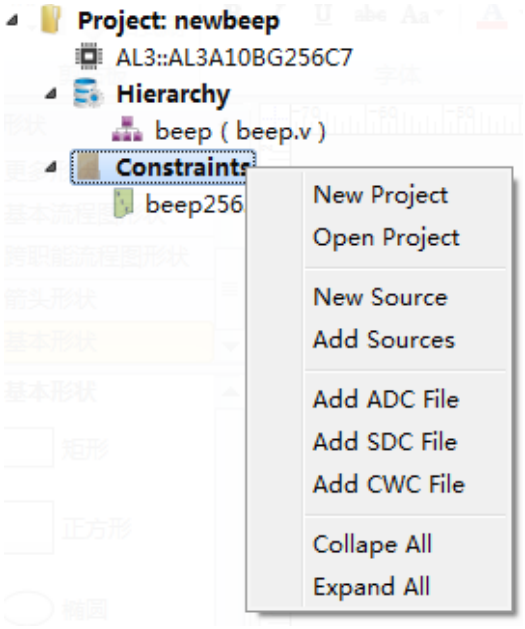
②．编辑一个已存在的 ChipWatcher

展开 **Tools → Debug Tools** ，选择 **ChipWatcher** ，根据提示选择 “**Edit an existing chip watcher**”， 点击“**OK**”，进入 ChipWatcher 的主界面。

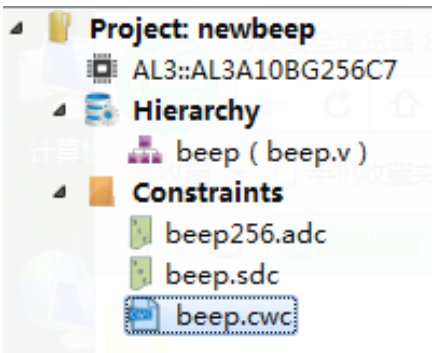


③．在工程中添加一个已存在的 ChipWatcher 配置文件(.cwc)，并双击打开。

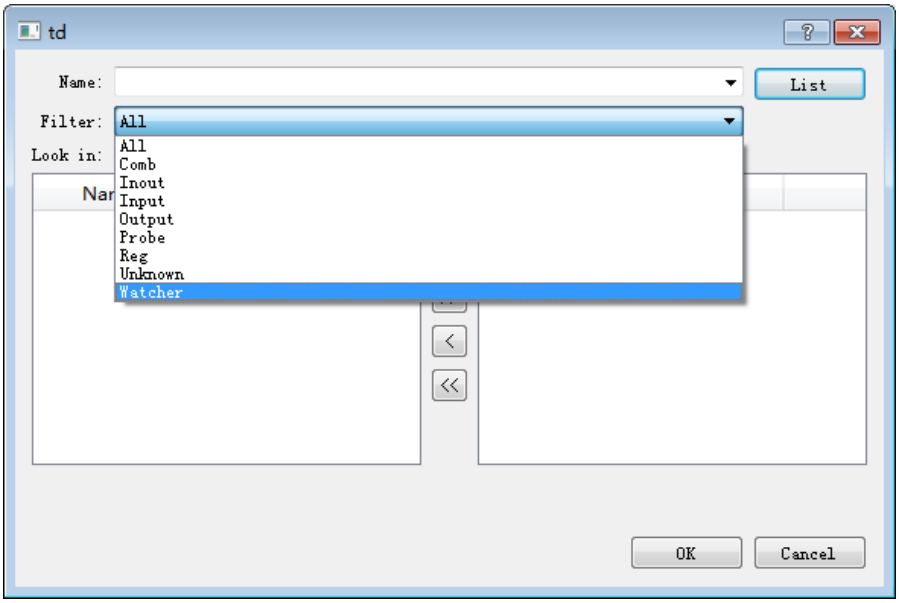
在 **Project** 栏目中，右键单击 **Constraints**，选择“**Add CWC File**”，为工程添加已生成的配置文件(.cwc)。



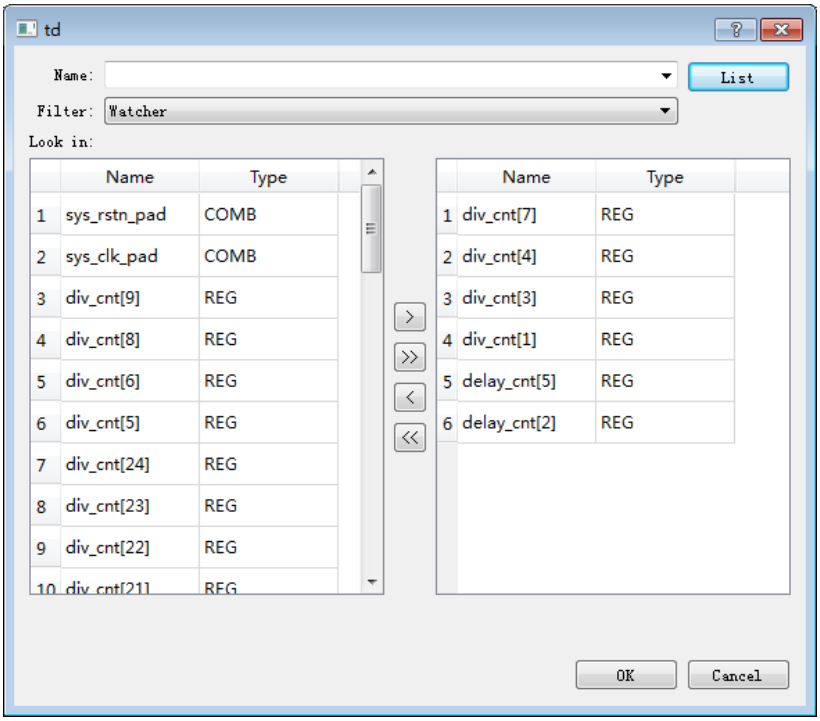
双击该文件即可打开 ChipWatcher 的主界面。

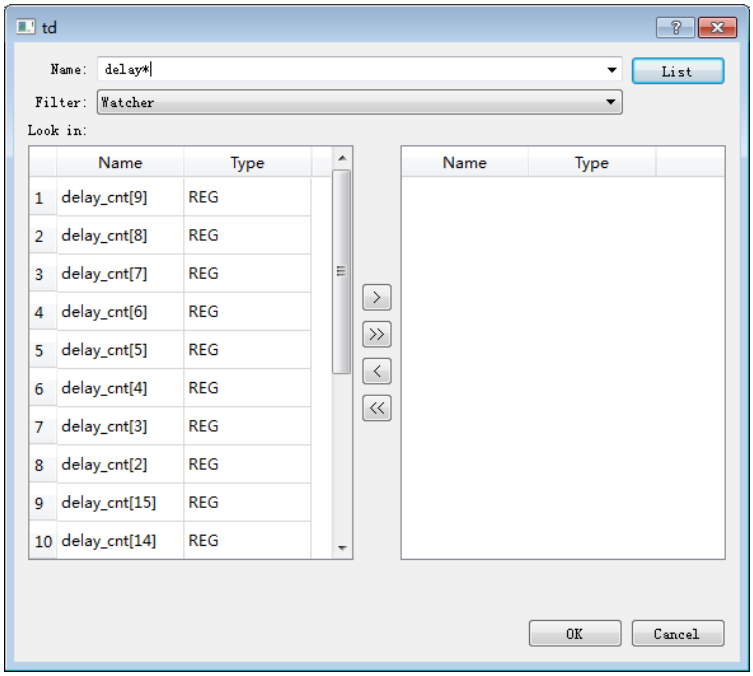


2. 在 **setup** 界面的空白处右键单击，选择“**Add Node…**”，出现如下 Node Filter 界面，展开 **Filter** 下拉菜单，默认类型为 **Watcher**。若用户选择 filter 类型为 Watcher 之外的信号将不能保证被正确采样;

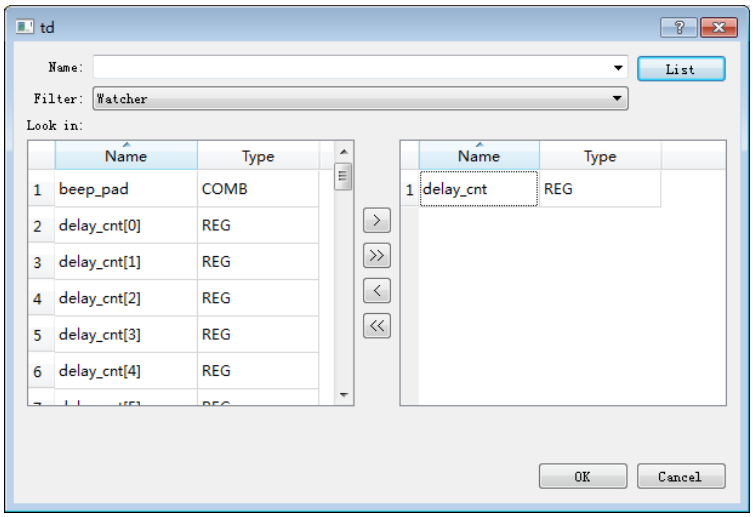


3. 通过向左向右箭头选择或删除信号，也可在 **Name** 一栏中搜索信号（支持通配符*）；



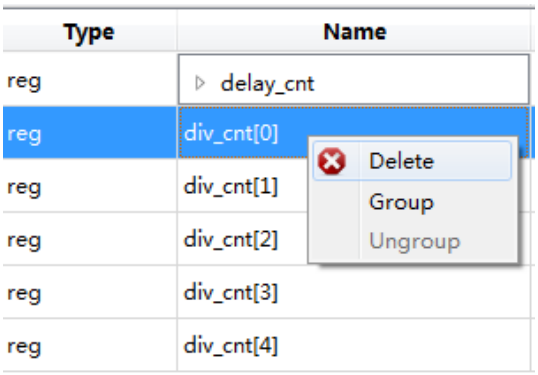


可根据需要，添加 net 或者 bus 信号，点击 OK 按钮，完成信号的添加

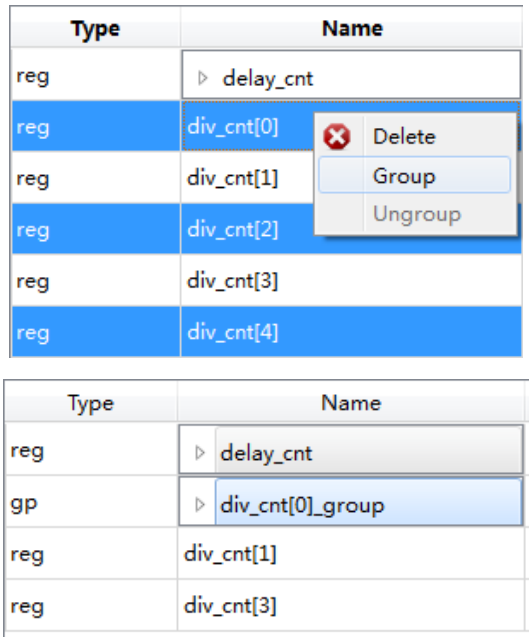


4. 对于已添加的 net 或者 bus,右键单击信号名,可进行 **Delete/group/ungroup** 操作;

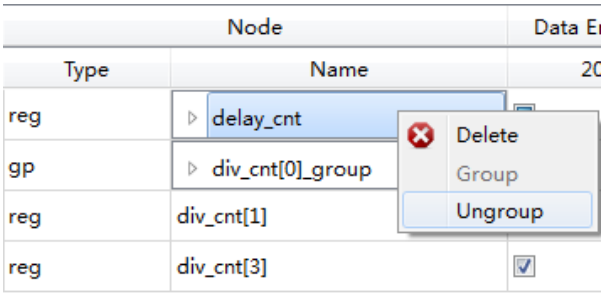
   单个信号或整组 bus 都可直接进行删除操作;

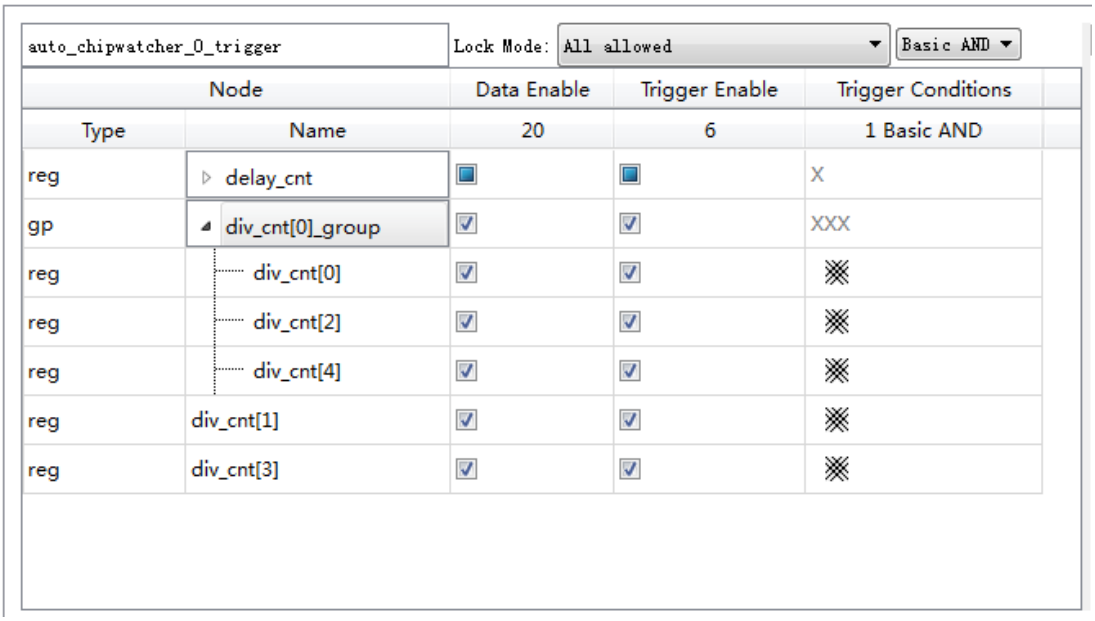


同时，为了便于信号的观察，可对多个线网信号进行 **Group** 操作；

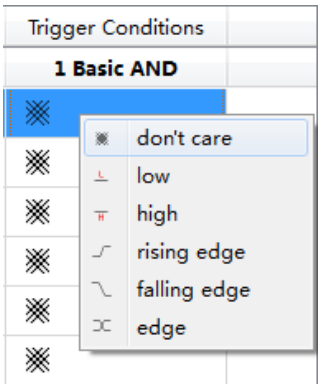


对于 bus 或 group 可以进行 **Upgroup** 操作。同时，鼠标选中 net、bus 或 group 可以对其进行拖拽，从而调整当前 Node 的排列顺序。

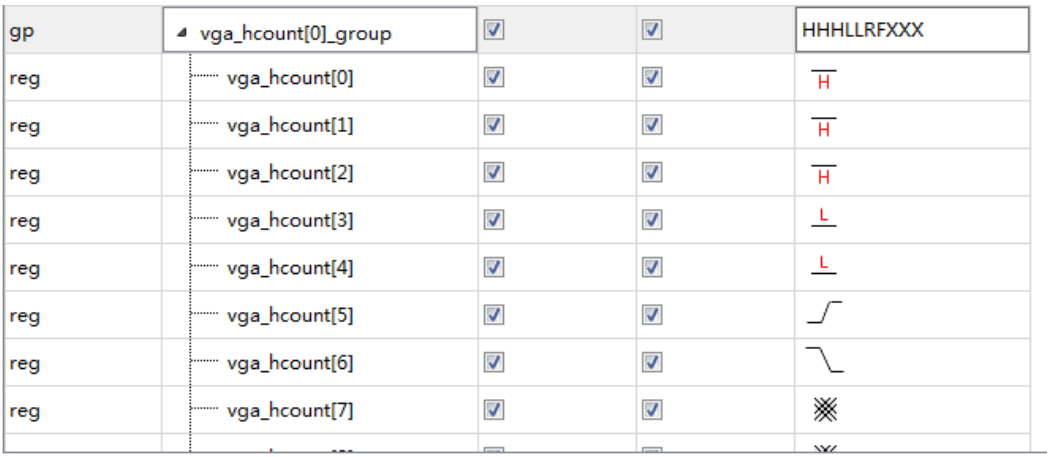




5. **Data Enable** 是指选择需要采集并显示波形的信号，在复选框中打勾表示使能该信号；**Trigger Enable** 是指将该信号的某一状态作为触发条件；**Trigger Conditions** 是指需满足该条件时才能对信号进行触发；**Basic AND** 是指需同时满足以下所有触发条件时才能对信号进行触发；**Basic OR** 是指只要满足以下任一触发条件即可对信号进行触发。右键单击触发条件一栏，可更改触发条件，如下所示，触发条件依次为: 任意位置、低电平、高电平、上升沿、下降沿和双沿（上升沿或下降沿）；

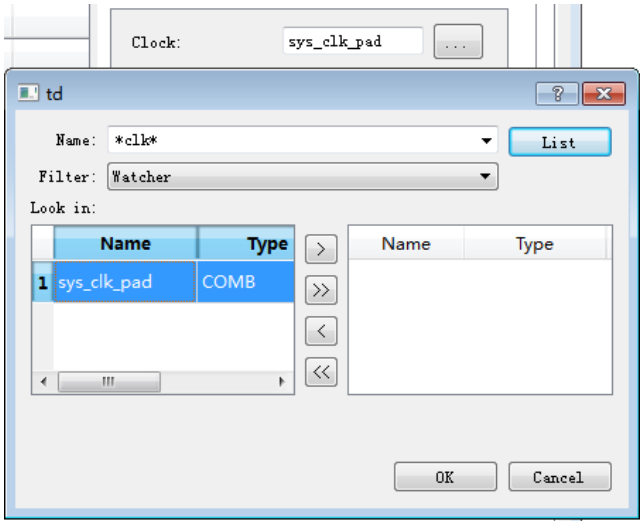


同时，可以双击 bus 或者 group 的 **trigger conditions** 进行编辑，可输入的字符为: x/X、l/L、h/H、r/R、f/F、e/E。不填的 net 以 X 填充，输入后可查看到相应 net 的 trigger condition 发生了变化。

