《FPGA 应用开发实战技巧精粹》

作者：华清远见

第 2 章 Xilinx FPGA 设计技巧

本章简介

这一章主要介绍 Xilinx FPGA 的设计技巧。通过本章学习，读者可以了解 Xilinx FPGA 的设计流程和技巧，包括 Xilinx 设计软件的使用方法，以及针对 Xilinx 器件的一些特殊技巧。读者还可以在本章中查阅 Xilinx FPGA 的设计方法和技巧。

2.1 ISE 基本使用技巧

ISE 是 Xilinx 开发的集成综合开发软件，用于设计输入、综合、布局布线、调试直到下载 FPGA 的全过程，设计者可以在 ISE 集成环境中完成所有的 FPGA 设计。本节主要介绍 ISE 的使用技巧和方法。

2.1.1 新建项目的技巧

❖ 技巧内容

使用 ISE 设计 FPGA，首先要新建一个项目，ISE 集成开发环境提供对整个工程的集成管理和开发，设计者可以在 ISE 的环境中完成所有 FPGA 设计环节。

❖ 技巧详解

新建项目的步骤如下。

（1）选择【File】/【New Project】，启动“Create New Project”对话框，如图 2.1 所示。在“Project Name”文本框中输入项目名称，在“Project Location”框中指定项目位置，在“Top-Level Source Type”下拉列表中指定顶层设计的类型，单击【Next】按钮。

（2）在“Device Properties”对话框中，选择使用的 FPGA 器件的型号，选择需要使用的综合工具和仿真工具以及使用的硬件描述语言，单击【Next】按钮。

（3）读者可以在建立好 ISE 项目以后再新建设计文件，所以这里单击【Next】按钮，直到“Project Summary”对话框出现，选择【Finish】按钮完成新建 ISE 项目。

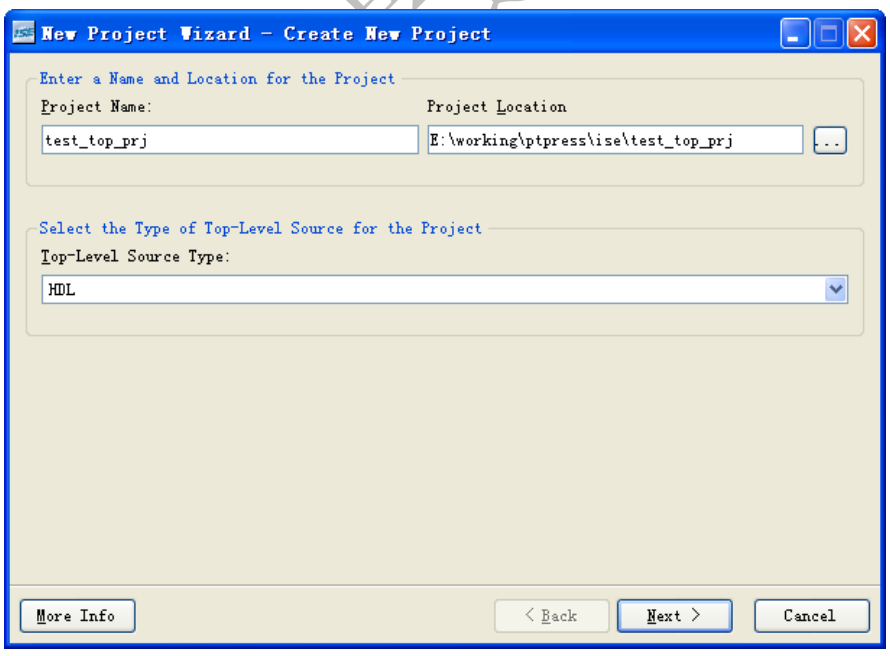

图 2.1 新建 ISE 项目

2.1.2 新建 HDL 文件的技巧

❖ 技巧内容

新建好 ISE 项目后，设计者需要新建 HDL（硬件描述语言）文件，HDL 文件是设计 FPGA 的基础。目前最流行的 HDL 语言有 VHDL 和 Verilog HDL。

ISE 中集成的 HDL 编辑器叫“HDL Editor”，它有一个“Language Templates”语法设计辅助模板，提供了 VHDL、Verilog HDL 语言和 UCF 用户约束的语法说明和例子。

❖ 技巧详解

新建 HDL 文件的方法分为以下几个步骤。

（1）启动 ISE，默认打开上次关闭的项目，选择【File】/【New】，在弹出的“New”对话框中选择“Text File”，单击【OK】按钮。

（2）接下来会打开“HDL Editor”编辑器，编写用户的 HDL 代码。

（3）输入用户代码后，选择【File】/【Save】，在弹出的对话框中输入文件名，选择保存的文件类型，单击【保存】按钮。保存后的文件会以不同的颜色显示关键字。

（4）单击【Language Templates】按钮，打开语言辅助模板，如图 2.2 所示。

（5）从左边选择模板的类型，右边窗口会显示模板的具体内容。

（6）在用户设计的 HDL 代码中，将光标定位到需要使用模板的位置，回到模板窗口，选择好需要使用的模板，单击【use template in file】按钮，将范例插入到用户的代码中，然后根据需要修改模板范例。

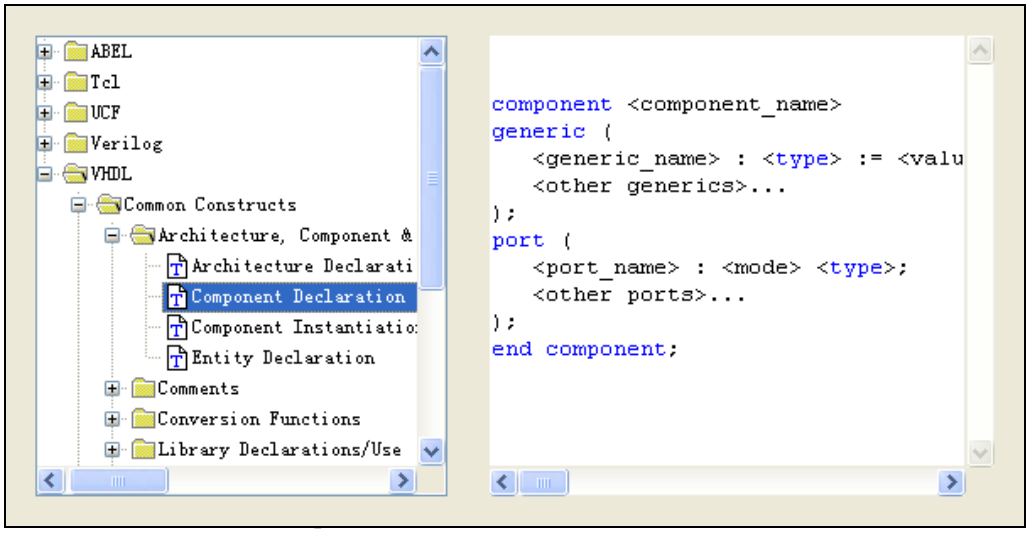

图 2.2　Language Templates 窗口

2.1.3　添加 HDL 文件的技巧

❖ 技巧内容

使用 HDL 语言进行设计的好处之一就是便于重用其他设计者的代码，所以需要在已有的工程中添加 HDL 代码。

❖ 技巧详解

添加 HDL 文件的方法分为以下几个步骤。

（1）如果只是添加文件，而不需要将文件复制到用户自己的工程中，选择【Project】/【Add Source】。

（2） 如果需要将添加的文件复制到用户自己的工程中，选择【Project】/【Add Copy of Source】。

（3）在弹出的“Add Existing Sources”对话框中，选择需要添加的文件，单击【打开】按钮，即可完成添加文件的操作。

2.1.4 新建原理图设计的技巧

❖ 技巧内容

原理图方式设计具有直观清晰的特点，几乎所有的 FPGA 设计软件都提供原理图设计输入方法，在 ISE 中集成了原理图输入工具 ECS（Engineering Capture System）。
设计者可以采用原理图方法来进行顶层的设计，而底层设计采用 HDL 代码，这样的设计结构清晰，便于设计和维护。

❖ 技巧详解

新建原理图的步骤如下。
（1）启动 ISE，默认会打开上次关闭的项目，选择【File】/【New】，在弹出的“New”对话框中选择“Schematic”，如图 2.3 所示，单击【OK】按钮。
（2）接下来会出现一个空白的原理图输入界面。
（3）在 ISE 中，该界面默认是嵌入在 ISE 集成环境中，为了获得更大的编辑空间，可以将窗口悬浮，以便更加方便地编辑原理图。单击【Float this Windows】按钮，可以将原理图编辑窗口悬浮，如图 2.4 所示。