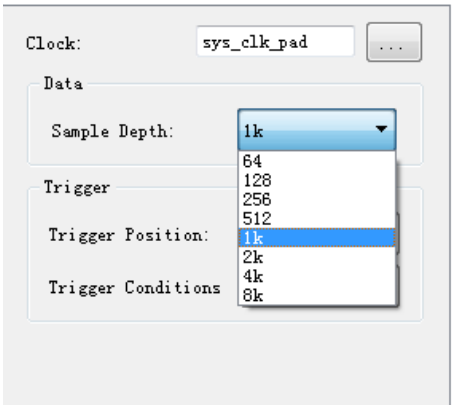


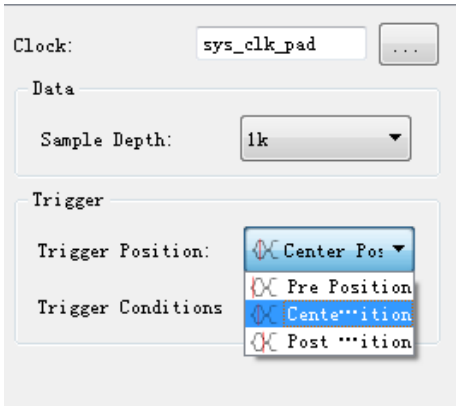
6. 点击 **Clock** 一栏后的…按钮，添加采样时钟，该信号将作为整个 ChipWatcher 模块的工作时钟，需要确保选中有效且恰当的时钟信号，否则可能导致无法正

确触发或者采样精度出现偏差；

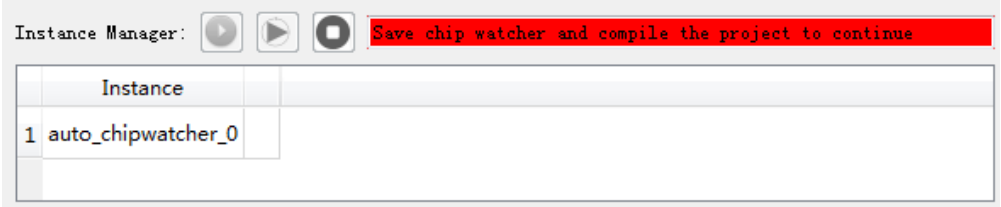


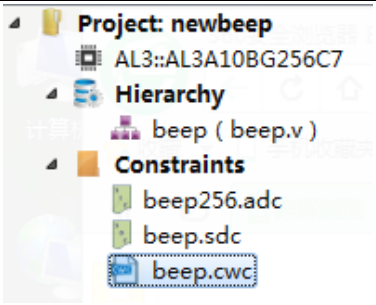
7. 选择采样深度和设置触发的位置。Pre Position 表示触发位置将处于整个采样数据的前三分之一处；Center Position 表示触发位置将处于整个采样数据的二分之一处；Post Position 表示触发位置将处于整个采样数据的后三分之一处；



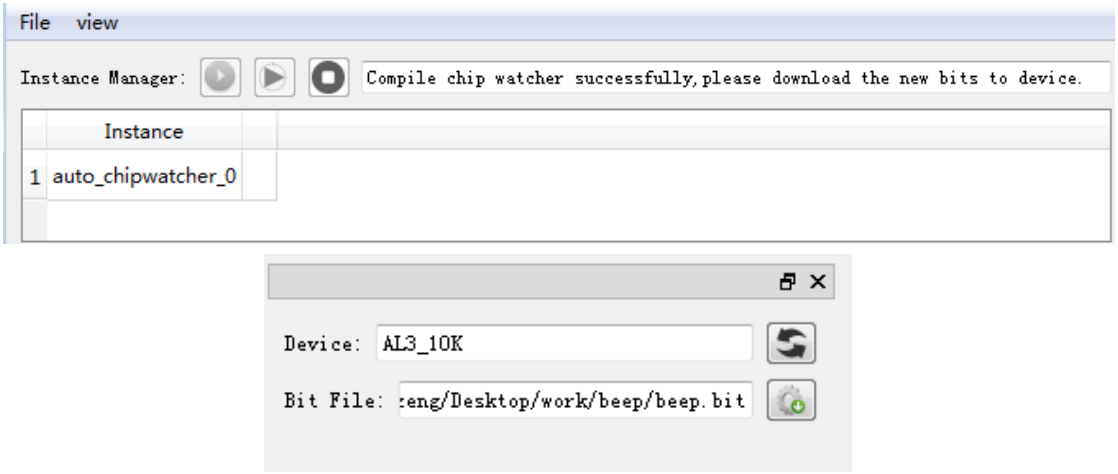


8. 设置好所有参数后，根据界面给出的提示，点击左上角的 **File → Save**，或使用快捷操作 ctrl+s，保存 ChipWatcher 的配置文件(.cwc)，将配置文件添加到工程，并重新编译工程；

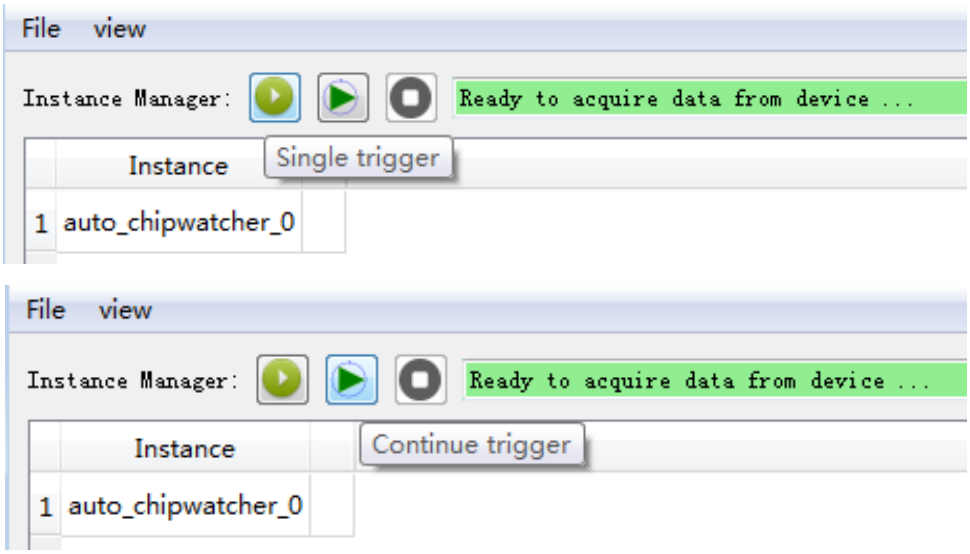




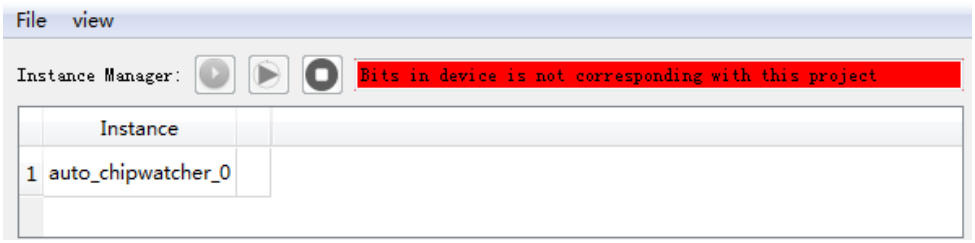
9. 编译完成后，根据提示下载新生成的 bit 文件；



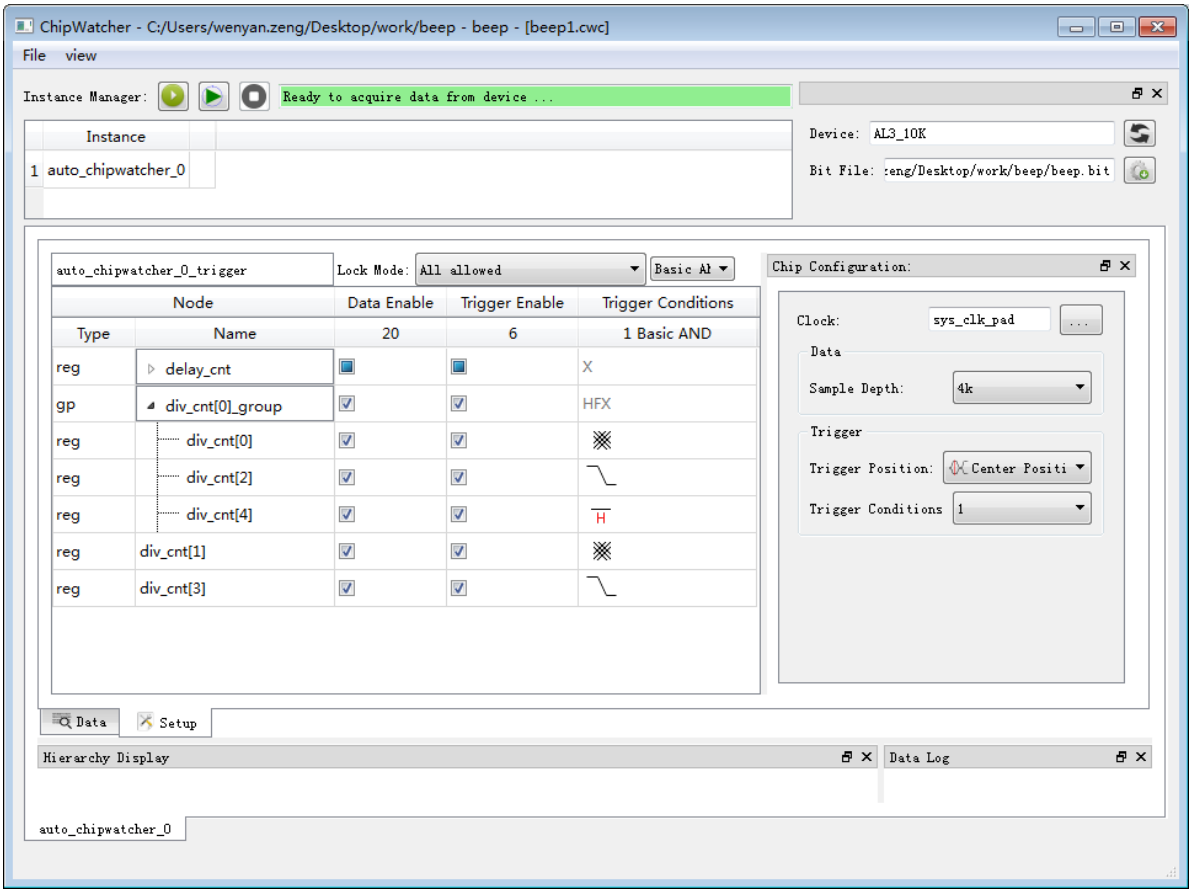
10. 下载完后，ChipWatcher 左上角的触发按钮变亮，并给出如下提示，此时，点击触发按钮，ChipWatcher 将开始监控指定的信号，一旦满足预设的触发条件，便会返回芯片中的数据。其中，Single trigger 为单次触发，即获取芯片中当前时刻当前条件下的数据；Continue trigger 为连续触发，可实时获取芯片中该条件下的数据；

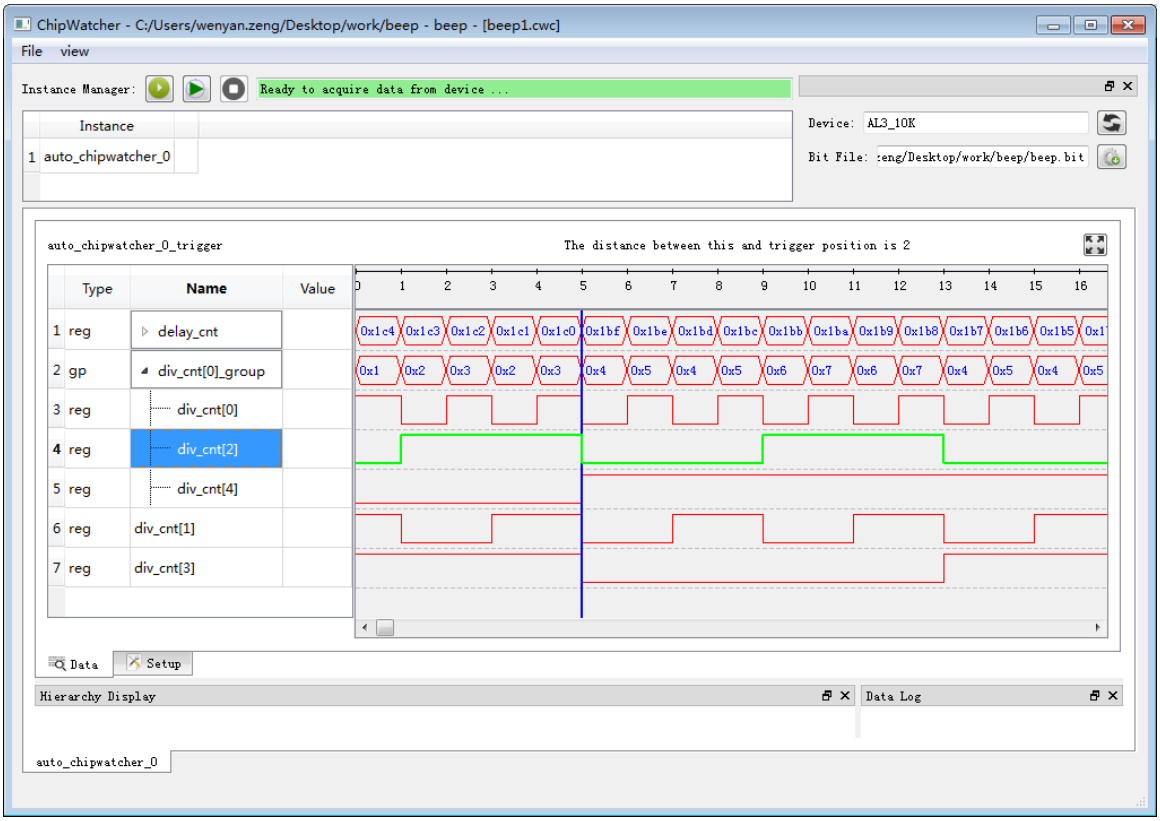


若下载的 bit 文件与当前 ChipWatcher 对象不符，将会给出如下提示。

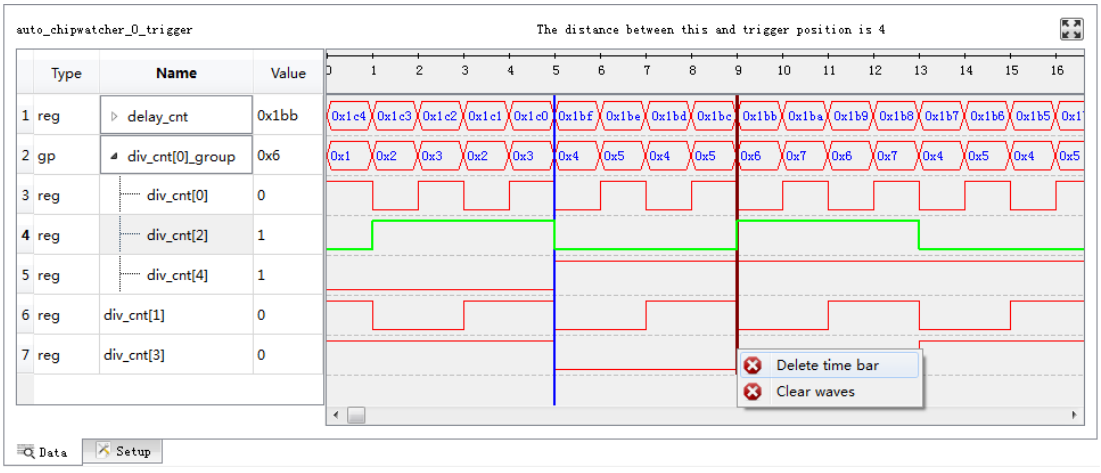


11. 一旦信号被触发，ChipWatcher 页面将由 Setup 界面切换至 Data 页面，并显示读回信号的波形。其中蓝色竖线表示触发位置，不可删除；

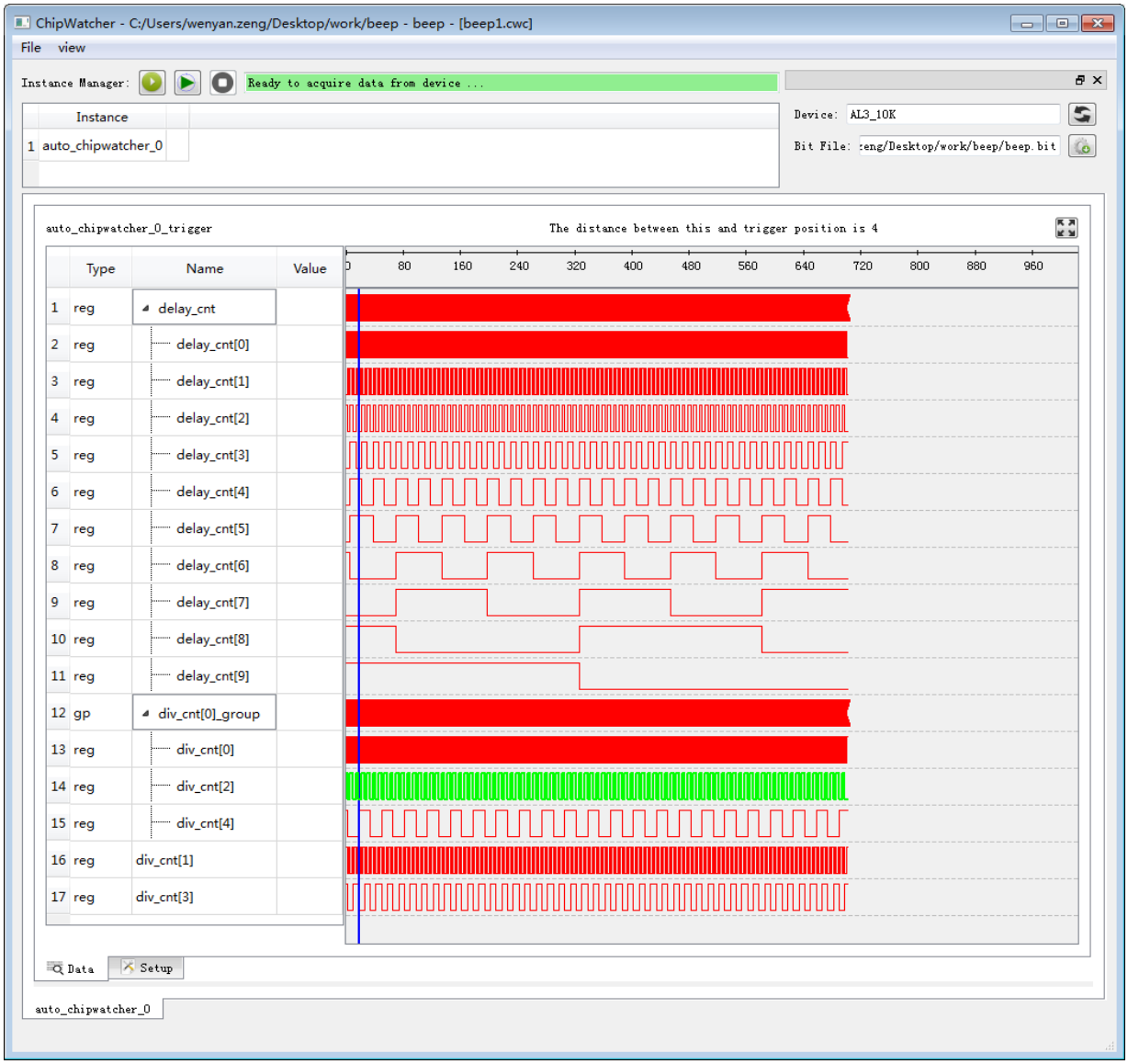




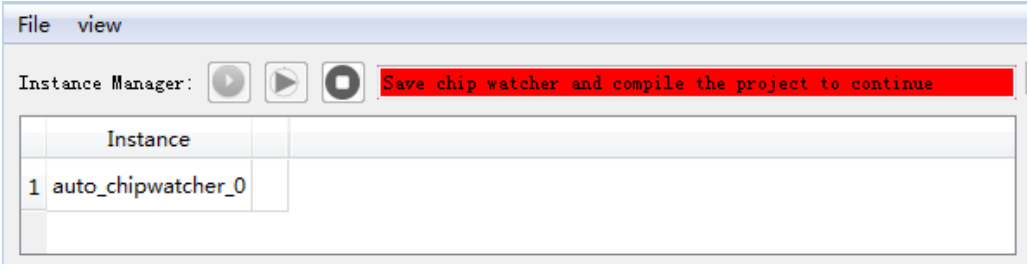
用户可双击波形某位置添加 **time bar**，如下图中的深红色线，**time bar** 可通过鼠标进行移动，也可右键单击进行删除。在移 **time bar** 时，左侧的 **Value** 一栏会显示该处各信号的值，并给出该位置至触发位置的距离。也可右键单击波形，选择“**Clear waves**”，将当前波形清除，重新触发。



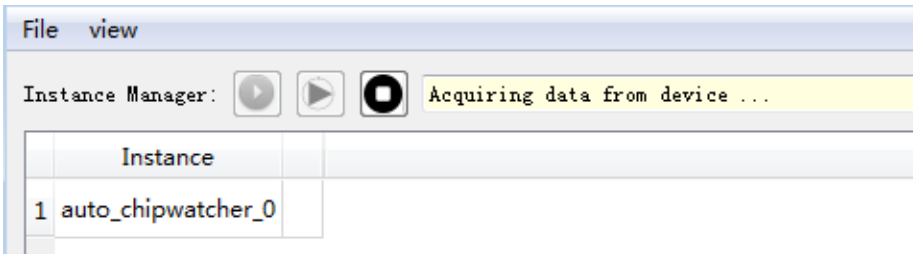
12. 当鼠标处于波形区域，鼠标滚轮向上表示放大波形，滚轮向下表示缩小波形，同时，也可点击右上角的“**Fit for view**”显示整个波形状态；

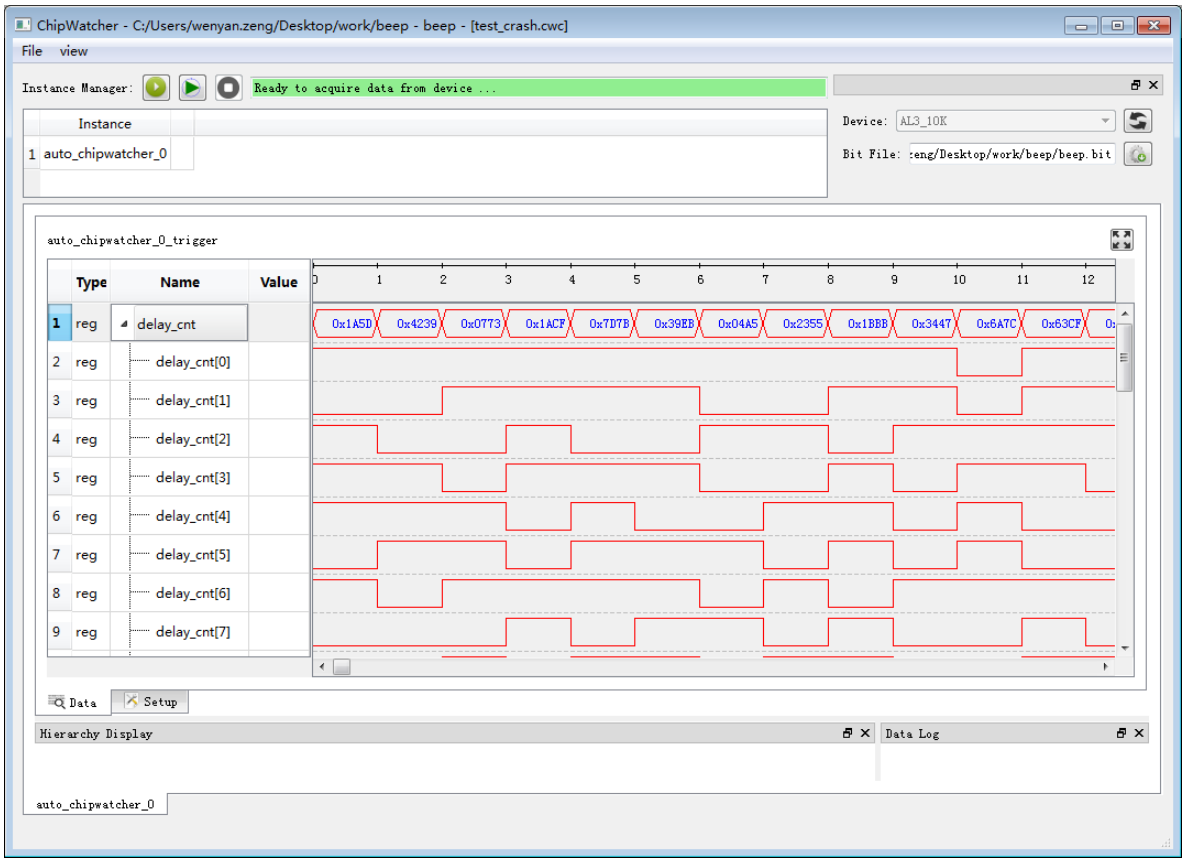


13. 若用户只更改了触发条件(如:将上升沿更改为下降沿,将低电平更改为高电平等),无需重新编译即可直接进行再次触发。而当用户更改了信号数量、采样深度、采样位置等条件时，则需重新保存、编译并下载，才能进行再次触发，此时，Chipwatcher 也会给出相应提示:



14. 若点击触发后，一直处于如下状态，表示芯片中数据无法满足触发条件，请更改触发条件，重新触发；此时，点击 stop 按钮，data 界面将导出当前关注信号的波形。





在没有工程的情况下使用 ChipWatcher：

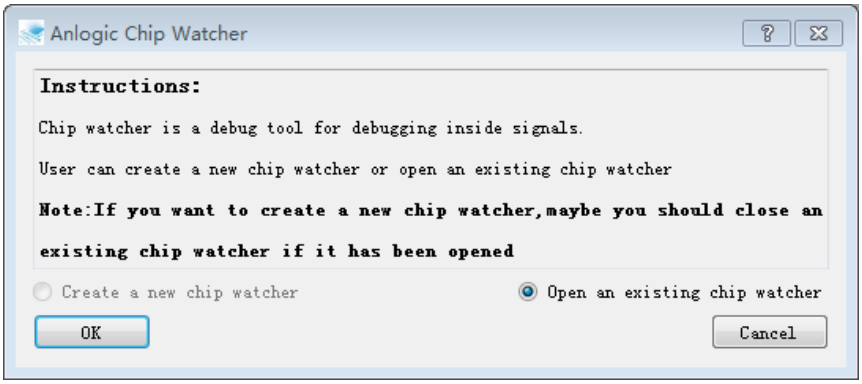
在没有工程的前提下，依然可以使用 ChipWatcher 打开已存在的 cwc 文件进行波形查看。需要注意以下问题：

1. 需要保证 ChipWatcher 所需文件 .cwc，.bid，.bit 文件在同一个文件夹；
2. 脱离工程的 ChipWatcher 界面不可添加、删除和更改信号；

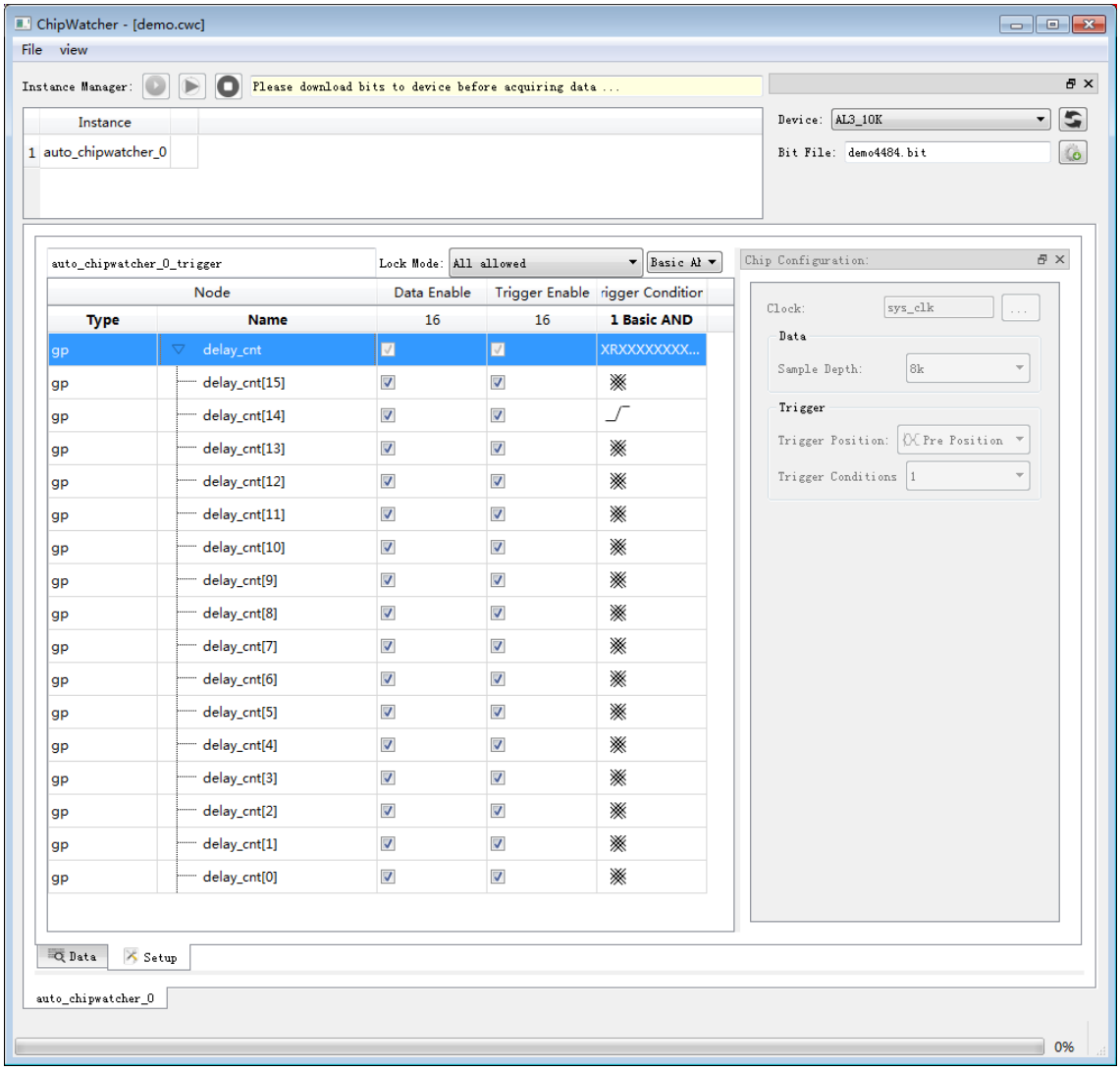
3. 脱离工程的 ChipWatcher 界面不可更改采样深度和触发位置。

具体使用方法如下：

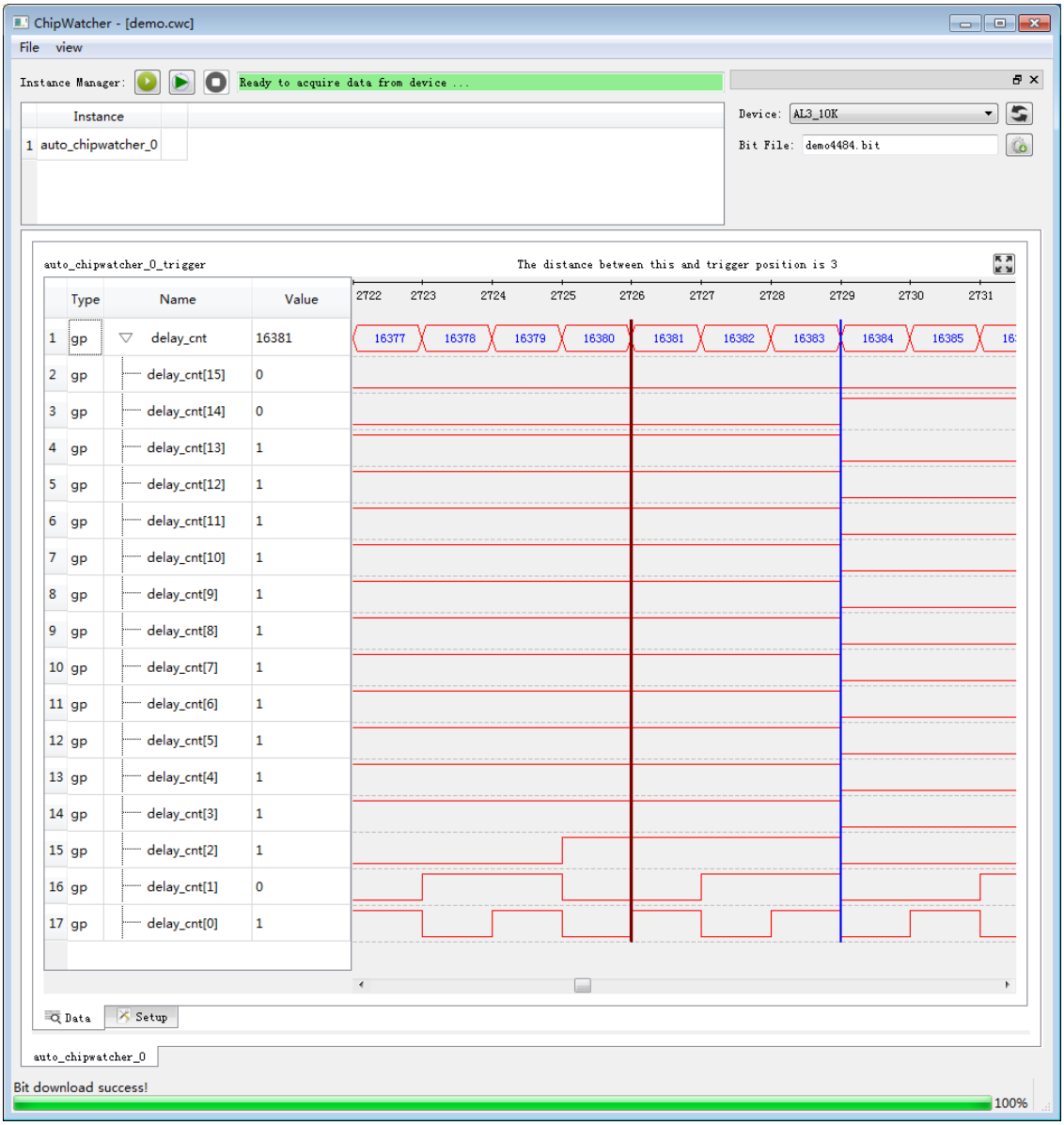
1. 打开 ChipWatcher 界面，只能选择“Open an existing chip watcher”，并点击“OK”；



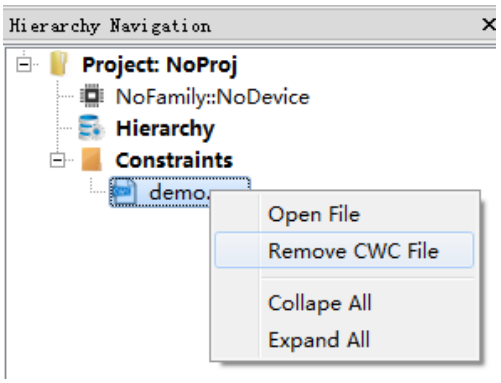
2. 打开一个已存在的 .cwc 文件；



3. 可修改触发条件，进行下载并触发；



4. 当需要打开另一个 .cwc 文件时，需要先将当前文件关闭并移除

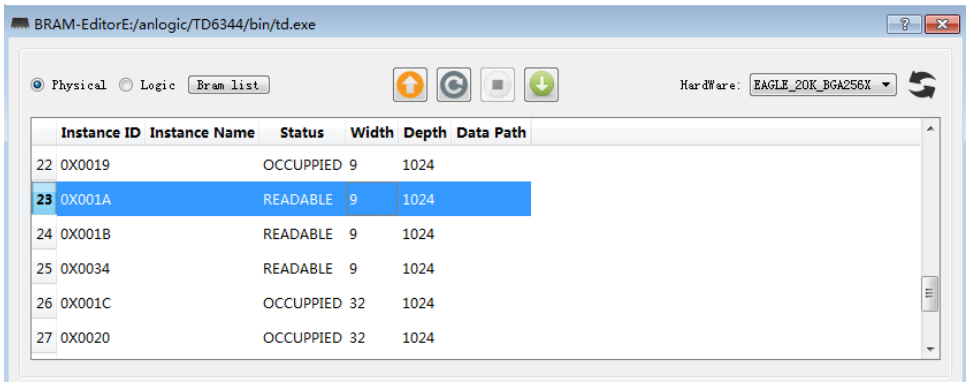


8.4 BramEditor

用户可以使用 **BramEditor** 从芯片中的 RAM 读取数据，并可对这些数据进行修改，修改后写进芯片，即可看到改动效果。

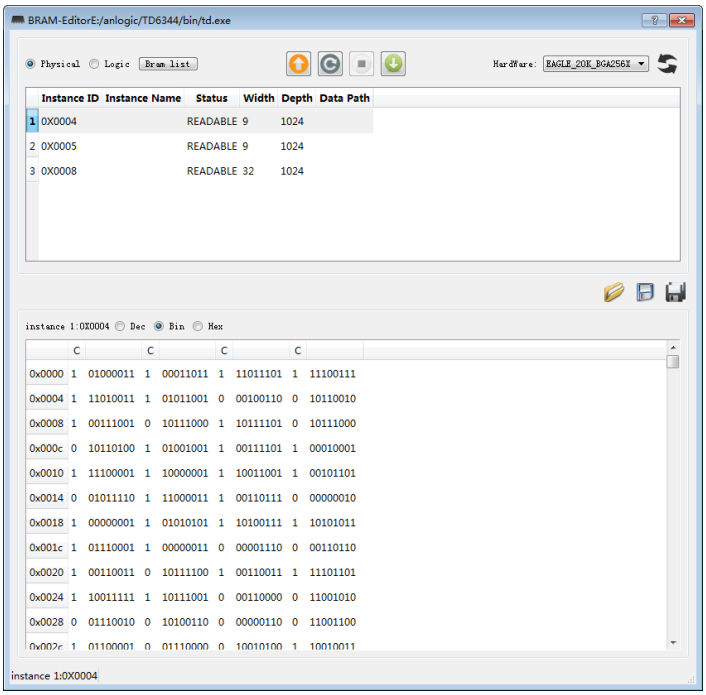
1. 展开 **Tools → Debug Tools** ，选择 **BramEditor；**