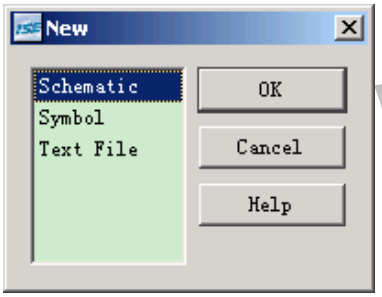

图 2.3 新建原理图对话框

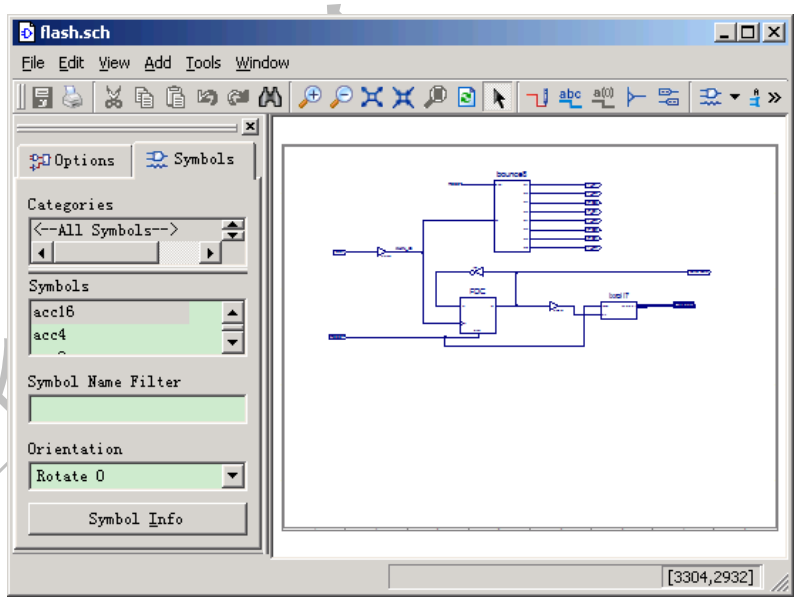

图 2.4 原理图编辑界面

2.1.5 在原理图中调用模块的技巧

❖ 技巧内容

在 ISE 中提供了很多模块供设计者使用，这些模块都是经过验证的、功能正确的设计，设计者调用这些模块可以大大加快设计。同时，设计者还可以自己设计具有特定功能的模块，以便在后续的设计中使用。

❖ 技巧详解

在原理图中调用模块的步骤如下。
（1）在原理图输入窗口的左边选择“Symbols”选项卡，如图 2.5 所示。

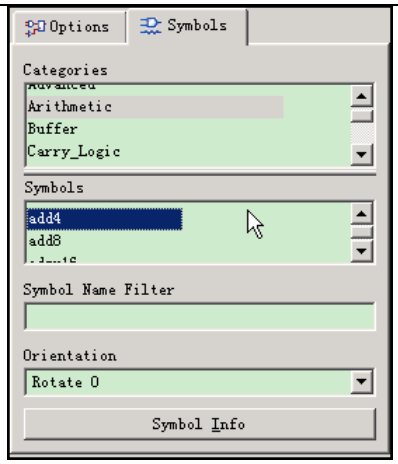

图 2.5　在原理图中调用 Symbols

（2）在“Categories”窗口中选择模块所属的类型，例如选择“Arithmetic”算术类型模块。

（3）在“Symbols”窗口中选择需要的模块，例如选择“add4”，将鼠标移动到原理图编辑窗口，则会看到一个 4 位的加法器。

（4）将模块移动到合适的位置，单击鼠标左键，放置模块。如图 2.6 所示。

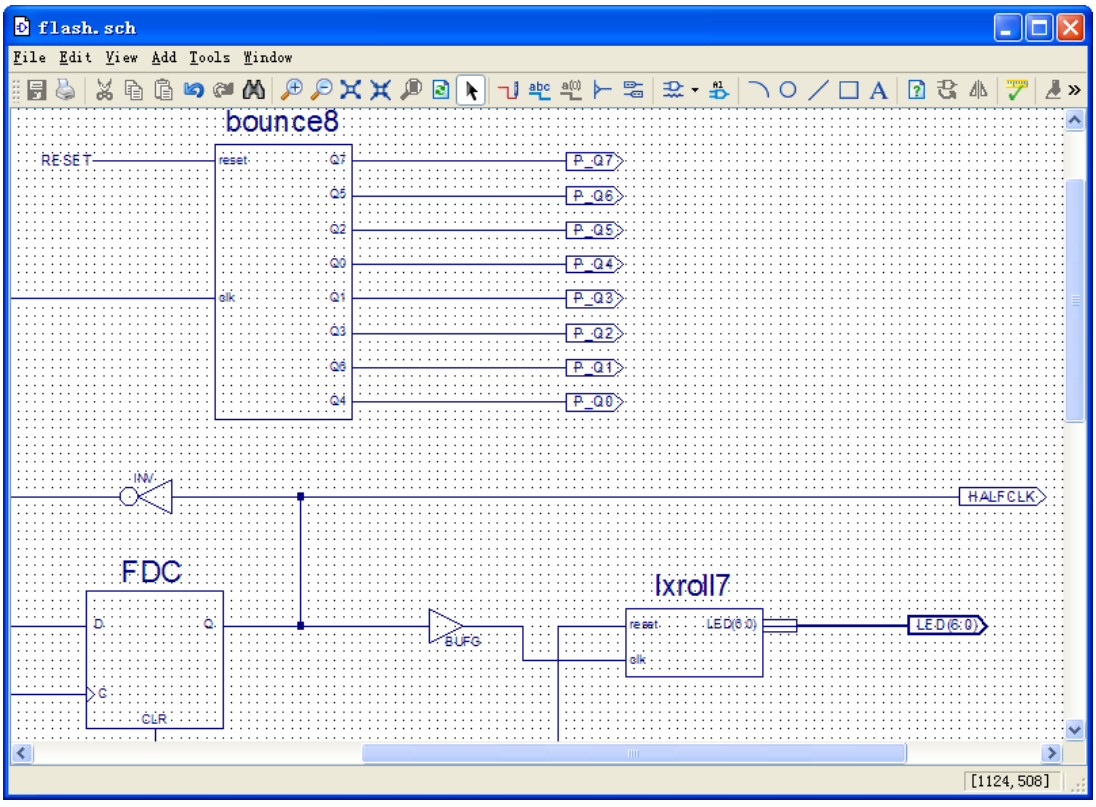

图 2.6　在原理图中放置调用的模块

2.1.6　编辑原理图的技巧

❖ 技巧内容

ISE 的原理图输入工具提供了许多实用的技巧，方便设计者快速编辑原理图，包括自动连线、快速添加端口等。

❖ 技巧详解

编辑原理图的技巧如下。

（1）首先在原理图中添加设计模块，并将模块放置到适当的位置。

（2）单击绘图工具栏中的【Add Wire】按钮，出现十字光标，移动十字光标连接原理图中的信号。

（3）单击【Add Net Name】按钮，在窗口左边“Options”选项卡下，“Name”区域中输入网络名称，如图 2.7 所示。然后单击需要命名的网络，该网络就会被命名为设计者指定的名称。

（4）完成设计后，需要添加 I/O 端口引脚。单击【Add I/O Marker】按钮，在“Options”选项卡中，指定 I/O 类型，然后在原理图中需要添加引脚的端口上，用鼠标左键拖出一个框，该框内的信号端口上会自动添加 I/O 引脚，如图 2.8 所示。