2. 若 **Hardware** 一栏显示“No Hardware”，请检测硬件各接口是否连接正确，以及芯片是否上电，最后点击旁边的刷新按钮进行刷新。在弹出的 **BramEditor** 对话框中选择一个 Instance，然后可对该 Bram 的信息进行读写。只有 **Status** 为 **READABLE** 的 Instance 可进行读写操作；

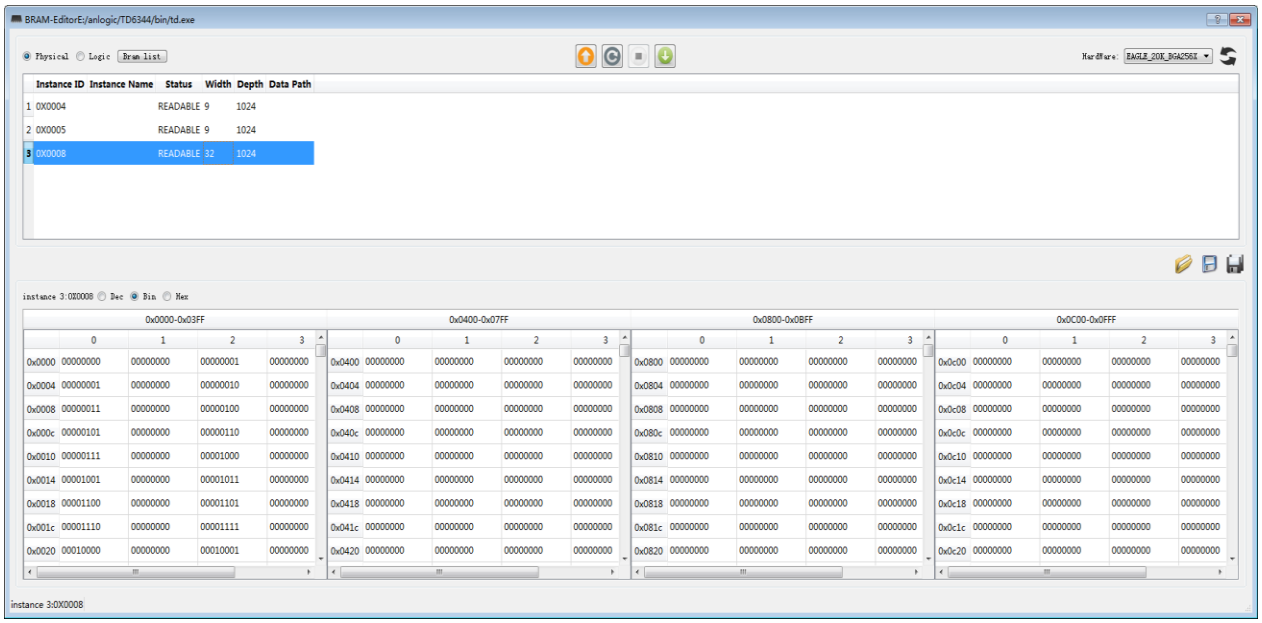


3. 点击按钮，从芯片读数据，用户可选择用十进制(**Dec**)、二进制(**Bin**)或十六进制(**Hex**)来显示读回的数据，默认为二进制。对于 Physical BRAM9K，深度为 1024，宽度为 9 位，最高位为校验位（第九位）；对于 Physical BRAM32K，深度为 1024，宽度为 32 位，没有校验位。对于 Logic BRAM，深度和宽度与用户设计的一致；

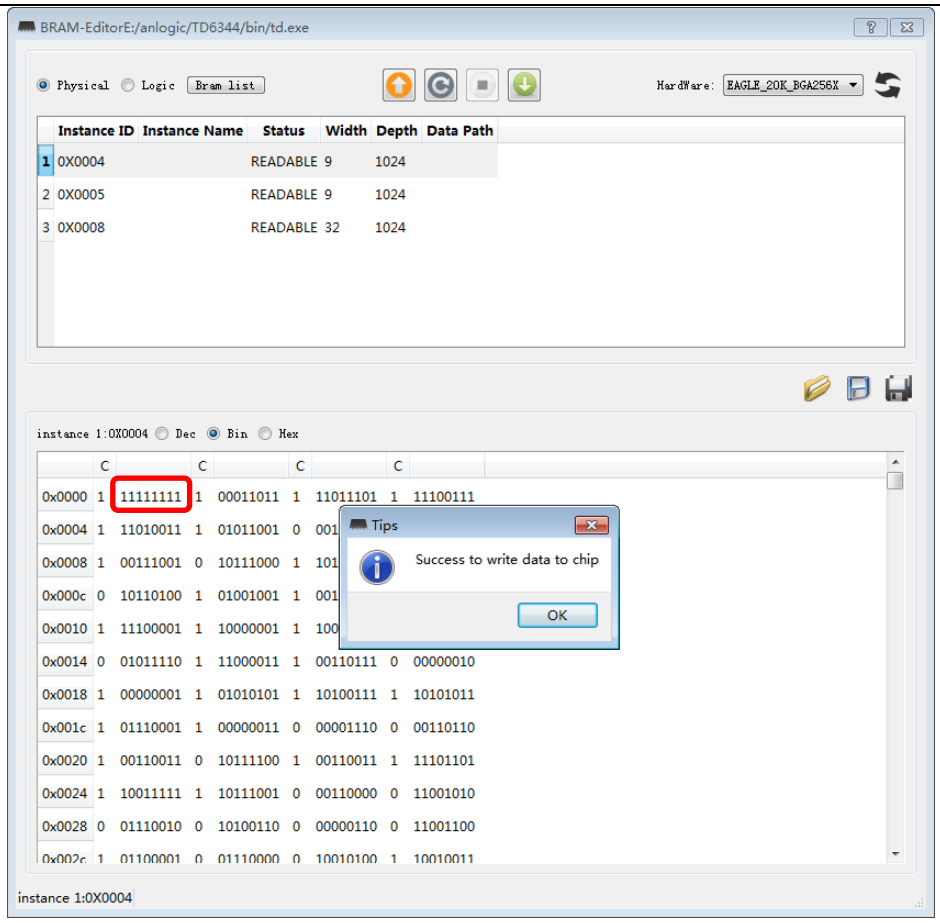
BRAM9K 的数据显示如下：



BRAM32K 的数据显示如下：

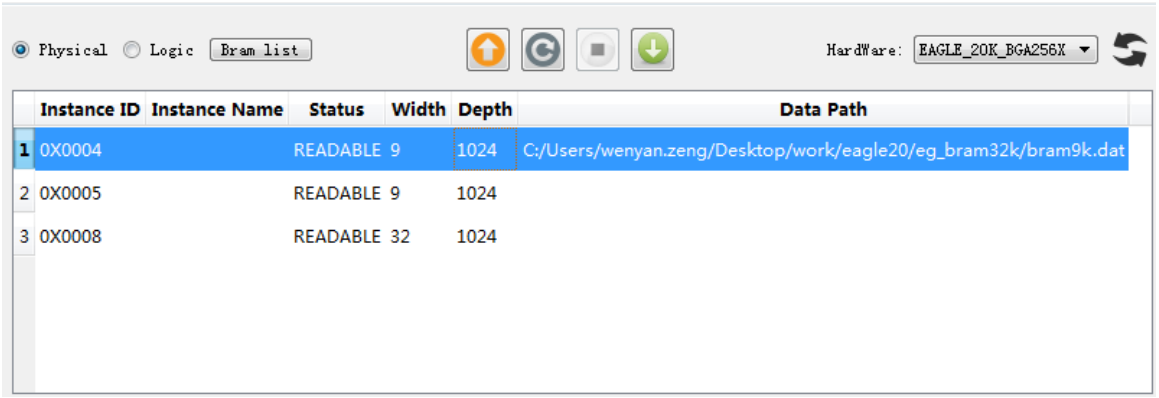


4. 双击某个数据可进行修改，修改后点击按钮将数据写回芯片。可使用按钮循环读取数据，按钮可停止循环；

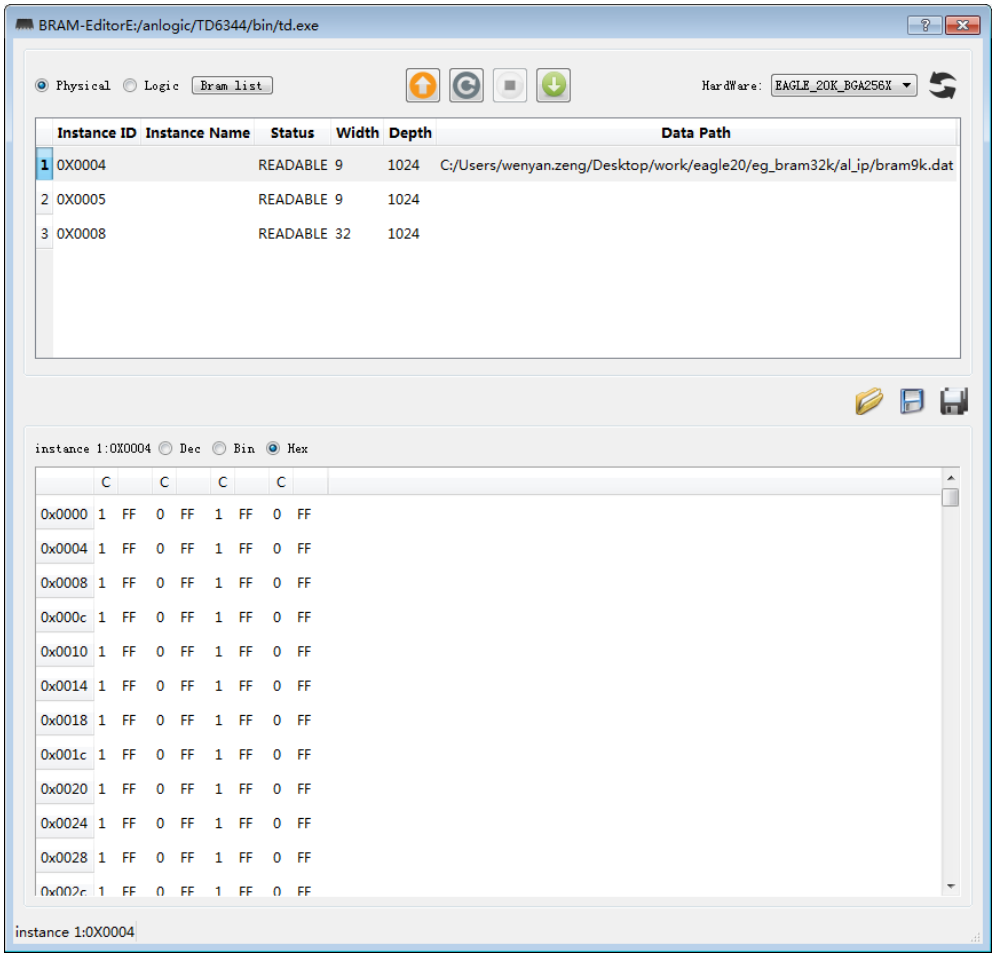


也可对 RAM 中的数据进行批量写入。

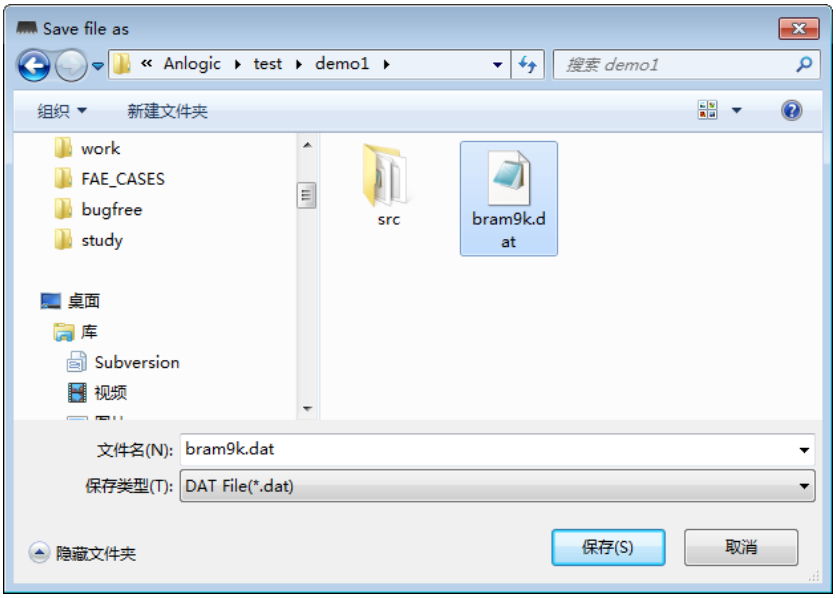
选择一个 Instance，点击打开一个待写入的.dat 文件，点击，将数据写入芯片，将会提示数据写入成功。若写入的数据与 RAM 的大小不相符，则会给出警告。最后点击，即可查看到写入后的数据。



写入 RAM 后，再读回来的数据如下所示

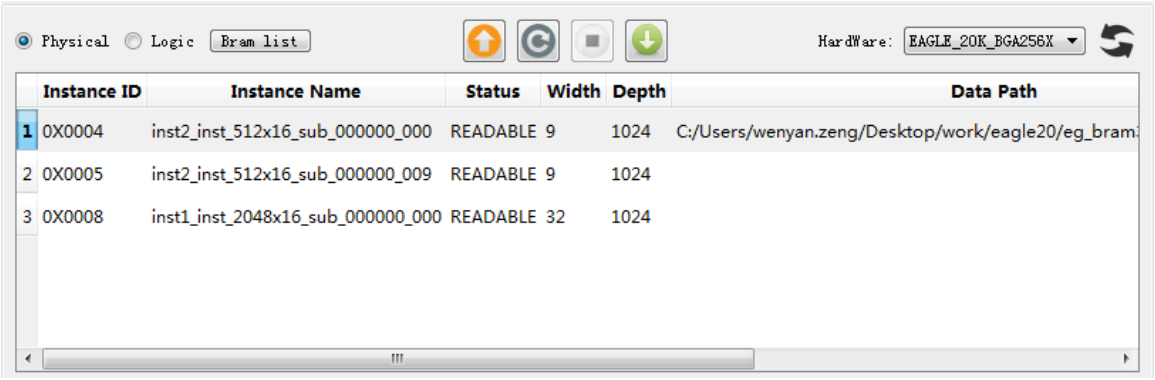


5. 读回的数据，可点击按钮保存为 dat 文件。

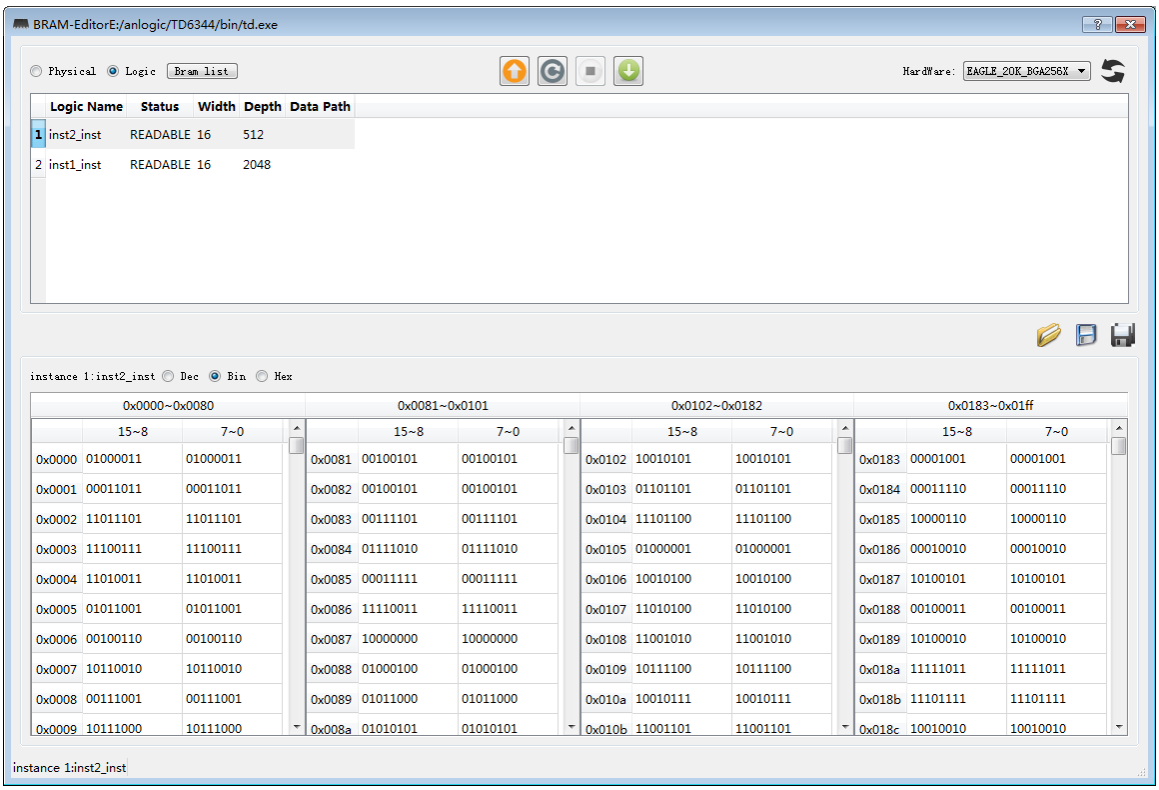


6. Flow 在运行的过程中，会生成一个包含 BRAM Instance Name 的.bid 文件，在

BramEditor 中，可点击界面左上角的 **Bram list** 按钮，将内部 Instance Name 与 Instance ID 对应，方便用户 Debug。