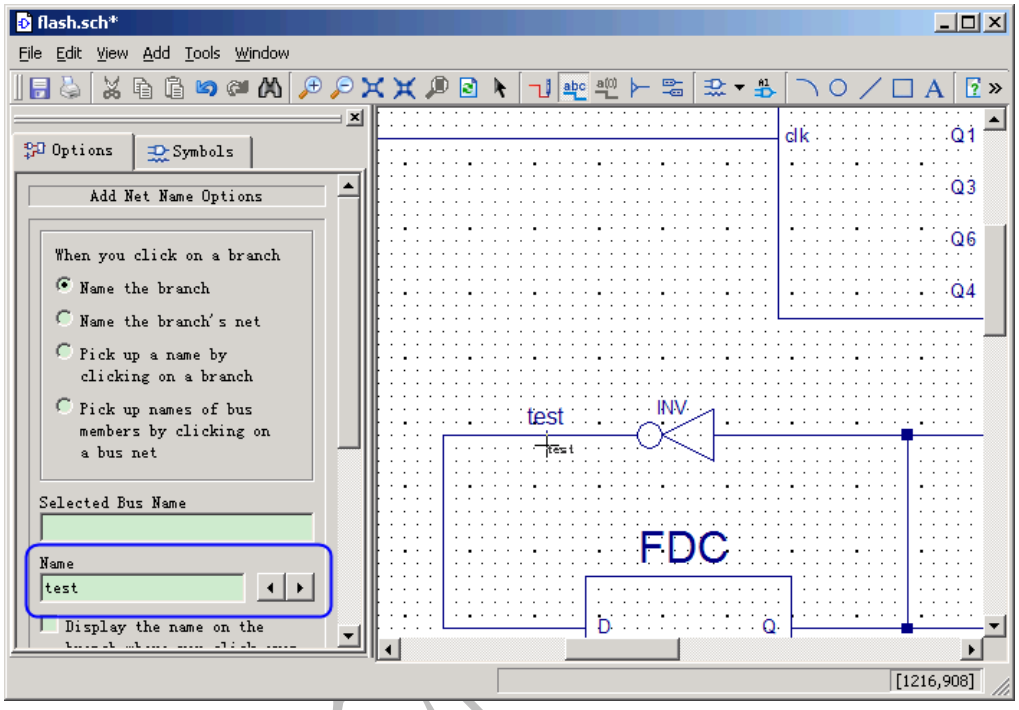

图 2.7　为信号添加网络名称

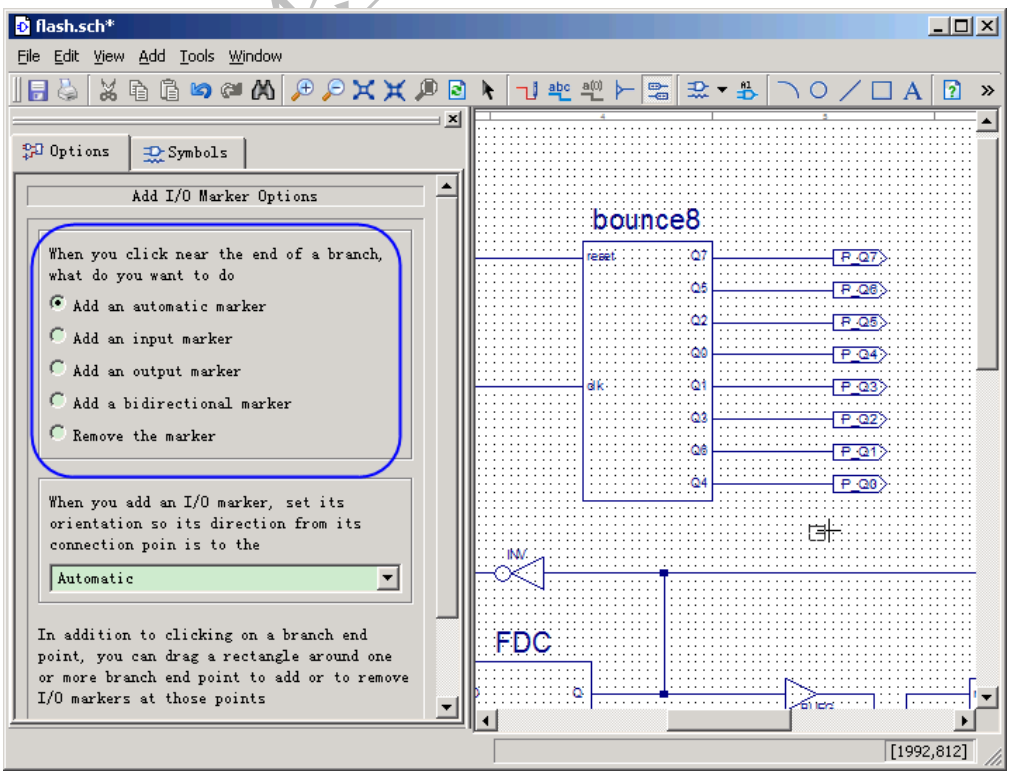

图 2.8　在原理图中添加 I/O 引脚的方法

（5）完成设计后，可以将设计生成用户器件，以便将来调用。在资源管理窗口中选择需要生成器件的文件，这里选择“flash”，在“Process for: flash”中打开“Design Utilities”，双击“Create Schematic Symbol”生成相应的用户模块。如图 2.9 所示。

（6）在原理图设计窗口的左边，选择“Symbols”选项卡，在“Categories”窗口中选择用户项目目录，然后选择用户模块，如图 2.10 所示。

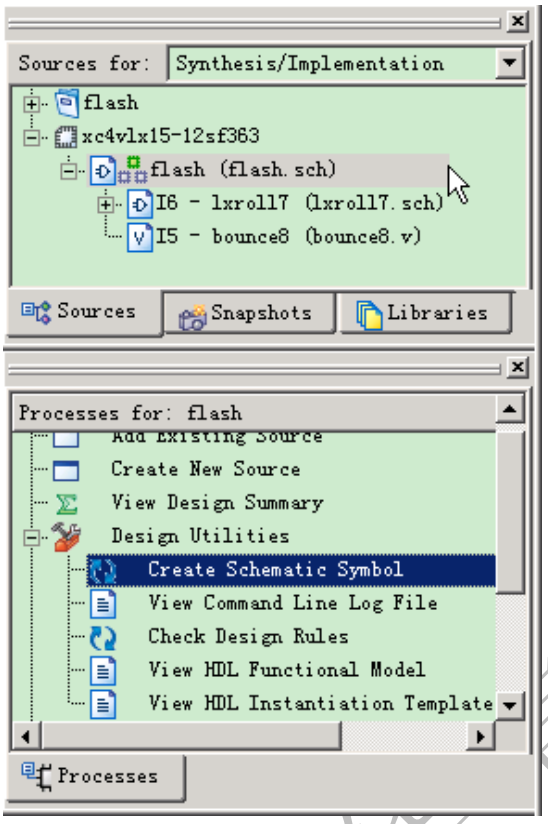

图 2.9　生成用户模块

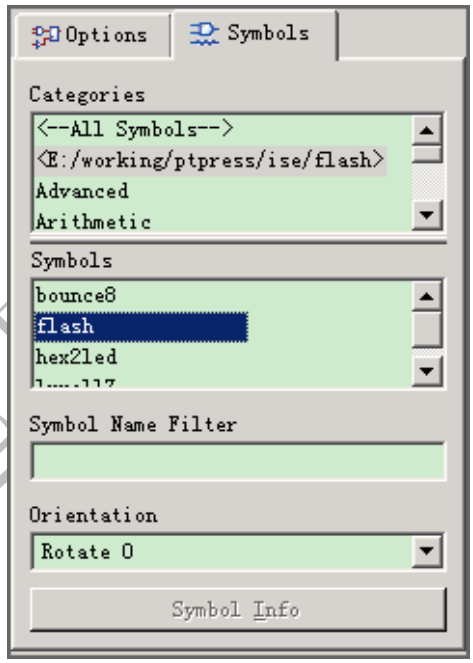

图 2.10　调用用户模块

2.1.7　用 Constraints Editor 设置约束的技巧

❖ 技巧内容

ISE 中提供“Constraints Editor”可以简单地进行时钟周期约束、输入延迟约束、端口约束和分组约束等，同时“Constraints Editor”会将约束结果自动保存到 UCF 文件中。设计者可以通过修改生成的 UCF 来完成约束，而不需要特别研究 UCF 文件的　　语法。

❖ 技巧详解

使用 Constraints Editor 设置约束的技巧如下。

（1）在 ISE 的资源管理窗口中，选择需要添加约束的顶层文件，在“Processes”窗口中，打开“User Constraints”，双击“Create Timing Constraints”，如图 2.11 所示。

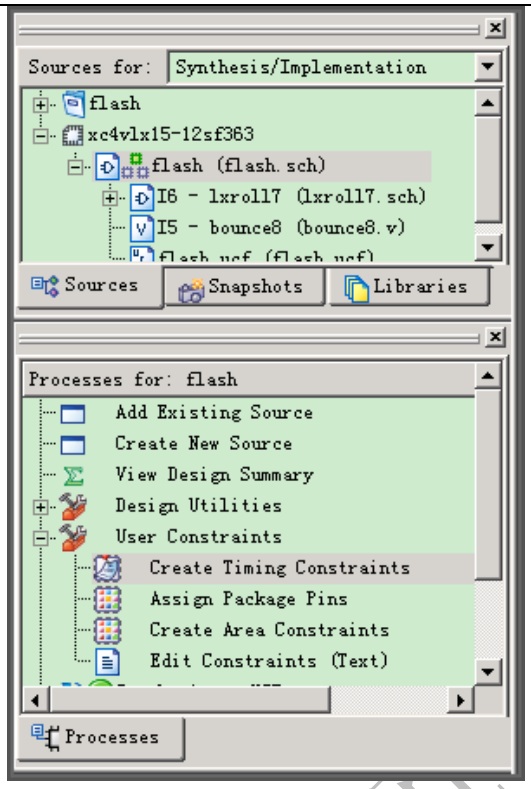

图 2.11　启动 Constraints Editor 的方法

（2）打开如图 2.12 所示界面，首先设置系统时钟周期约束。

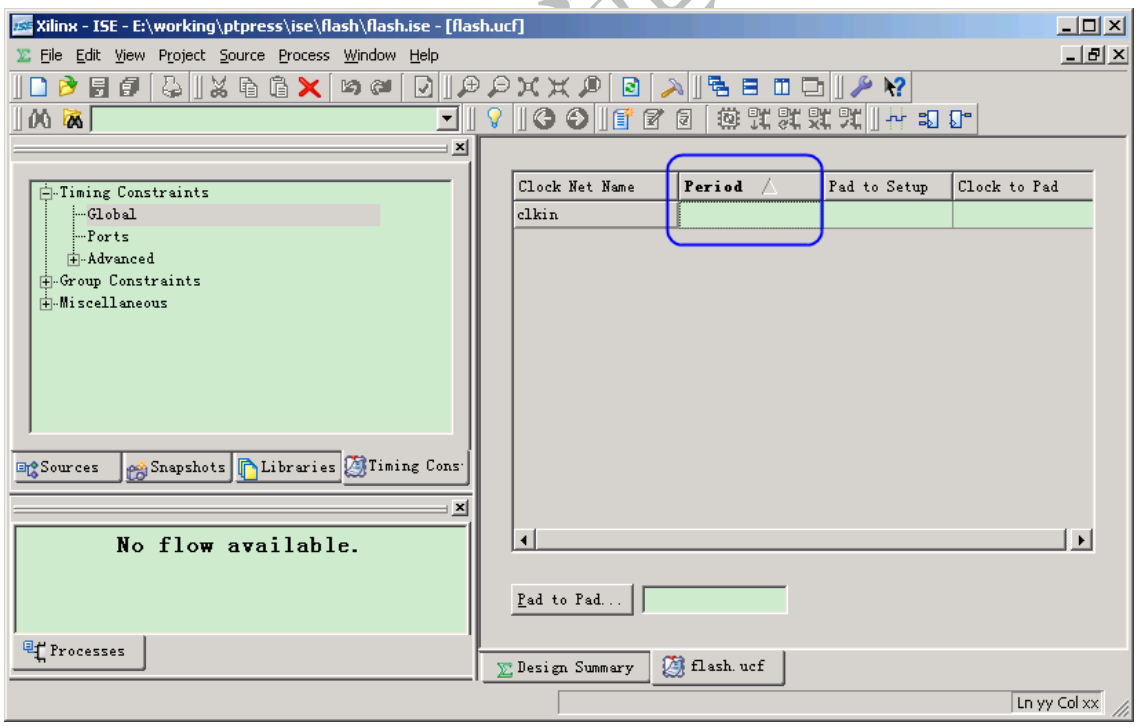

图 2.12　Constraints Editor 界面

（3）在“Constraints Editor”界面的左边打开“Timing Constraints”，单击“Global”，设置时钟周期的界面出现。双击右侧的“Period”标签下边的空白处，会弹出设置“Clock Period”的对话框。

（4）如图 2.13 所示，设置时钟周期的图形界面能够帮助设计者方便地设置约束，在最上方是图形说明各个约束的含义。这里设置时钟周期为 10ns，即 100MHz，占空比为 1：1。设置好以后，单击【OK】按钮。如果在使用中遇到问题，可以单击【Help】按钮查看帮助文档。

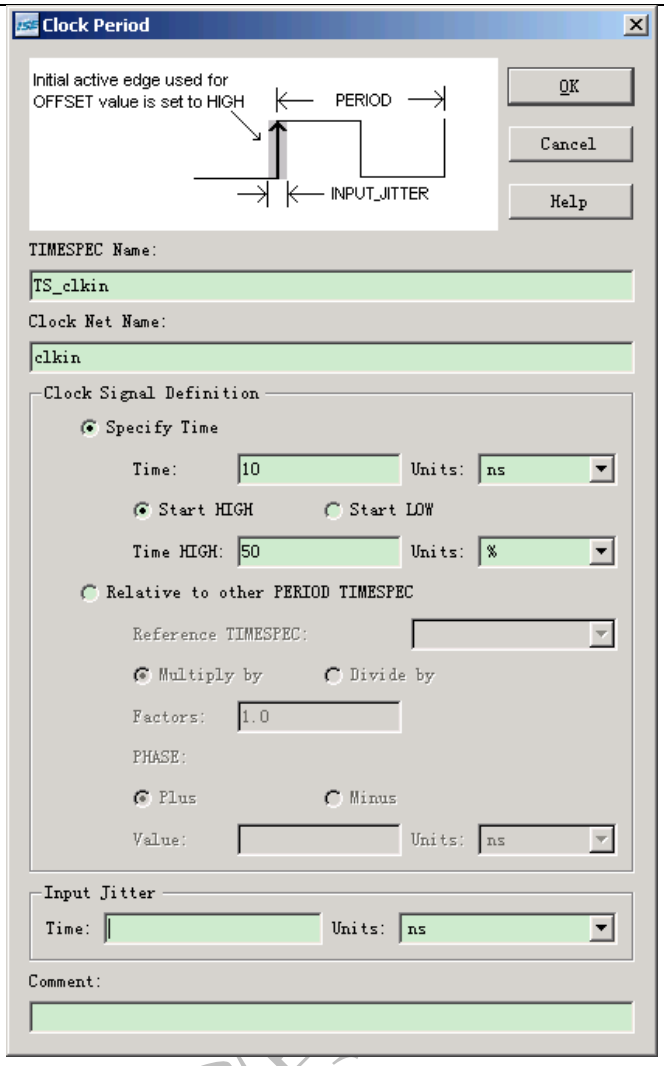

图 2.13　设置时钟周期约束对话框

（5）在图 2.12 中双击“Pad to Setup”标签下的空白处，会弹出设置建立时间的对话框，如图 2.14 所示。这里设置建立时间为 4ns，相对于 clkin 的上升沿。

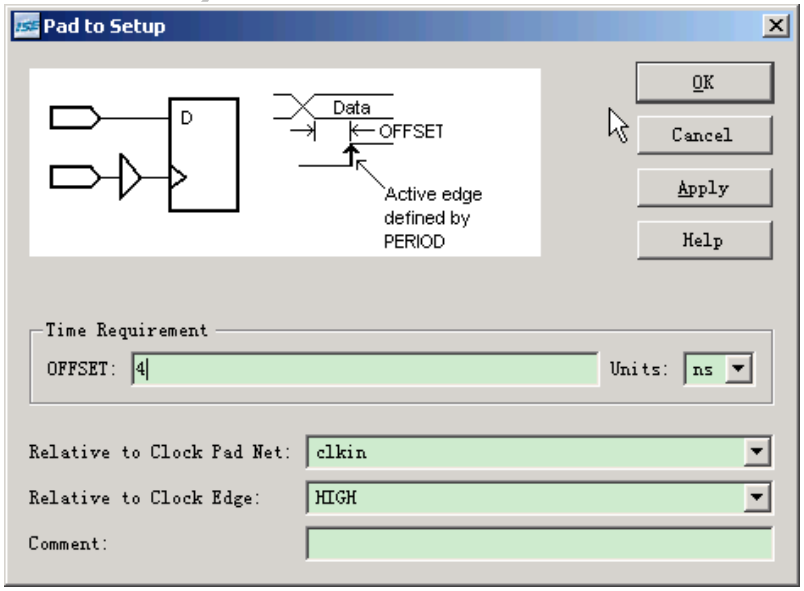

图 2.14　设置建立时间约束对话框

（6）在图 2.12 中双击“Clock to Pad”标签下的空白处，会弹出设置保持时间的对话框，如图 2.15 所示。这里设置建立时间为 2ns，相对于 clkin 的上升沿。

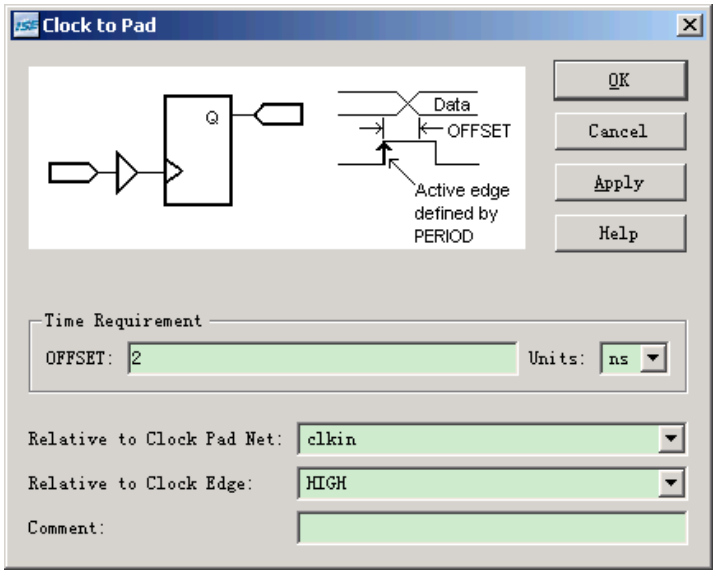

图 2.15　设置保持时间约束对话框

2.1.8　用 PACE 设置引脚与区域约束的技巧

❖ 技巧内容

ISE 提供“PACE”对引脚与区域设置约束，PACE 是一个图形化的约束设置工具。设计者可以在界面中很方便地设置引脚与区域约束。

❖ 技巧详解

使用 PACE 设置约束的技巧如下。

（1）在 ISE 中 Project Navigator 的“Source in Project”窗口中单击“Source in Project”左边的“+”号，然后双击“Assign Package Pins”或者“Create Area Constraints”，打　　开 PACE。

（2）如果项目中没有 UCF 文件，ISE 会提示用户添加 UCF 文件，也可以选择 ISE 自动生成一个 UCF 文件，如图 2.16 所示，这里选择【Yes】，ISE 会自动生成一个 UCF 文件。

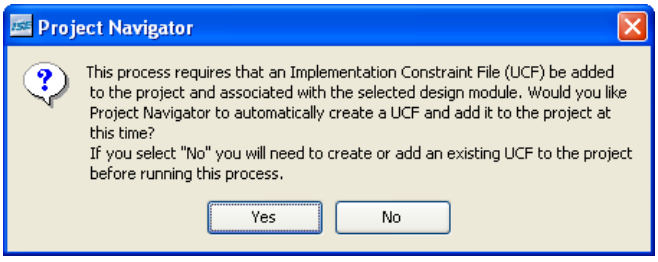

图 2.16　ISE 提示添加 UCF 文件或者生成 UCF 文件

（3）PACE 的用户界面如图 2.17 所示。

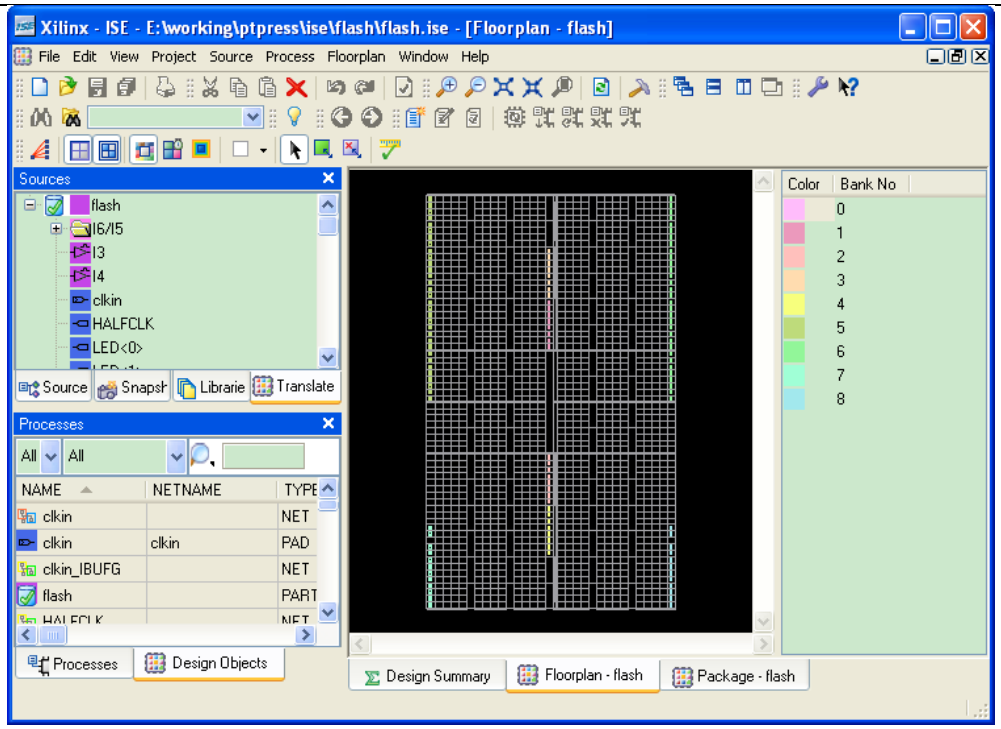

图 2.17　PACE 用户界面

（4）如果要将设计中的某一部分放置到 FPGA 中特定的位置，只需要在“Source”窗口中选中需要放置的模块，然后拖到“Floorplan”窗口中相应的位置即可，ISE 会自动计算资源使用的情况并分配区域，如图 2.18 所示。