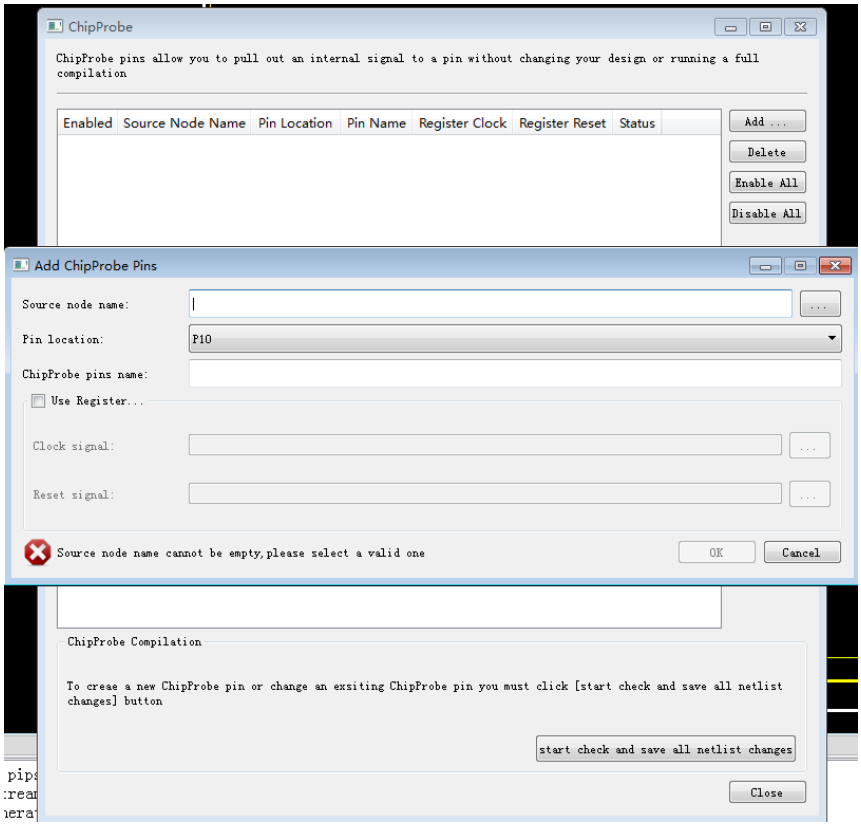
7. 点击 **Logic** 按钮，可切换到 Logic BRAM 显示的界面，其他操作与 Physical BRAM 一致。



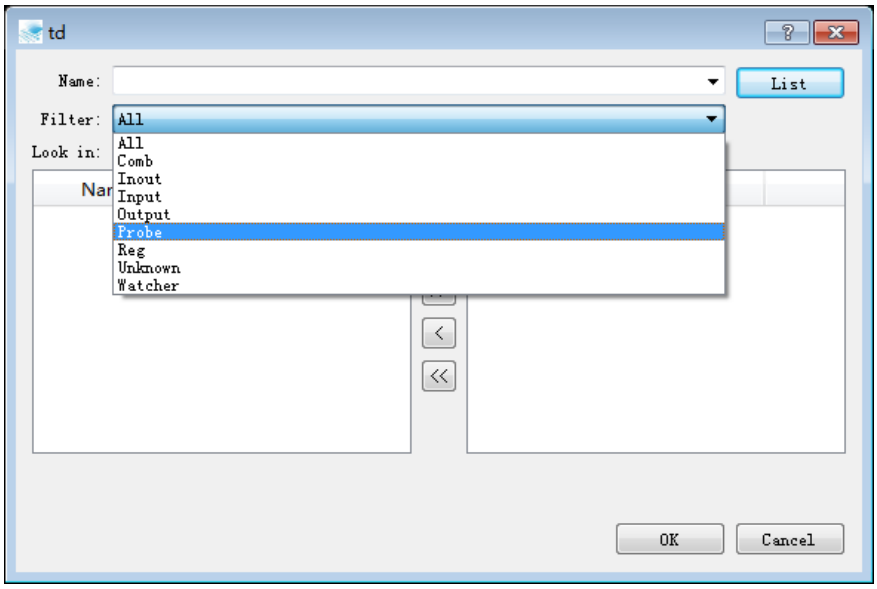
8.5 ChipProbe

使用 **ChipProbe**，用户可在不改变设计的情况下，将内部的一些信号引出到 IO 端口，从而可让用户用外部设备实时检查内部信号的变化情况。

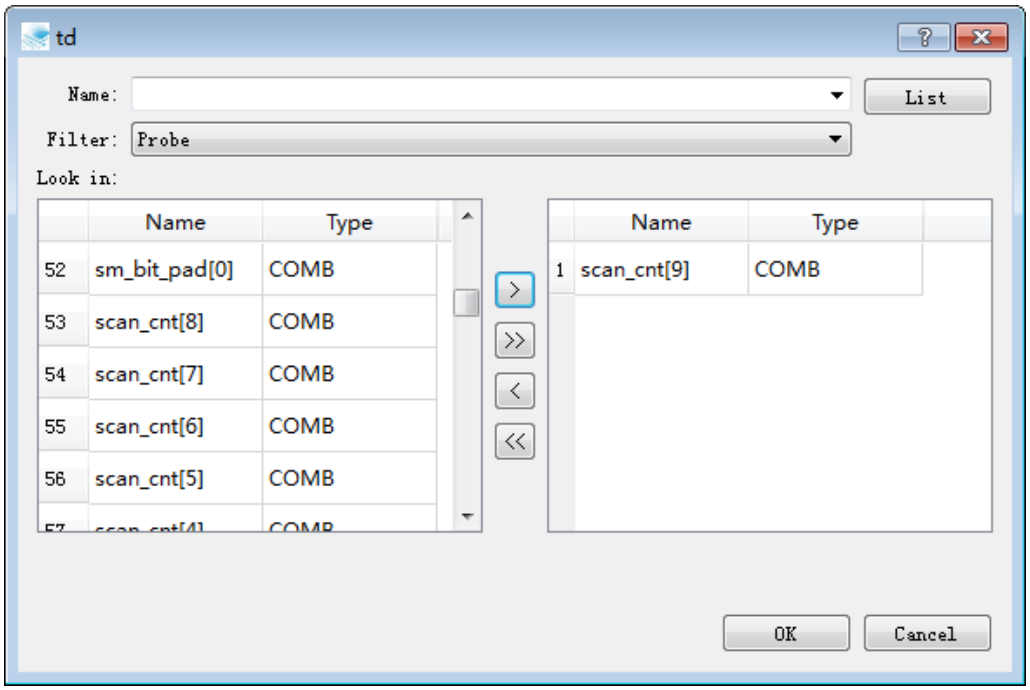
1. 运行完 HDL2Bit 流程后，展开 **Tools → Debug Tools** ，选择 **ChipProbe**
2. 在弹出的 **ChipProbe** 对话框中，点击 **Add** 来添加想要查看的内部变量。



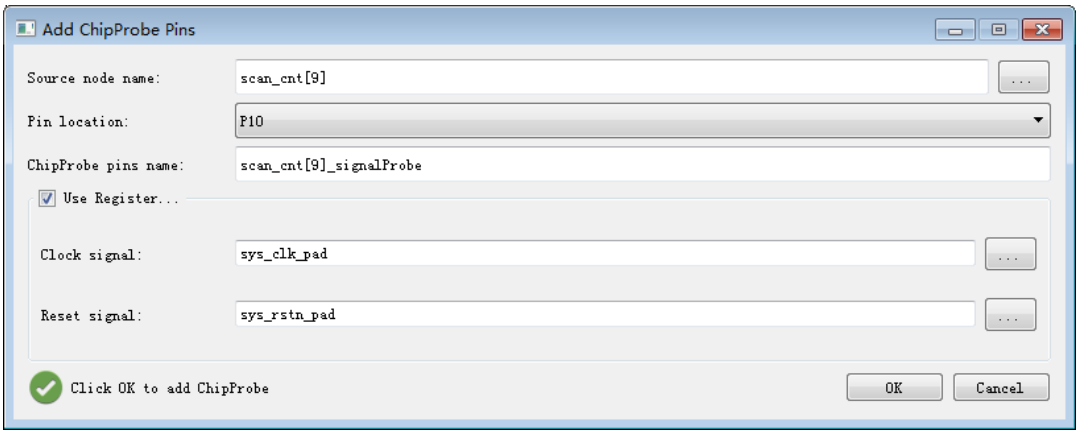
3. 用户可以手动输入 **Source node name**，也可以通过点击 ... 进行添加，在新弹出的对话框中，选择过滤类型为 **Probe**。在使用 ChipProbe 时，只有当所选内部信号的过滤类型属于 Probe 时，才能引出进行调试。



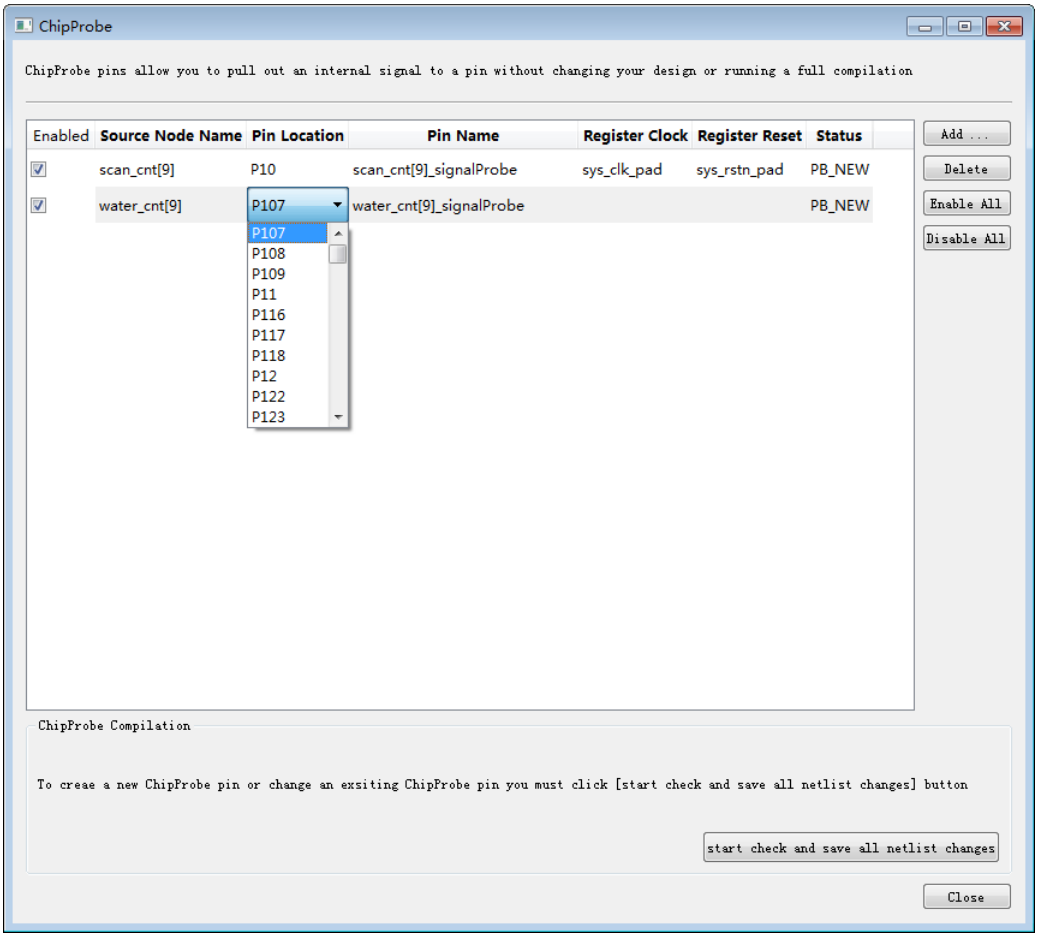
4. 点击 List，选择一个内部变量，点击 进行添加，添加完后点击 **OK**。



5. 添加完后，给内部信号选择一个输出引脚。若需要使用 register 来锁存引出的内部信号，则需要选择一个 **Clock signal**，也可为其添加一个 **Reset signal**，如下图所示。若不需要使用 register，则直接点击 **OK**。



6. 可通过 **Add** 继续添加，也可通过 **Delete** 进行删除。用户还可通过右键单击 **Pin Location** 改变输出引脚。



7. 可勾选 **Enabled** 下面的小方框激活某一个内部信号，也可通过 **Enable all** 激活所有信号，激活后，选择右下角的 **start check and save all netlist changes**，重新编译生成位流文件，下载到芯片后，将外部设备连接事先选定的输出引脚，进行检测调试。

9 附录

9.1 ADC 约束说明

ADC（Anlogic Design Constraint）文件作为 TD 软件的用户物理约束文件，包含由用户指定的各类管脚及单元物理信息相关的约束。

1. 管脚特性约束

定义如下：

set_pin_assignment {pin_name} {attributes}

Pin_name: 电路的管脚名

Attributes: 各类物理相关的属性，目前可支持的约束有 BANK，LOCATION，PULLTYPE，IOSTANDARD，PREEMPHASIS，SLEWRATE，DRIVESTRENGTH，VREF, DIFFRESISTOR，PCICLAMP 等 9 种。其中，对于位置约束（LOCATION），将检查目标位置是否合法，以及是否有多个管脚被分配至同一位置，一旦添加了 ADC 文件，要求为所有管脚锁定位置。

注：PreEmphasis 预加重技术，是一种在发送端对输入信号高频分量进行补偿的信号处理方式，能有效提高输出信噪比。只有 LVDS 输出时需要加上，LVDS_E 是模拟 LVDS 输出，没有预加重的设置。

格式范例：

```
> set_pin_assignment {sys_clk} {LOCATION=P23; PULLTYPE=PULLUP; VREF=
  NONE; SLEWRATE=MED; DRIVESTRENGTH=8; IOSTANDARD=LVCMOS25;
  VCM=0.8; VOD=350m}
```

表 10-1 IO 特性设置参数

PAMERATER	VALUE
BANK	BANK1, BANK 2, BANK 3, BANK 4, BANK 5, BANK 6, BANK 7, BANK 8
LOCATION	芯片封装不同，管脚数和管脚名都不相同
PULLTYPE	PULLUP, PULLDOWN, NONE, KEEPER
IOSTANDARD	LVCMOS12,LVCMOS15,LVCMOS18,LVCMOS25,LVCMOS33,LVDS25, SSTL15_I,SSTL15_II,SSTL18_I,SSTL18_II,SSTL33_I,SSTL33_II, HSTL15_I,HSTL15_II,HSTL18_I,HSTL18_II, VREF1_DRIVER,VREF2_DRIVER
SLEWRATE	FAST, MED, SLOW
DRIVESTRENGTH	4, 8, 12, 16, 20, NA
VREF	VREF1, VREF2, NONE
DIFFRESISTOR	100, NONE
PCICLAMP	ON, OFF
VCM	0.8, 0.9, 1.2
VOD	150, 200, 250, 350, (add 480 for eagle_20 & eagle_s20)，单位为 mv

注：只有当用户设置了 LVDS25 的情况才能设置 VCM，VOD 选项。

2. 单元特性约束

定义如下：

set_inst_assignment {inst_name} {attributes}

Inst_name: 电路的单元实例名，instance 的名字可在生成的网表文件（prj_gate_sim.v）中查找。

Attributes: 目前仅支持位置约束 Location，location 可以在 Chip Viewer 中查找。Chip Viewer 的详细使用手册，请参考该文档的 9.2 Chip Viewer 章节。

格式范例：

```
> set_inst_assignment {PLL_INST1} {location = x34y37z0;}
> set_inst_assignment {PLL_INST2} {location = x0y0z0;}
> set_inst_assignment { _al_u451|lcd/lcd_rs_reg} {location = x21y17z1;}
```

9.2 SDC 约束说明

TD 的 Timer 模块支持用户输入的时序约束格式为 SDC 标准格式的常用命令子集，下面列出了 TD 支持的 SDC 命令以及相关参数选项。

1. 指代对象相关：

   a) **all_clocks / all_inputs / all_outputs / all_registers**：指代全部该类别对象；

   b) **get_cells / get_pins / get_nets** [-hierarchical] [-nocase] [-nowarn] <filter>

   返回设计中 cells / pins / nets 的集合，-hierarchical 表示层次化搜索，将在每一层次的模块中匹配对应名称，-nocase 表示不区分大小写，-nowarn 表示没有匹配成功时也不给出警告。

   c) **get_clocks / get_ports** [-nocase] [-nowarn] <filter>

   返回时钟/输入输出端口的集合，参数含义同上。

2. 时钟设定相关（时钟信息是最基础的时序约束）：

   a) **create_clock** -add -name <string> -period <double> -waveform <string> target

   定义一个时钟：target 指明了时钟源，可以是 nets 或 ports，如果 target 为空则说明定义了一个虚拟时钟；-name 指定了时钟名，如该项为空则时钟名为 target 列表的第一项；-period 为周期，该选项必须指定，且数值需大于 0；-waveform 指定了第一个上升沿与第一个下降沿的时间点，暂时仅支持每周期包含两个时钟沿的情况；-add 说明在 target 管脚上已经定义过时钟的情况下，将新时钟加入至该管脚，否则覆盖已有时钟，主要用于 clock multiplexer 的情况。