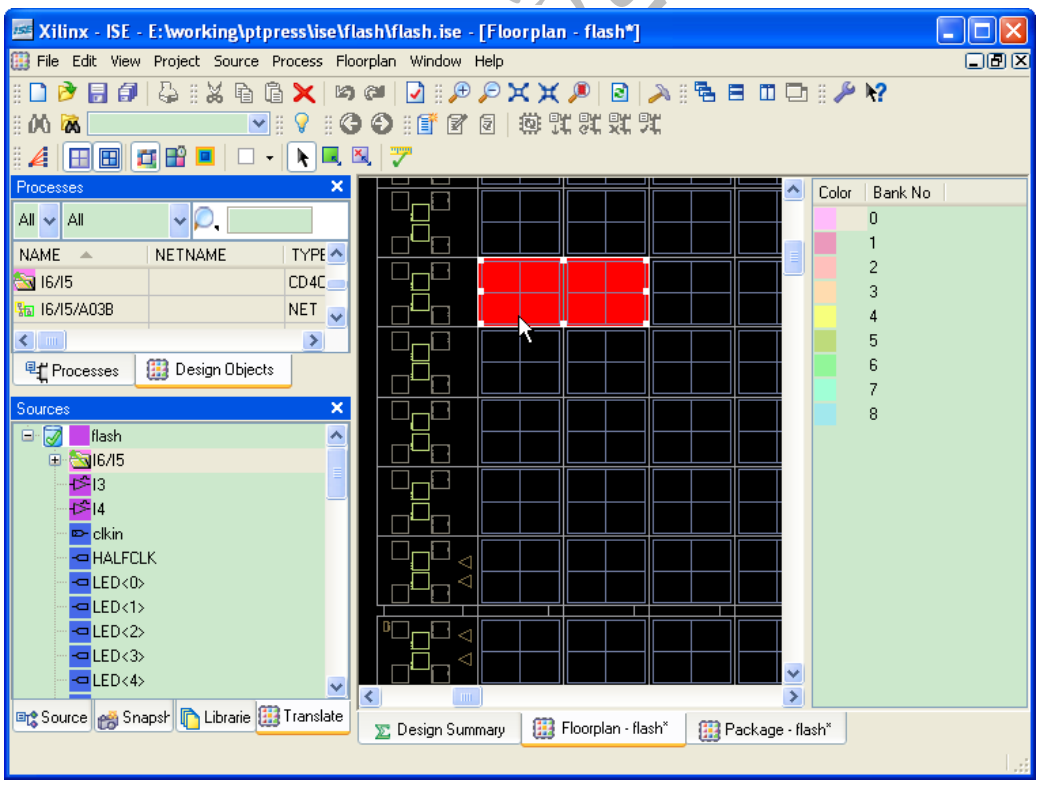

图 2.18　添加区域约束的方法

（5）单击【保存】按钮，会看到在 UCF 文件中添加了如下约束语句。表示模块“I6/I5”被放置到区域“SLICE_X3Y121：SLICE_X0Y120”这个矩形区域中。

```
INST "I6/I5" AREA_GROUP = "AG_I6/I5";
AREA_GROUP "AG_I6/I5" RANGE = SLICE_X3Y121:SLICE_X0Y120;
```

（6）如果设计中包含多个时钟，设计者就会比较关注时钟区域，单击【View Clock Regions】按钮，可以观察 FPGA 中时钟域分布情况，如图 2.19 所示。不同的颜色表示不同的时钟域。

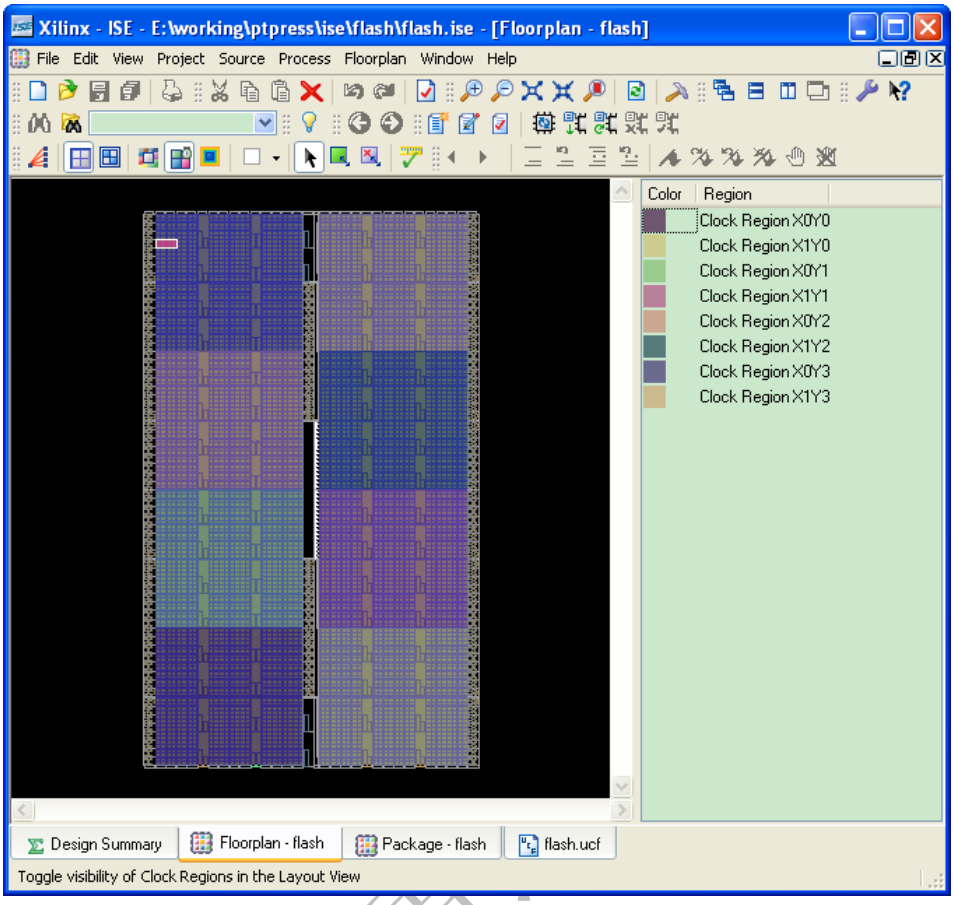

图 2.19　查看时钟域分布情况

（7）分配好位置约束后，需要对引脚进行约束。这一步很关键，因为电路板上的引脚位置已经固定，如果分配错误，电路工作就会出错。ISE 提供了很方便的引脚分配方法。

（8）单击 PACE 窗口下的“Package”选项卡，会出现引脚分配窗口，如图 2.20 所示。

（9）在“Source”窗口中标有引脚符号的信号名表示该信号为引脚，选择需要分配的引脚，将其拖放到器件结构窗口中特定的引脚上就实现了引脚位置约束，完成后单击【保存】按钮。在 UCF 文件中，会产生以下的约束语句，“LOC”表示位置约束，“=”号后面的值就是具体的引脚位置。

```
NET "clkin" LOC = B2;
NET "HALFCLK" LOC = C2;
NET "LED<0>" LOC = D2;
NET "LED<1>" LOC = E2;
NET "LED<2>" LOC = F2;
NET "LED<3>" LOC = G2;
NET "P_Q0" LOC = B3;
NET "P_Q1" LOC = C3;
NET "P_Q2" LOC = D3;
NET "P_Q3" LOC = E3;
```

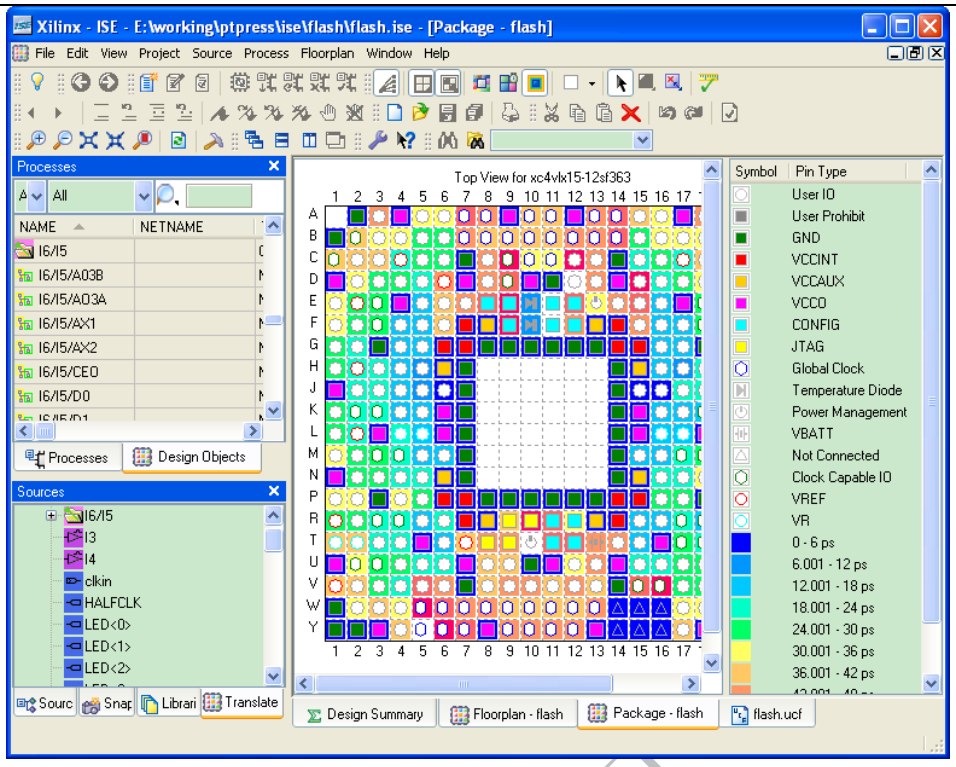

图 2.20　引脚分配界面

2.1.9　使用 XST 进行综合的技巧

❖ 技巧内容

Xilinx 在 ISE 中提供了自带的综合工具 XST（Xilinx Synthesis Technology），XST 相对于专业的综合工具而言并没有多大优势，但是对于 Xilinx 的器件而言，使用 XST 直接综合还是相当方便的，因为可以从 ISE 中直接调用 XST。

❖ 技巧详解

使用 XST 进行综合的步骤如下。

（1）在 ISE 的资源管理窗口中，选择需要综合的顶层文件。

（2）在“Process”窗口中，右键单击“Synthesize- XST”，在弹出的菜单中选择“Properties”，如图 2.21 所示。

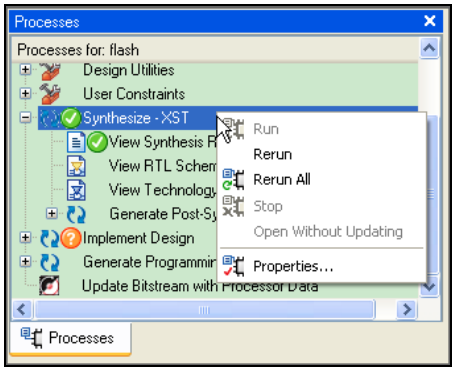

图 2.21　打开 XST 综合工具属性对话框

（3）打开“Process Properties”对话框后，可以看到在“Category”窗口中有 3 个选项：“Synthesis Options”、“HDL Options”和“Xilinx Specific Options”，如图 2.22 所示。

（4）选择“Synthesis Options”，可以看到如图 2.22 所示的界面，在这里主要需要设置的是“Optimization Goal”和“Optimization Effort”这两个选项。

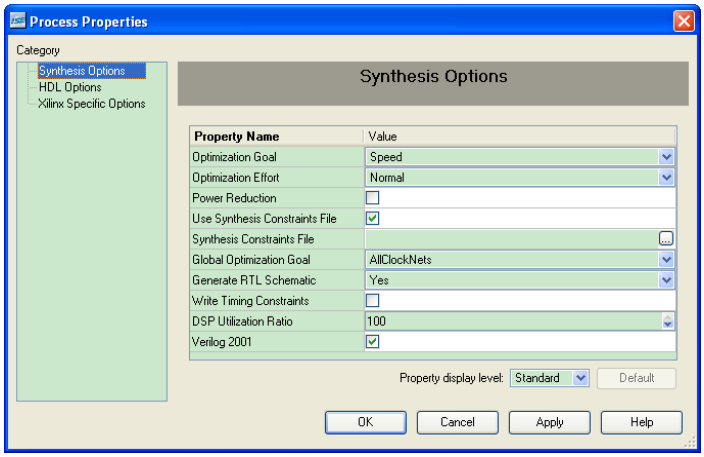

图 2.22　XST 综合属性设置对话框

- “Optimization Goal”表示的是优化目标，可以选择是以速度为优化目标，还是以面积为优化目标。
- “Optimization Effort”表示优化难度，即综合器工作的难度。如果设计者对时序或者面积约束要求较高，可以选择优化难度为“High”。

（5）设计者还可以对“HDL Options”进行设置，选择“Category”中的“HDL Options”，出现如图 2.23 所示的界面。

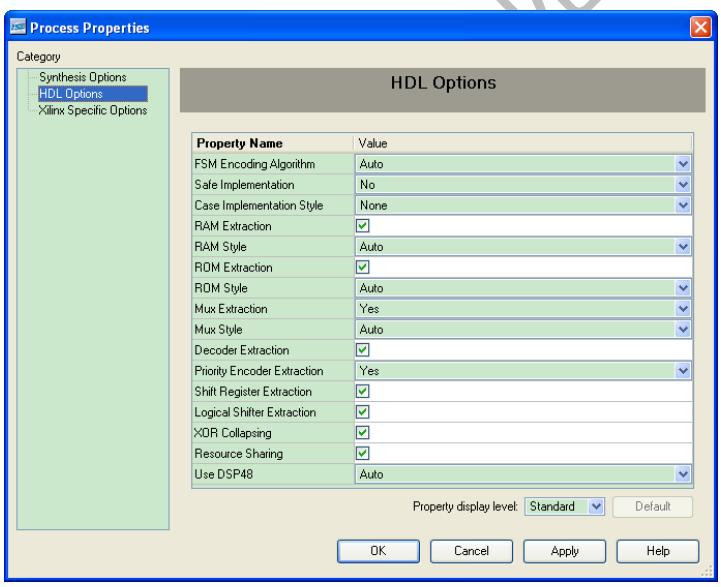

图 2.23　设置 XST 综合工具的 HDL 属性

（6）在“HDL Options”中，设计者需要设置与 HDL 语言编写规则和编译方式相关的属性，其中有几个常用的属性需要设计者注意。

- “FSM Encoding Algorithm”表示状态机的编码方式，包括了 One-Hot 编码、格雷码等。设计者可以根据设计要求选择，对于初学者，可以选择“Auto”让 ISE 自动选择编码方式。
- “RAM Style”表示 RAM 的类型，可以选择使用“Block RAM”或者“Distribute RAM”。如果设计中使用的 RAM 对时序要求不高，而且 RAM 容量也比较小，可以选择“Distribute RAM”，以节约 FPGA 中“Block RAM”资源，反之选择“Block RAM”。
- “Resource Sharing”表示是否允许 XST 综合工具复用一些逻辑模块。因为在设计中有的模块是重复的，如果能够按照时分复用的方式来进行综合，可以大大降低资源的　消耗。

（7）　在“Category”窗口中的最后一个选项是设置 Xilinx 专用参数。设置好以后单击【OK】按钮。

（8）双击“Process”窗口中的“Synthesize-XST”，开始执行综合操作。完成后，会在“Transcript”窗口中显示报告信息，设计者可以根据这些信息对设计进行分析。

（9）如图 2.24 所示，在完成综合以后，展开“Synthesize-XST”之前的“+”号，可以对综合结果进行分析。

- View RTL Schematic：双击该选项，可以查看 RTL 级的综合结果，该结果应该与设计相同。
- View Technology Schematic：双击该选项，可以看到和器件相关的综合结果，该结果表示了设计被映射到 FPGA 中的结果。
- Generate Post-Synthesis Simulation Model：双击该选项，可以生成综合后仿真的 HDL 文件，在这个 HDL 文件中包含了器件的延迟信息，可以调用这个 HDL 文件来进行后仿真，以验证时序的正确性。

以查看 RTL 综合结果为例，双击“View RTL Schematic”，ISE 会打开 RTL 查看窗口，该窗口中可以看到综合后生成的顶层接口，如图 2.25 所示。

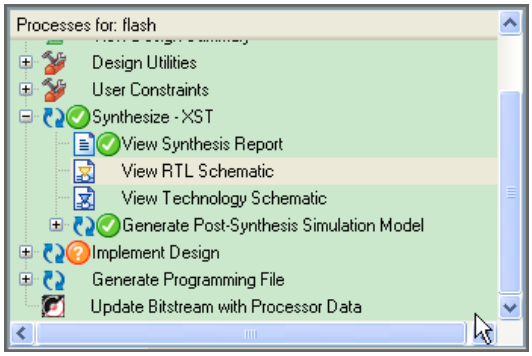

图 2.24　查看综合结果

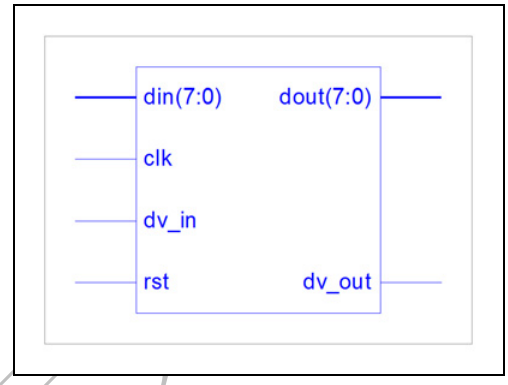

图 2.25　查看 XST 综合 RTL 级结果

为了查看详细内容，可以双击该顶层模块，或者单击【PUSH】按钮或者【Forward a Schematic】按钮查看详情，如图 2.26 所示。如果要回到上一级图形，单击【Pop】按钮或者【Back a Schematic】按钮即可。