   格式范例：

   > create_clock –name sys_clk -period 20 -waveform {0 10} [get_ports sys_clk]

b) **create_generated_clock** -add -name <string> -source <list> -divide_by <double> -multiply_by <double> -edges <string> -duty_cycle <double> -invert -edge_shift <string> -master_clock <string> target

定义一个派生时钟，属于 master clock 所在的时钟域，在时序报告中也将具有和 master clock 相同的 start point。

-source 指定了其 master clock 所在的源点，可以是 nets 或 ports 的列表，-master_clock 指定了 master clock 的名称；-name 指定了时钟名，如果该项为空则时钟名为 source 列表的第一项；target 为当前生成时钟的源点。-add 说明在 target 管脚上已经定义过时钟的情况下，将新时钟加入至该管脚，否则当前生成时钟被忽略，这点与 master clock 定义时不同。

生成时钟的周期与波形由 master clock 调整而来，-divide_by / -multiply_by 指频率除以/乘以指定倍数，-invert 与这两条选项配合使用，使时钟波形反转，而 -duty_cycle 配合-multiply_by 选项使用，用以调节占空比；-edges 选项包含三个整数，分别指定了新生成时钟的第一个上升沿，第一个下降沿，第二个上升沿对应源时钟的第几个边沿；-edge_shift 选项将指定-edges 选项中三条时钟沿的偏移量，因此将包含 3 个正负皆可的整数，单位为基本时间单位（默认为 ns）。

格式范例：

> create_generated_clock -divide_by 1.25 -source [get_ports clk] -name sys_clk [get_ports sys_clk]

c) **derive_pll_clocks** [-gen_basic_clock]

自动在所有用到的 PLL clkc[x]端口生成时钟约束，生成时钟的频率、相位都将严格按照 PLL 内部的参数设定。-gen_basic_clock 将在对应的 PLL refclk 上定义

FIN 频率的基准时钟，否则将自动搜索 refclk pin 以及所连 net 上定义的时钟。

该命令生成的时钟将在 flow 运行时才生效，因此在 timing wizard 后续的设置中还无法引用。

> derive_pll_clocks -gen_basic_clock

d) **derive_clocks** -period <double> -waveform <string>

在各个未定义时钟的时钟管脚上指定一个默认时钟，-period 为周期，该选项必须指定，且数值需大于 0；-waveform 指定了第一个上升沿与第一个下降沿的时间点，暂时仅支持每周期包含两个时钟沿的情况。

格式范例:

> derive_clocks -period 10 -waveform {0 5}

e) **set_clock_latency** -clock <list> -max -min -source delay

设定时钟的延时，delay 为延时的数值；-clock 指定时钟的列表；

-max/ -min 选项指定本命令所指定的为最大/最小延时，默认为两者相同；

-source 选项说明指定的为 source latency，否则为 network latency。

network latency 在布线后的时序分析中，将被实际的互联延时所取代。

> set_clock_latency -clock [get_clocks {clk_vga_25m}] -source 2

f) **remove_clock_latency** -clock <list> -source target

取消之前所设定的时钟延迟，各选项含义同上。

格式范例:

> remove_clock_latency -source [get_clocks clk_vga_25m]

g) **set_clock_uncertainty** -setup -hold uncertainty target

设定时钟的 uncertainty 数值，目前仅支持对于单个时钟本身设置而不支持跨时钟域的 uncertainty 定义；target 为 clock 的列表，uncertainty 为数值项；-setup/-hold 选项指明本命令所对应的是最大/最小路径时序分析时所对应的数值，如果该选项没有指明，则对两种检查同时生效。

格式范例：

> set_clock_uncertainty -setup -hold 1.00 [get_ports clk]

h) **remove_clock_uncertainty** -setup -hold target

移除之前所设定的时钟 uncertainty 数值，各选项含义同上。

格式范例：

> remove_clock_uncertainty -setup -hold [get_ports clk]

3. 时序路径约束设定：

a) **set_input_delay / set_output_delay** -clock <list> -max -min delay target

设置输入/输出端口的延时，delay 项为延时数值，target 可以是 pins 或 ports 的对象列表；-clock 指定了延时所对应的时钟信息；-max/-min 选项则指定了本命令所设置的值为最大/最小延时，默认为相同。

格式范例：

> set_input_delay -clock sys_clk 1.0 [all_inputs]
> set_output_delay -clock lcd_clk 3 [get_ports {disp_RGB} {led} {sm_bit}]

b) **remove_input_delay / remove_output_delay** -max -min target

移除之前设定的输入/输出延时，各项参数含义同上。

格式范例:

> remove_input_delay -clock sys_clk 2 [get_ports {data[0]}]

> remove_output_delay -clock sys_clk -0.55 [all_outputs]

c) **set_max_delay / set_min_delay** -from <list> -to <list> -through <list> delay

设定时序路径的允许最大/最小延时，delay 项为延时数值；-from 必须为时序路径的起点,即 input 的列表;-to 必须为时序路径的终点,即 output 的列表;-through 选项可以是 nets，ports 列表，指定了该时序路径必须通过的中间点，当有多个 through 选项时，目标时序路径必须依次经过每一个中间点。

格式范例:

> set_max_delay -from [get_ports {sw}] -to [get_ports {sm_bit}] -through [get_nets {lcd/delay_cnt[2]}] 1

d) **set_false_path** -setup -hold -from <list> -to <list> -through <list>

设定时序路径为虚假路径，因而不对其进行时序分析；-setup/-hold 选项指定了在 setup/hold check 时不分析目标路径。-from 为时序路径的起点，可以是 clocks 或 inputs 的列表,-to 为时序路径的终点,可以是 clocks 或 outputs 的列表,-through 指定了该时序路径必须通过的中间点，可是是 ports 或 nets 的列表。

格式范例:

> set_false_path -from [get_clocks clk] -to [get_clocks sys_clk]

> set_false_path -from [get_clocks {clk_vga_25m}] -to [get_ports {sm_bit}] -through [get_nets {dpram_top0/lfsr_done}]

e) **set_multicycle_path** -setup -hold -start -end -from <list> -to <list> -through <list> multiplier

设置允许多个时钟周期延时的时序路径，multiplier 项为时钟周期数；-setup/-hold 选项设定该命令所约束的时序路径类型，仅使用-setup 选项时，setup check 将允许时序路径最多使用 N 个时钟周期，但同时 hold check 会变为要求时序路径最短经过 N-1 时钟周期。仅使用-hold 选项时，则将 hold check 的约束设置为 N 而不影响 setup check 的约束。

在常规使用场景下，为了让 setup check 能使用多个时钟周期而不影响 hold check，需要两条命令搭配使用，即设定一条 N 个周期的 setup 约束，再配合一条 N-1 个周期的 hold 约束。

-start/-end 为跨时钟域时使用的选项，指定延时为 launch/capture 时钟对应的周期，默认情况下，setup check 使用 capture 时钟而 hold check 使用 launch clock。-from 为时序路径的起点，可以为 clocks 或 inputs 的列表，-to 为时序路径的终点，可以为 clocks 或 outputs 的列表，-through 指定了该时序路径必须通过的中间点，可是是 ports 或 nets 的列表。

格式范例：

> set_multicycle_path -setup -end -from [get_clocks {clk_vga_25m}] -through [get_nets {uart/uart_tx/num[3]}] 2

f) **set_clock_group** -exclusive -asynchronous -group <list>

为时钟域设定分组，不分析跨时钟组的时序路径。一般来说，-exclusive 选项表示这些时钟组在逻辑上不会同时出现，而-asynchronous 表示完全不相干的时钟，不过在具体实现以及效果上，这两个选项是一样的，该命令会在所有-group 列

出的时钟之间定义 false path 约束集合。

> set_clock_groups -exclusive -group [get_clocks {clk_vga_25m}] -group [get_clocks {sys_clk}]

g) **set_clock_route** <net_name>

用于指定特定时钟线网不走专用时钟互连。对于 design 中 clock 数目超过了 TD 的限制时，可手动指定哪些 clock 线网不走时钟网络。若指定的时钟不走时钟网络时，会导致布局布线失败，TD 会及时给出 error。

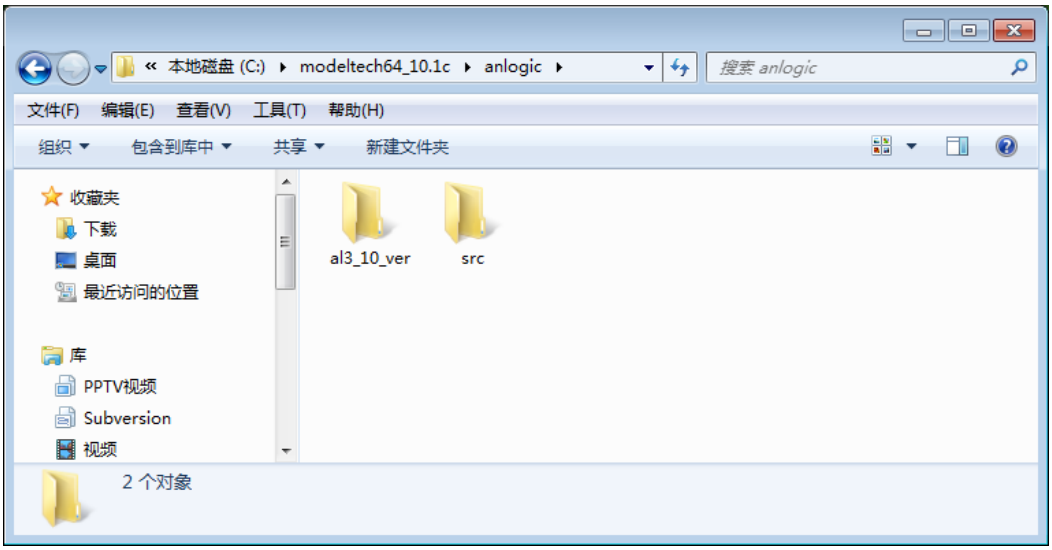
> set_clock_route lcd12864_en_pad

9.3 ModelSim 仿真流程

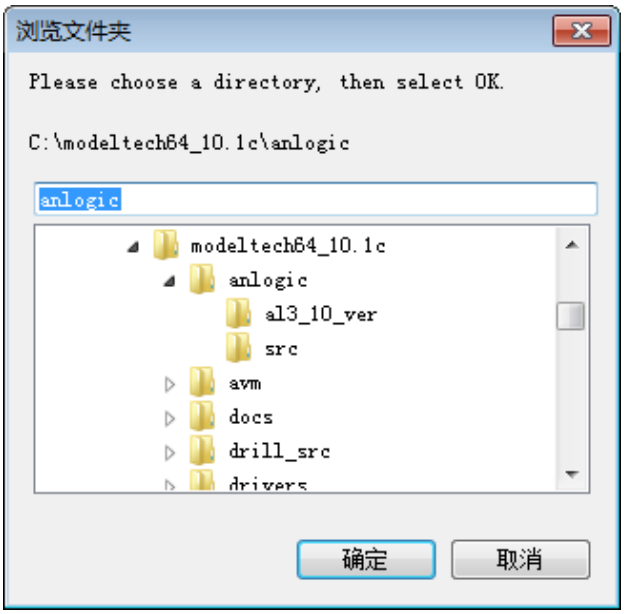
9.3.1 添加仿真库

以 AL3_10 器件为例，TD 软件自带有仿真模型，并可在 modelsim 进行编译，步骤如下：

1. 在 modelsim 的安装目录下，新建文件夹，如：Anlogic,

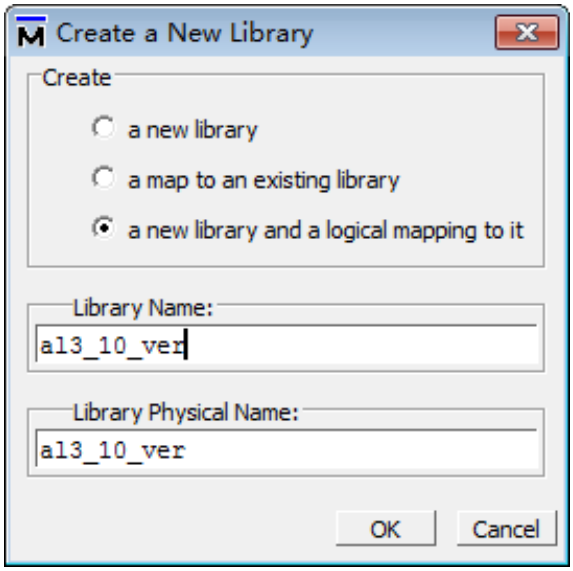


2. 启动 modelsim，选择 file → change directory 将路径转到 anlogic 文件夹下

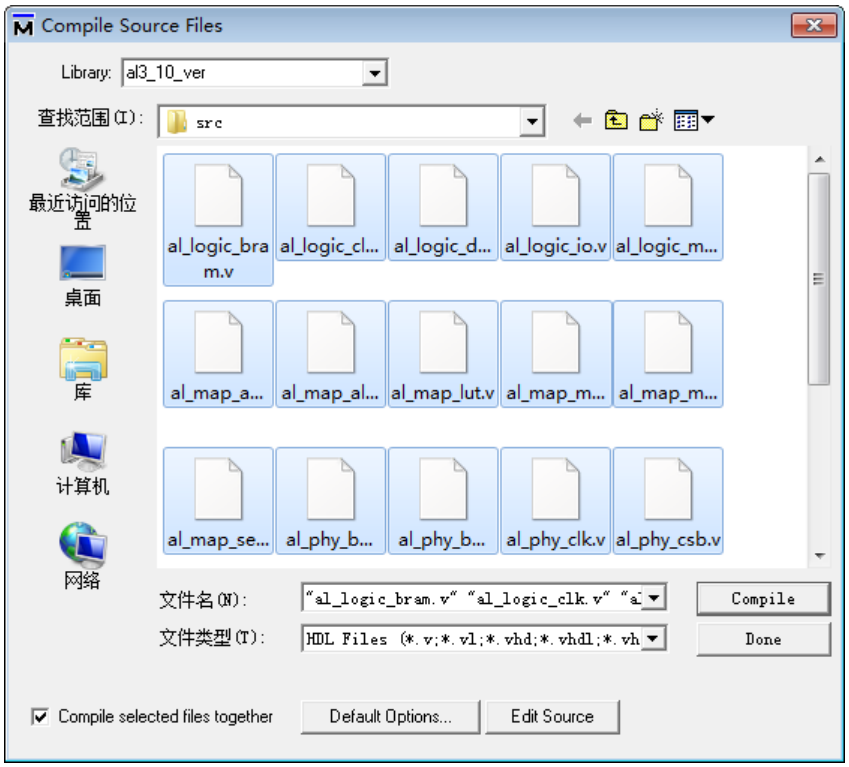


3. 在 anlogic 文件夹下新建文件夹，如：src，以存放 TD 的仿真模型源文件，并将 TD 安装路径下的 sim 目录下的所有文件复制过来。

4. 在 modelsim 的 file → new → library 下新建名为 al3_10_ver 的库

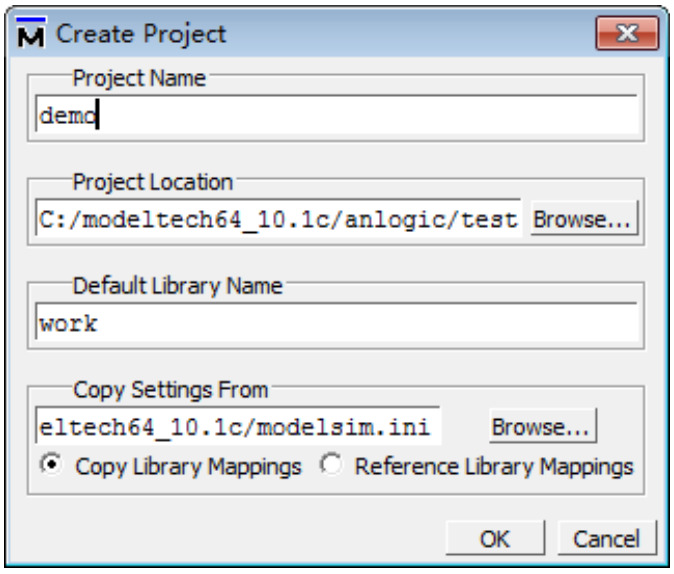


5. 打开 compile → compile，弹出 compile souce files 对话框，library 中选择刚建立的 al3_10_ver，查找范围选择 src 下的所有文件，勾选 compile selected files together，执行编译命令