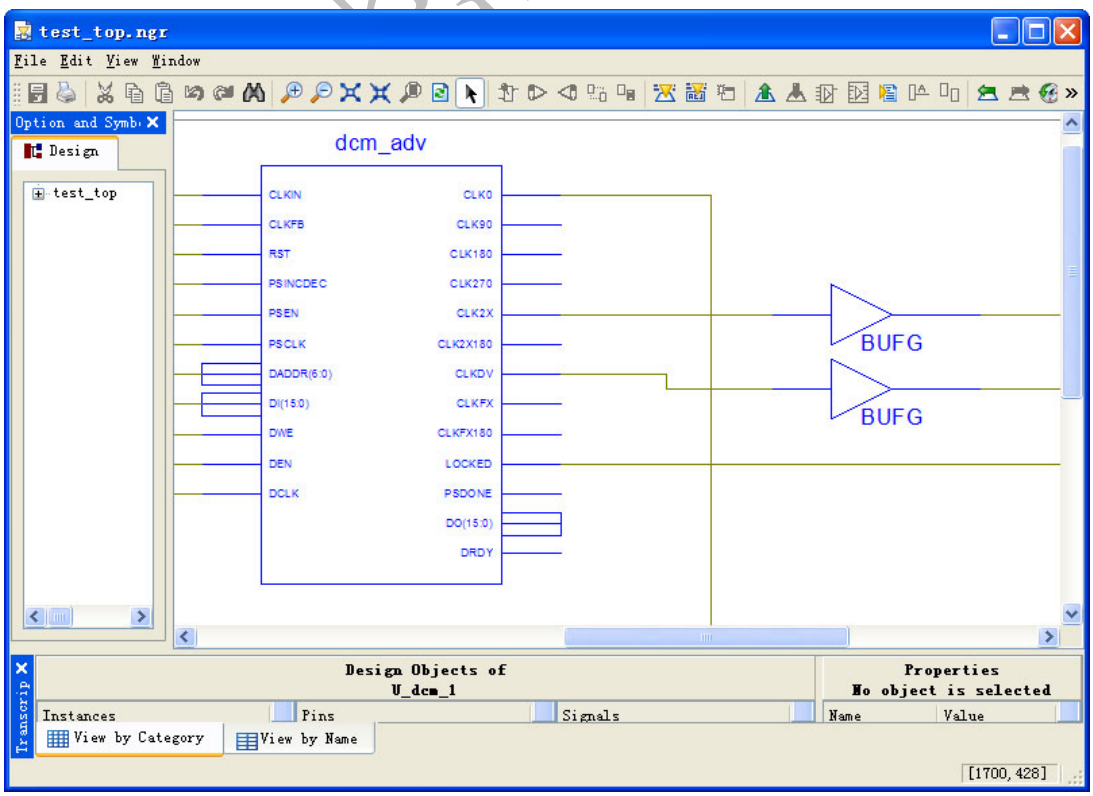

图 2.26　XST 综合的 RTL 详细结果

2.1.10 设计实现的技巧

❖ 技巧内容

完成综合以后，需要将设计映射到 FPGA 中，对应于 ISE 中的操作就是“Implement Design”，完成后会生成用于生成下载文件的 NCD 文件。

ISE 中设计实现分为 3 步，分别是：Translate、MAP 和 Place & Route。

- Translate：读取综合生成 EDIF 和 NGC 格式的网表文件，生成 Xilinx 的 NGD 格式的文件。NGD 文件是用逻辑元件表示的网表，包括触发器、逻辑门和 RAM 等逻辑 元件。
- MAP： 输入文件是 Translate 生成的 NGD 格式的网表文件，输出为 NCD（Native Circuit Description）格式的文件。MAP 的功能是将 NGD 文件中的逻辑元件映射成 Xilinx FPGA 中的元件，比如 Logic Cell、I/O cells 或者是 Block RAM 等。
- Place & Route：接收 MAP 生成的 NCD 文件，将各个元件放置到 FPGA 中适当的位置，并通过布线器连接各个元件，完成在 FPGA 中的设计实现。Place&Route 输出 NCD 文件，用于生成下载文件。

❖ 技巧详解

在 ISE 中完成设计实现非常简单，只需要做简单的设置即可，步骤如下。

（1）在 ISE 的资源管理窗口下“Process”窗口中，右键单击“Implement Design”，弹出“Process Properties”对话框。

（2）单击“Process Properties”对话框中的“Translate Properties”，设置 Translate 属性，如图 2.27 所示。如果没有出现如图 2.27 所示的界面，单击“Property display level”旁的下拉菜单，选择“Advanced”。需要强调的是，如果设计中调用了其他设计，或者需要使用其他 IP 的网表文件进行布局布线，则需要在“Macro Search Path”中指定存放这些网表文件的地址。

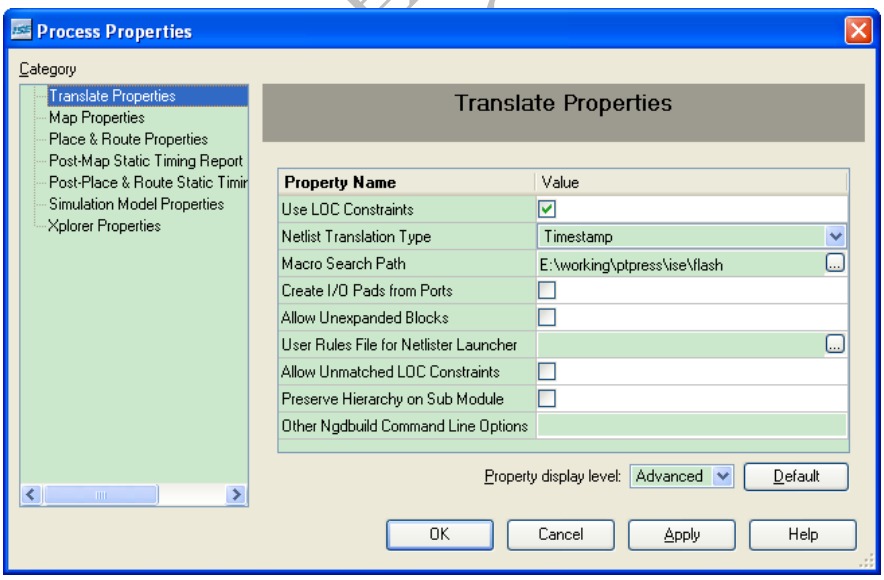

图 2.27 设置 Translate 属性

（3）依次设置好其他的参数，单击【OK】按钮。

（4）在“Process”窗口中双击“Implement Design”即可完成 Translate、MAP 和 Place & Route 的全部操作，设计者也可以单独执行各个步骤。

2.1.11 生成下载文件的技巧

❖ 技巧内容

Xilinx 的下载文件有两种主要的类型：后缀名为 BIT 的文件，该类文件可直接下载到 FPGA；后缀名为 MCS 的文件，该类文件可下载到 PROM 中，上电后 PROM 中的内容会配置 FPGA。

❖ 技巧详解

生成下载文件的步骤如下。

（1）生成 BIT 文件的方法很简单，在 ISE 资源管理器窗口的“Process”窗口中，双击“Generate Programming File”就会在 ISE 工程目录下生成和顶层设计同名的 BIT 文件。

（2）为了生成 MCS 文件在“Process”窗口中，展开“Generate Programming File”，双击“Generate PROM，ACE，or JTAG File”，如图 2.28 所示。

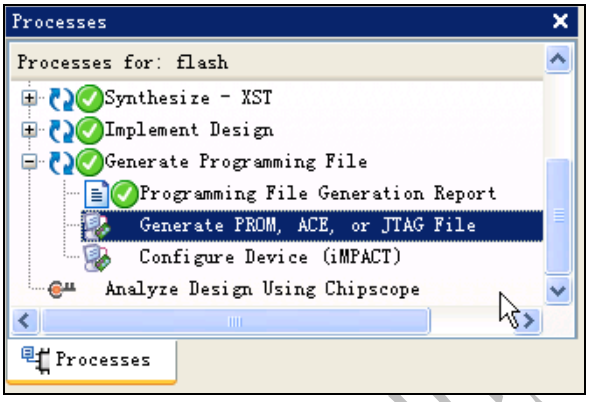

图 2.28　启动生成 MCS 文件

（3）在弹出的“Welcome to iMPACT”窗口中，选择“Prepare a PROM File”，单击【Next】按钮。

（4）弹出“Prepare PROM Files”对话框，选择目标器件为“Xilinx PROM”，PROM 文件类型选择 MCS，指定 PROM 文件的文件名和存储位置，如图 2.29 所示，单击【Next】按钮。

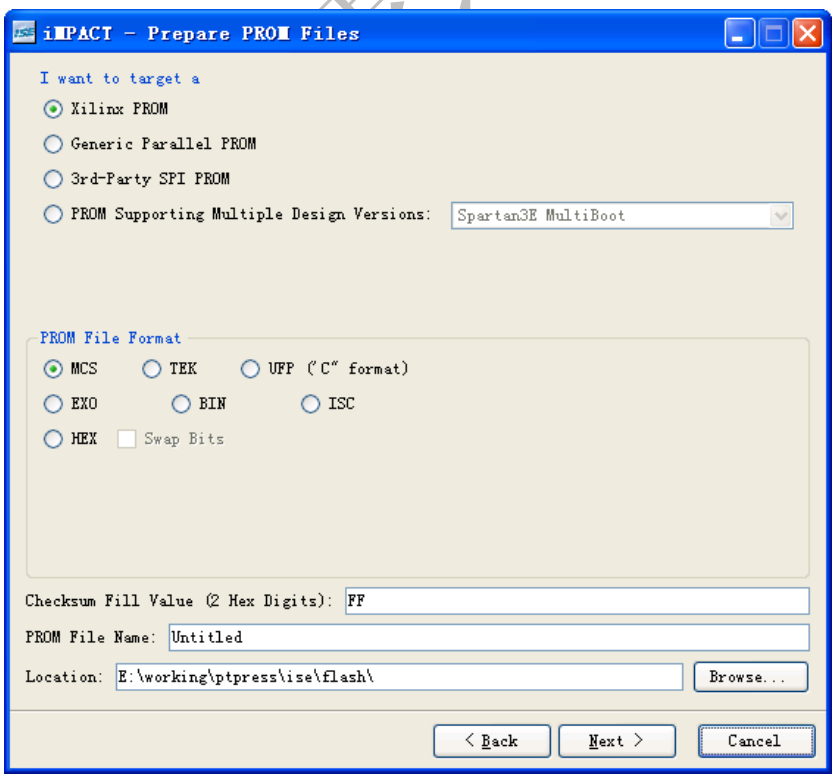

图 2.29　生成 MCS 文件设置对话框

（5）在指定 PROM 器件对话框中，选择 PROM 类型和型号，如果 PROM 器件容量不够大，可以选择“Enable Compression”，如图 2.30 所示。如压缩后还不能放到 PROM 器件中，就需要考虑更换 PROM 器件了。