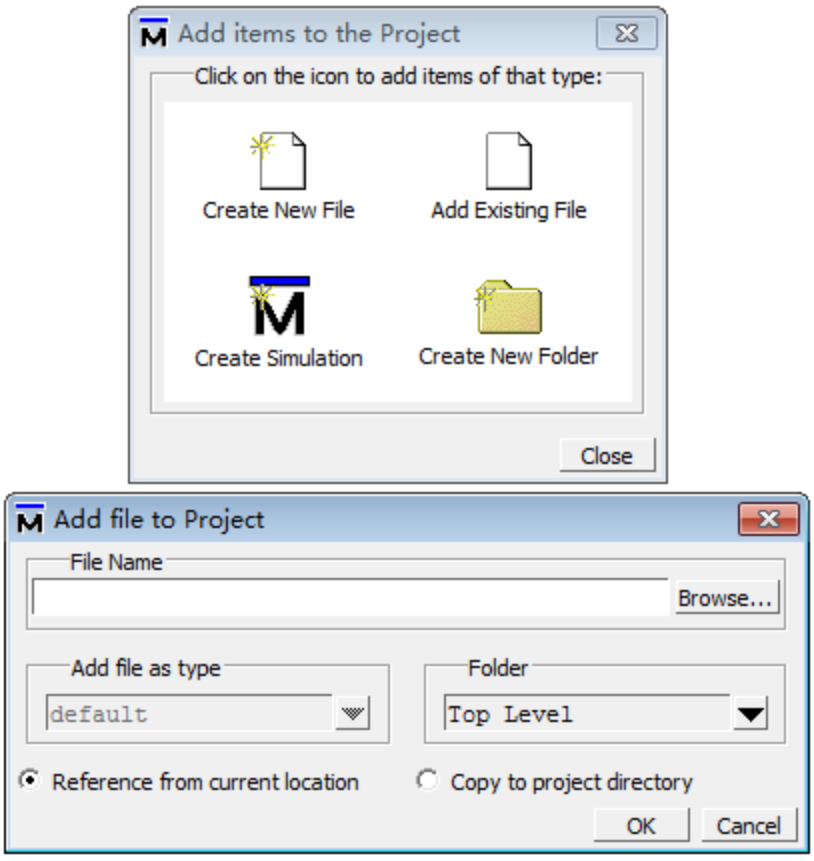


9.3.2 仿真

1. 在 modelsim 中，点击 file → new → project，新建 project，如：demo

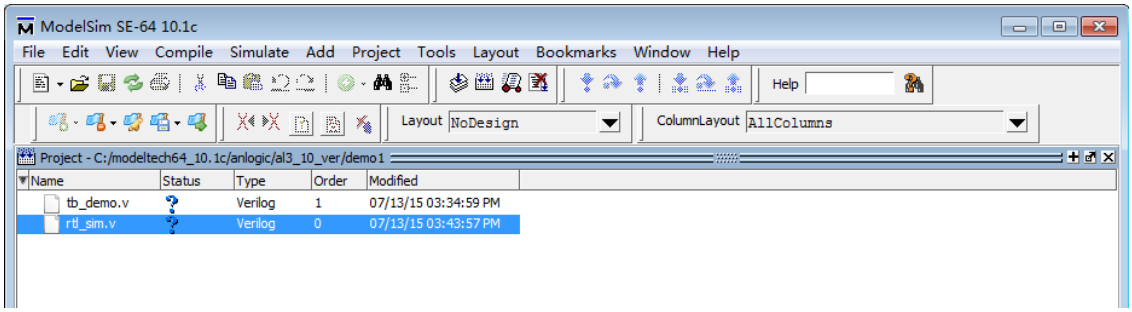


2. 可点击 add existing file 添加设计文件，也可点击 Create New File 创建新的设计文件，并将其添加到工程。

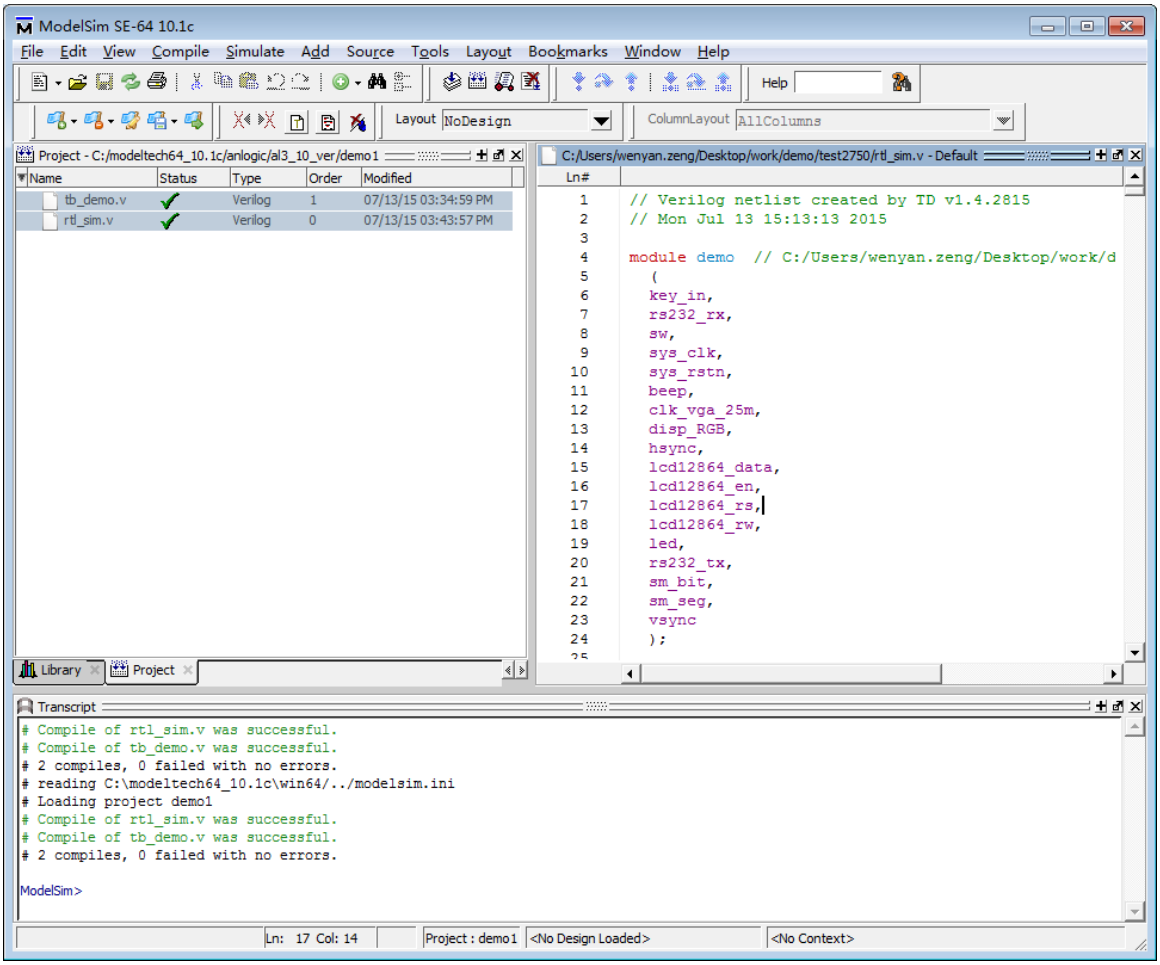


3. 选择一个已存在的设计源文件和其 testbench 文件，这里以 TD 生成的 RTL 级电路仿真模型 rtl_sim.v 为例进行仿真。若仿真时碰到关于 glbl 的问题，请用户在 testbench 中引用 Anlogic 的 glbl 模块，如下所示：

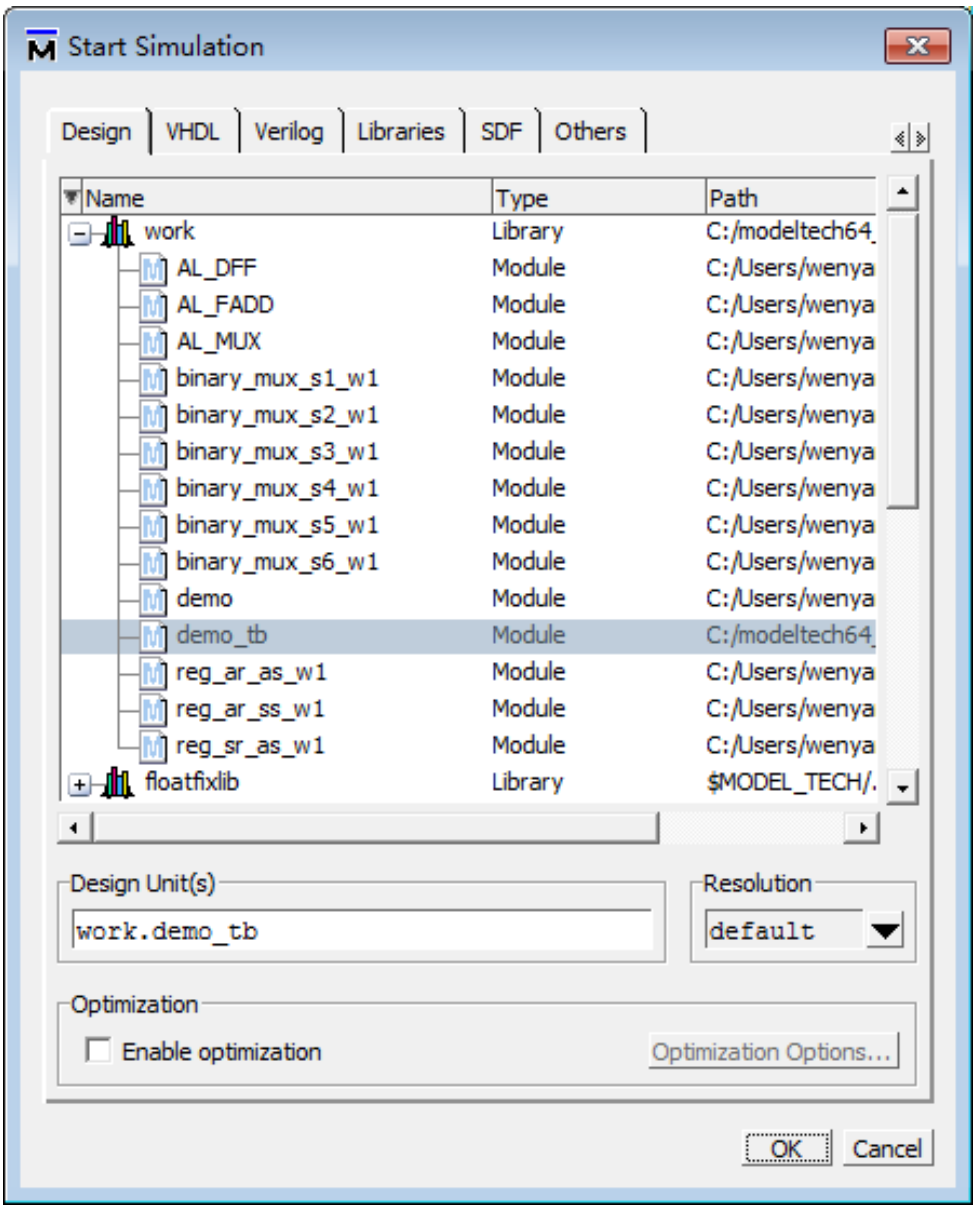
```
20      always
21        #(PERIOD/2) clk1=~clk1;
22      parameter LEAF_PERIOD = 10.0;
23      always
24        #(LEAF_PERIOD/2) clk2 = ~clk2;
25
26        glbl glbl();
```



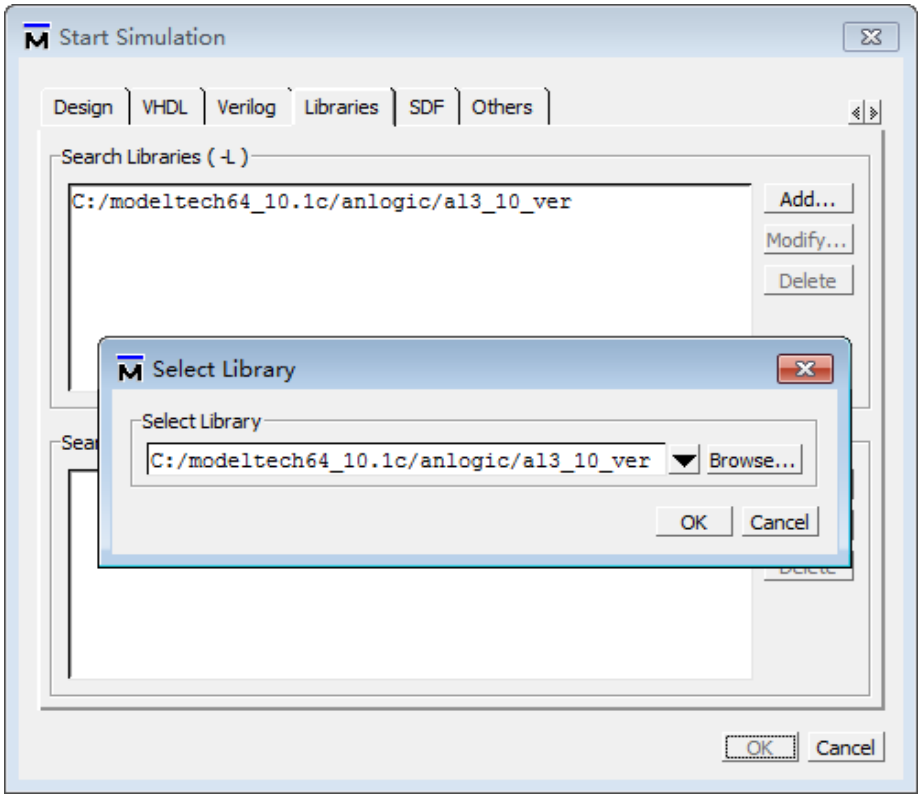
4. 点击进行编译，编译成功后，源文件的状态将会由“？”变成“✔”



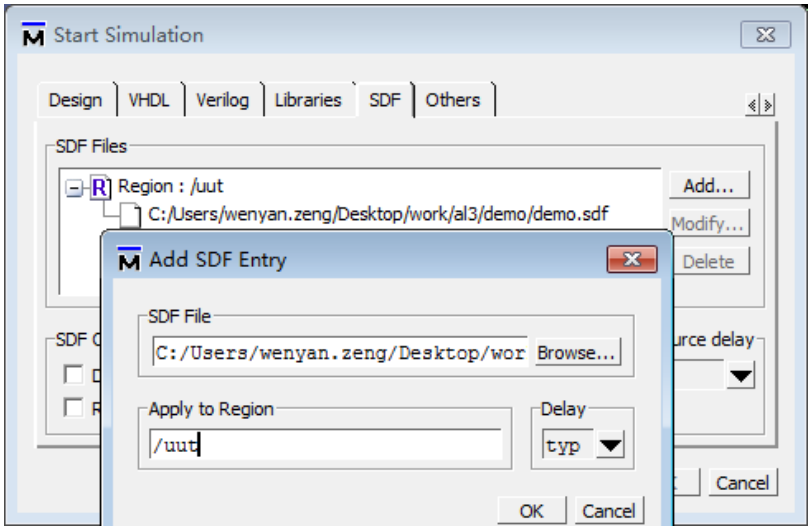
5. 点击 simulate → start simulate，在 work Library 中选择 testbench 文件进行仿真，如果想仿真后，在模块列表中查看各信号参数或波形的变化情况，可将“Enable optimization”前面的勾去掉，否则，Modelsim 会将信号参数优化掉，导致信号列表为空。



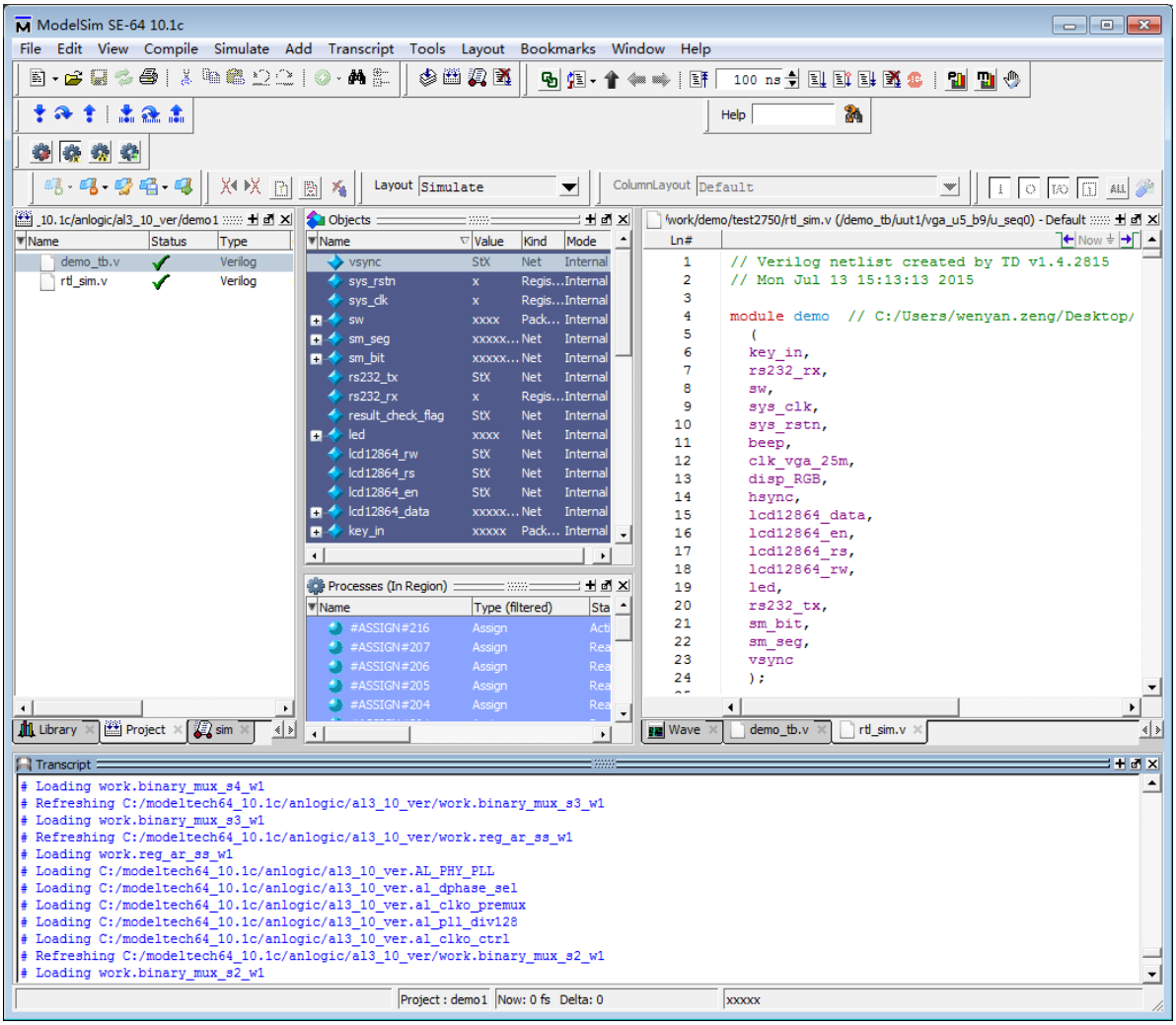
6. 然后选择 libraries 点击 add，选择 al3_10_ver，点击 OK 进行仿真。



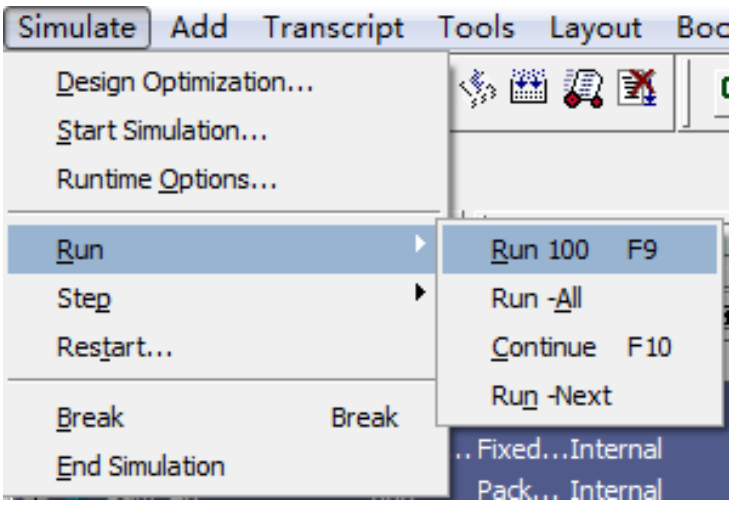
注：如果是时序仿真，在仿真时需指定工程的 sdf 文件，其中 Apply to Region 是指 tb 文件中例化的 Instance name。时序仿真的网表文件为：prj_name_phy_sim.v



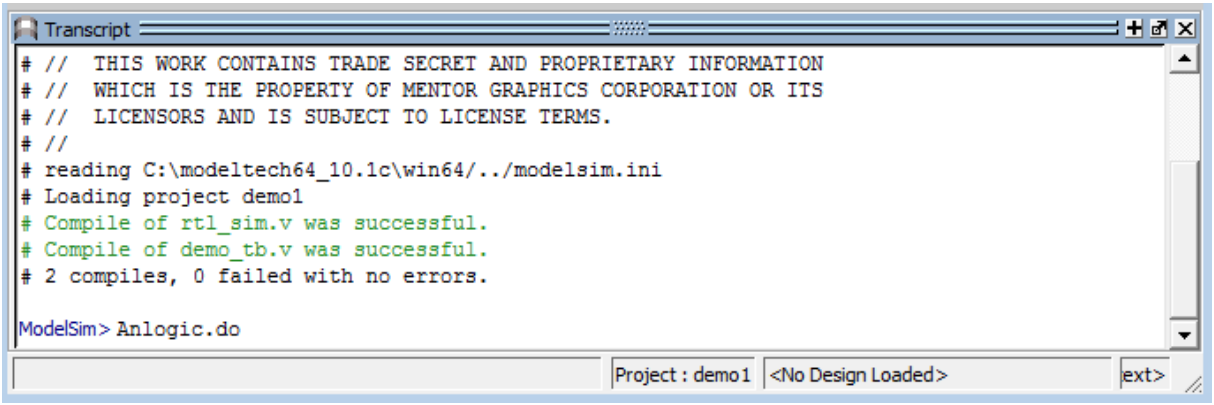
7. 仿真结束后，可在 Objects 下查看信号列表。可通过右键单击某信号，选择“Add wave”，运行完后可查看到波形的变化情况。



8. 点击 Simulate → Run → Run 100，或者在导航栏中将点击，即可运行仿真100ns。也可手动输入仿真时间。



上述过程可直接运行脚本来完成，如脚本：Anlogic.do



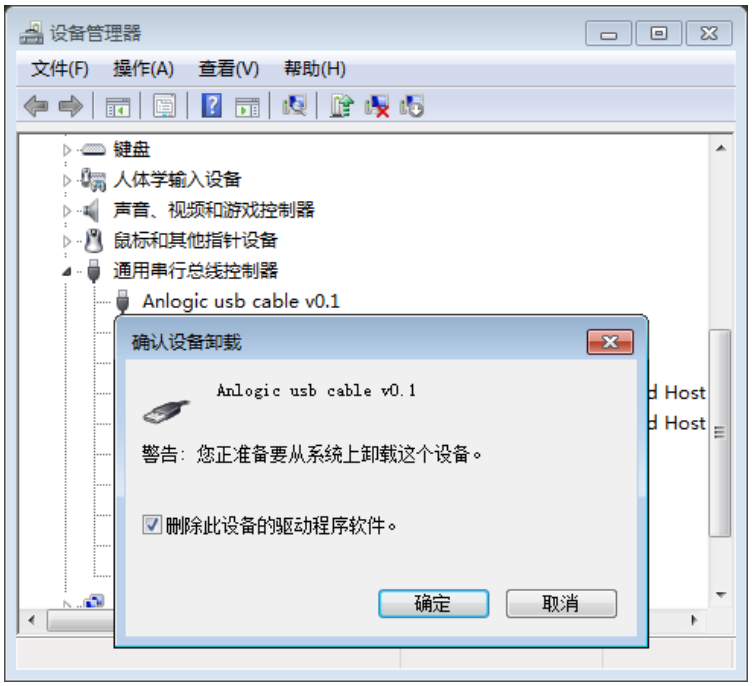
Anlogic.do 如下：

```
#
# Create work library
#
if {[file exists work]} {
   vdel -lib work –all }
vlib work
#
# Compile sources
#
vlog    "C:/Anlogic/test/work/test.v"
vlog    "C:/ Anlogic/test/work/TestBench. v"
vlog    "C:/ Anlogic/test/work/Addbit.v"
#
# Call vsim to invoke simulator
#
vsim -voptargs="+acc" -L al3_10_ver -gui work.TestBench
#
# Source the wave do file
#
add wave *
#
# Set the window types
#
view wave
view structure
view signals
run 100ns
```

9.4 USB 下载驱动安装

USB 驱动安装

TD 软件现已更新 USB 驱动,若用户此前已安装 ReadyDriverPlus 软件及 Anlogic usb cable 驱动，请先将该软件及驱动一并卸载掉，重新安装 USB 驱动。

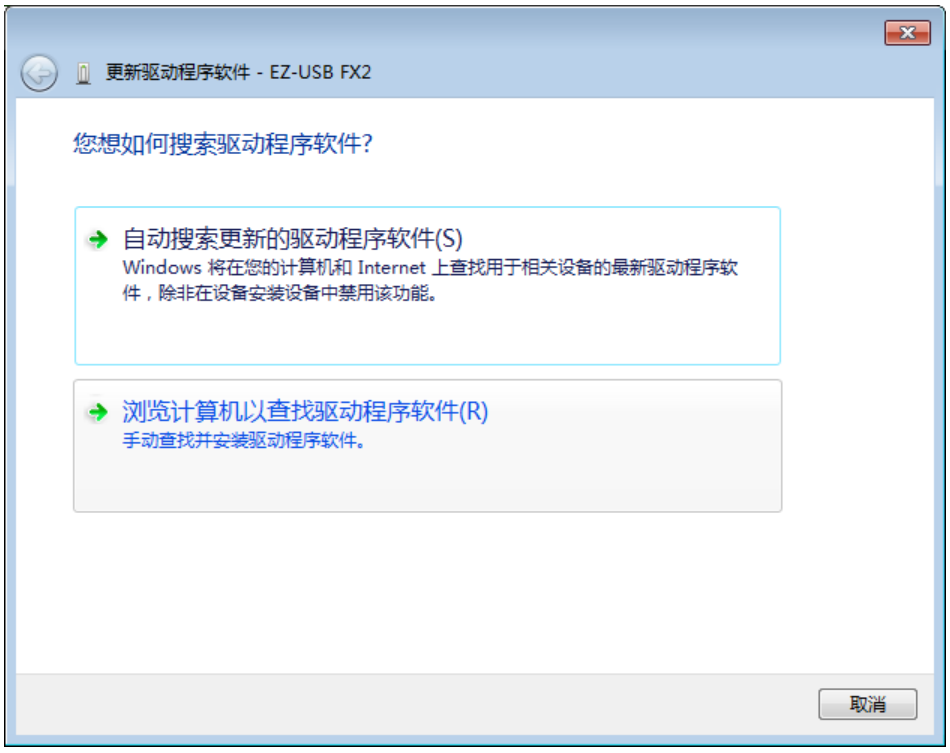


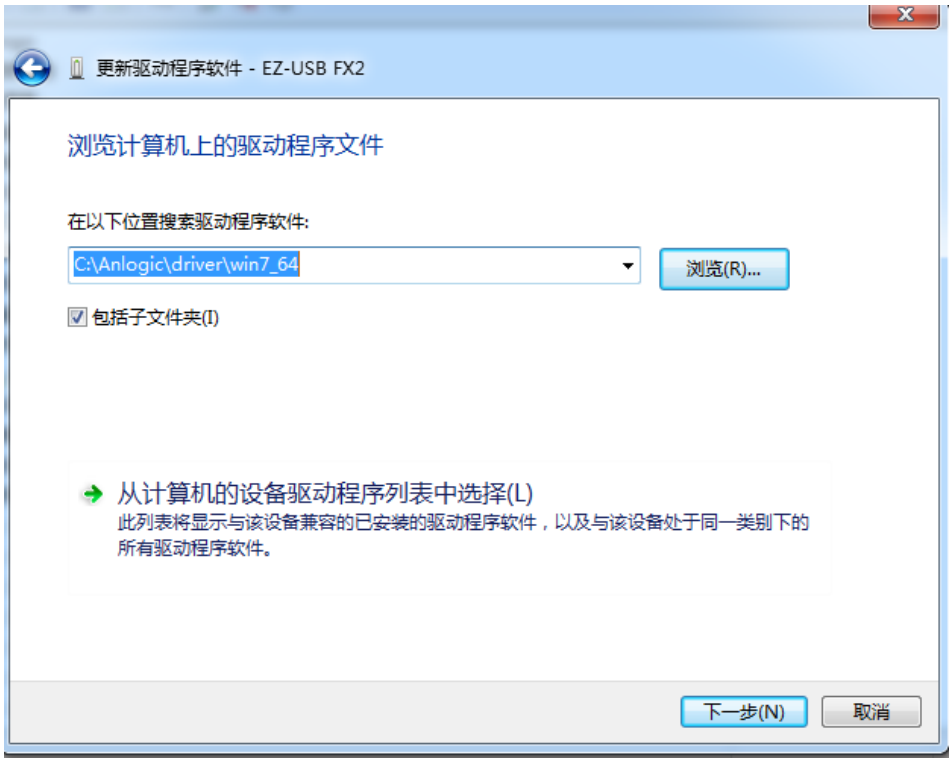
具体安装步骤如下：

1. 将下载线插入计算机，并打开设备管理器，在“其他设备”中可看到未知设备 EZ-USB FX2



2. 右键单击 EZ-USB FX2，选择“更新驱动程序软件”，将会跳出如下提示框，选择“浏览计算机以查找驱动程序软件”，并选择 TD 安装路径下的 driver





3. 根据系统选择 win7 、win8 或 win10 的 64/32 位驱动，如：

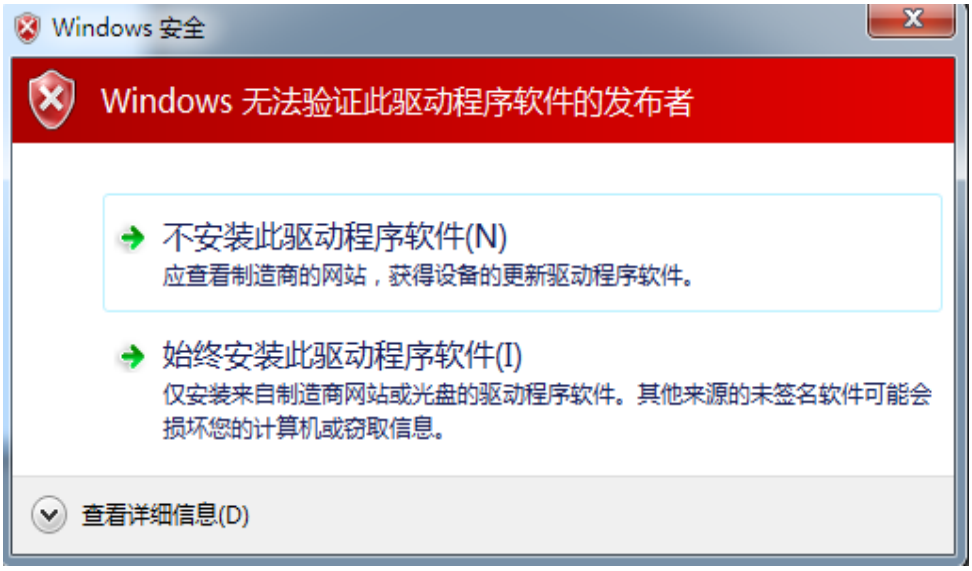
win7 32 位系统，则选择 win7_32；

win7 64 位系统，则选择 win7_64；

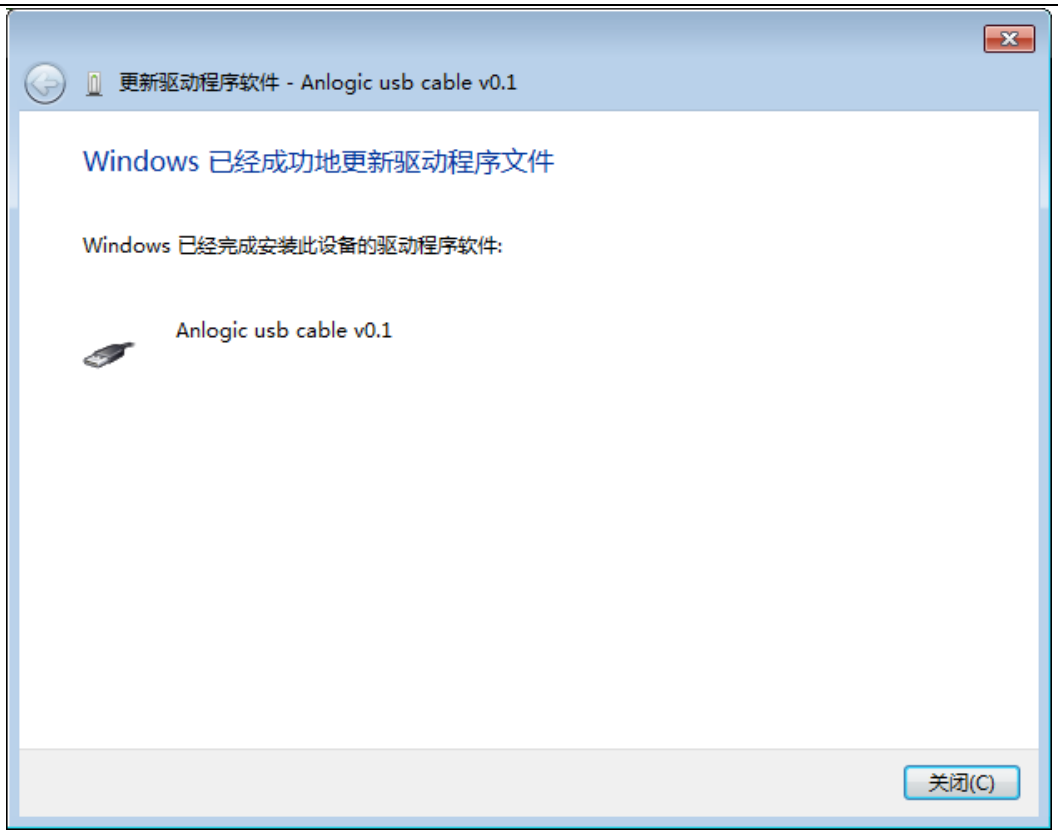
win8 或 win10 32 位系统，则选择 win8_10_32；

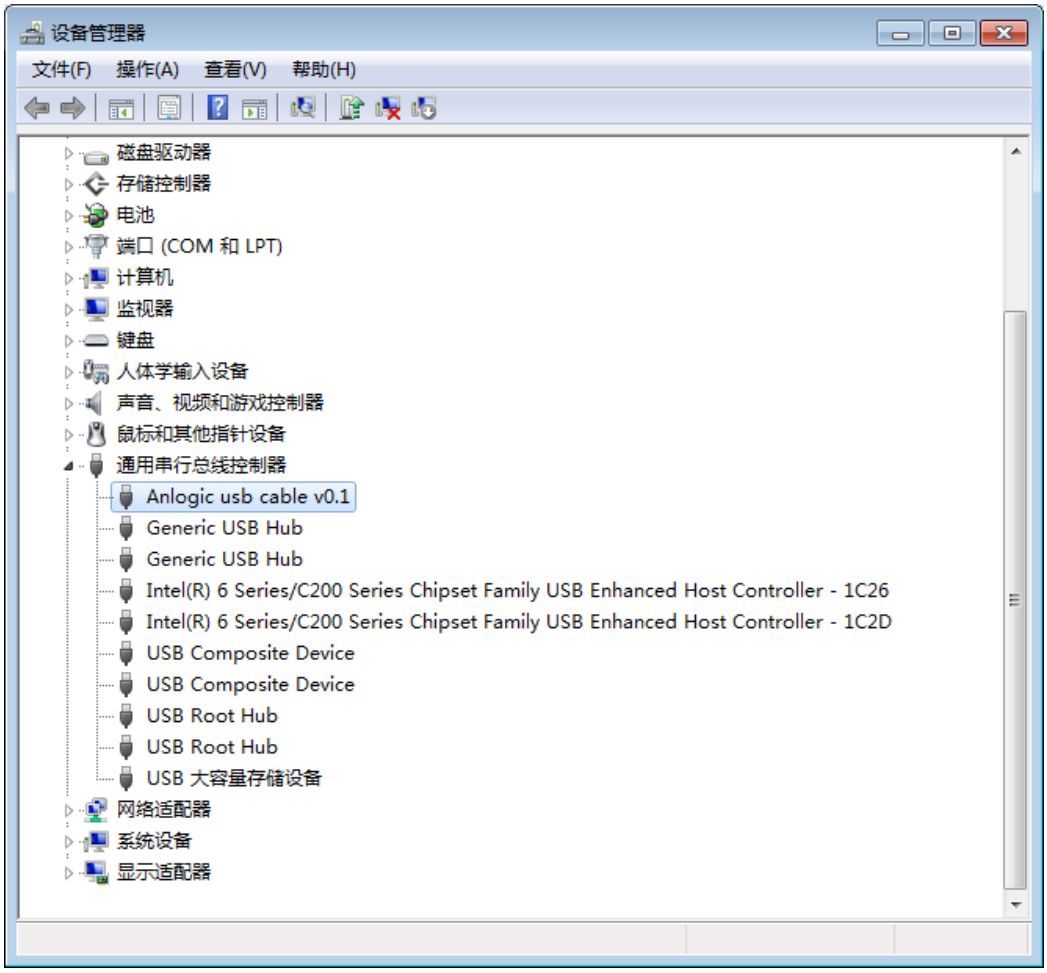
win8 或 win10 64 位系统，则选择 win8_10_64。

安装时会有警告，驱动程序未经认证，选择继续安装



4. 安装成功后打开设备管理器，展开“通用串行总线控制器”，能看到“Anlogic usb cable v0.1”，并且没有感叹号，USB 驱动安装成功





5. 驱动安装好后，重新启动计算机。

9.5 安全声明

在用户安装与使用的过程中，TD 软件不会访问任何网络数据端口，不会对软件本身进行自动更新，更不会在后台上传任何数据，一切数据信息皆保留在本地。