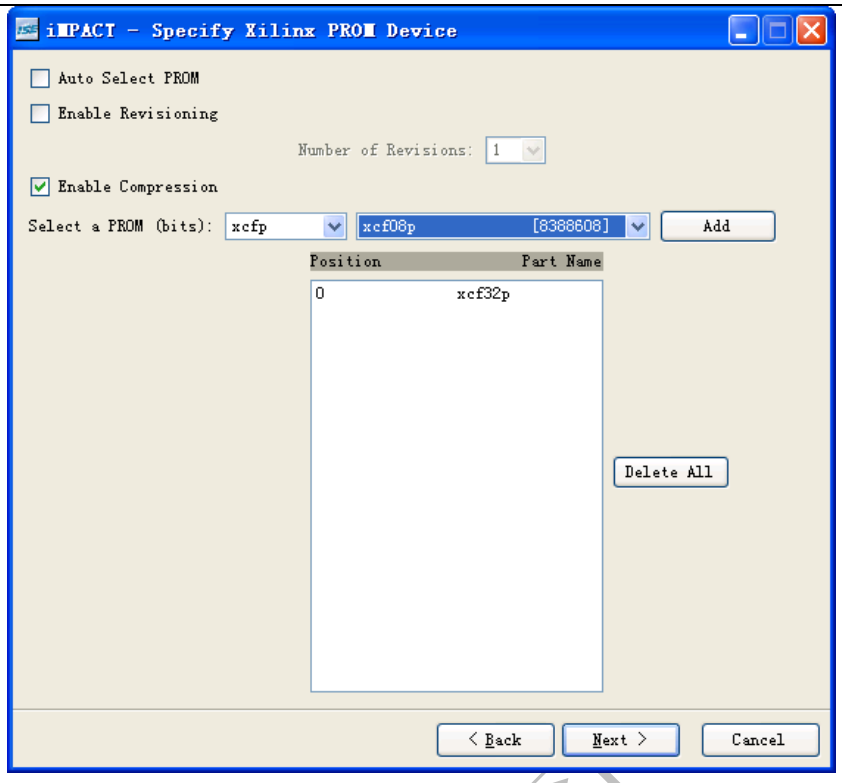

图 2.30　为 MCS 文件选择器件

（6）单击【Next】按钮，再单击【Finish】，会提示添加生成 MCS 文件的 BIT 文件，单击【确定】按钮。

（7）添加好 BIT 文件后，会出现图 2.31 所示的界面。图中可以看到 PROM 器件的使用率以及关联的 BIT 文件。在“Processes”窗口中双击“Generate File”生成 MCS 文件。

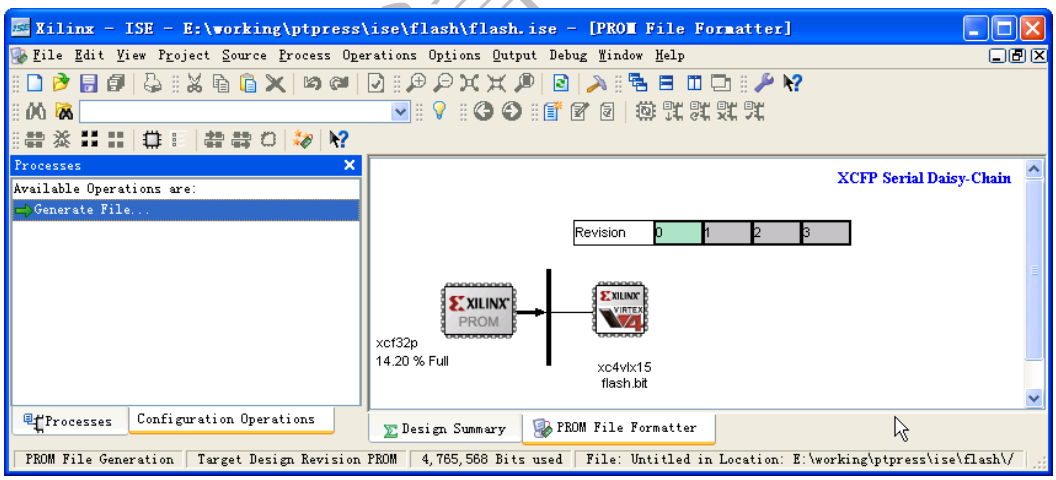

图 2.31　为 BIT 文件生成 MCS 文件

（8）成功后，会提示“PROM File Generation Succeeded”。

2.1.12　下载 FPGA 的技巧

❖ 技巧内容

生成下载文件后，需要下载到 FPGA，以便验证其功能的正确性。下载 FPGA 有两种方式：一种是直接通过 JTAG 口将 BIT 文件下载到 FPGA，下载完成后 FPGA 就开始执行 BIT 文件定义的功能；另外一种是将 MCS 文件下载到 PROM，这样每次上电以后，PROM 就会配置 FPGA。

❖ 技巧详解

设计者可以通过两种方式配置 FPGA，步骤如下。

（1）将 JTAG 下载线连接到电路板上，打开电源。

（2）最直接的方法就是在 ISE 的 Project Navigator 的“Process”窗口中，双击“Configure Device (iMPACT)”，如图 2.32 所示。

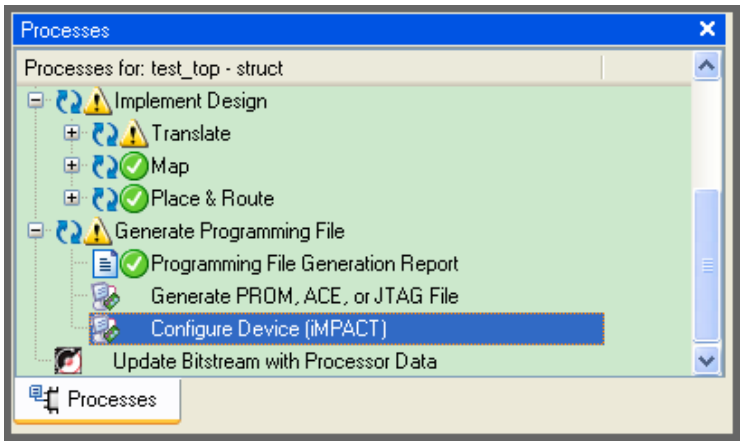

图 2.32　在 ISE 中启动 FPGA 配置程序

（3）弹出如图 2.33 所示的 iMPACT 欢迎界面，选择“Configure devices using Boundary-Scan（JTAG）”，在下拉菜单中选择自动连接，单击【Finish】按钮。

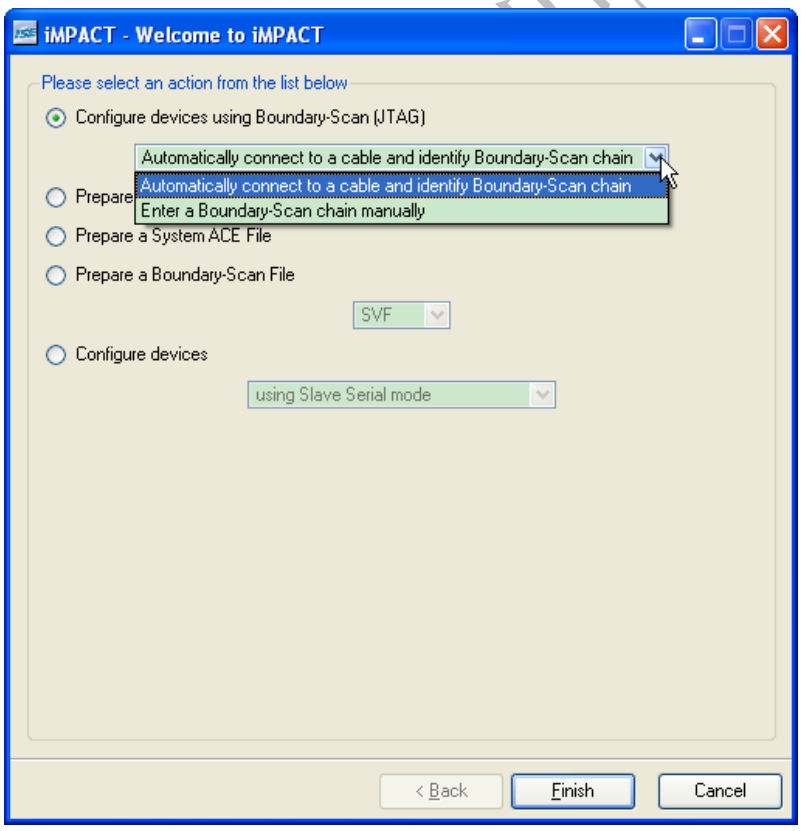

图 2.33　iMPACT 欢迎界面

（4）接下来 ISE 会自动扫描连接在 JTAG 链上的器件，根据设计者的电路板设计的不同，会出现如图 2.34 所示的界面。

ISE 会要求设计者为连接在 JTAG 链上的器件指定配置文件，如果没有配置文件，可以单击【Bypass】按钮，跳到下一个器件。

这里是型号为“xc5vlx50t”的 BIT 文件，然后单击【Open】按钮，现在已经准备好配置 FPGA 了。

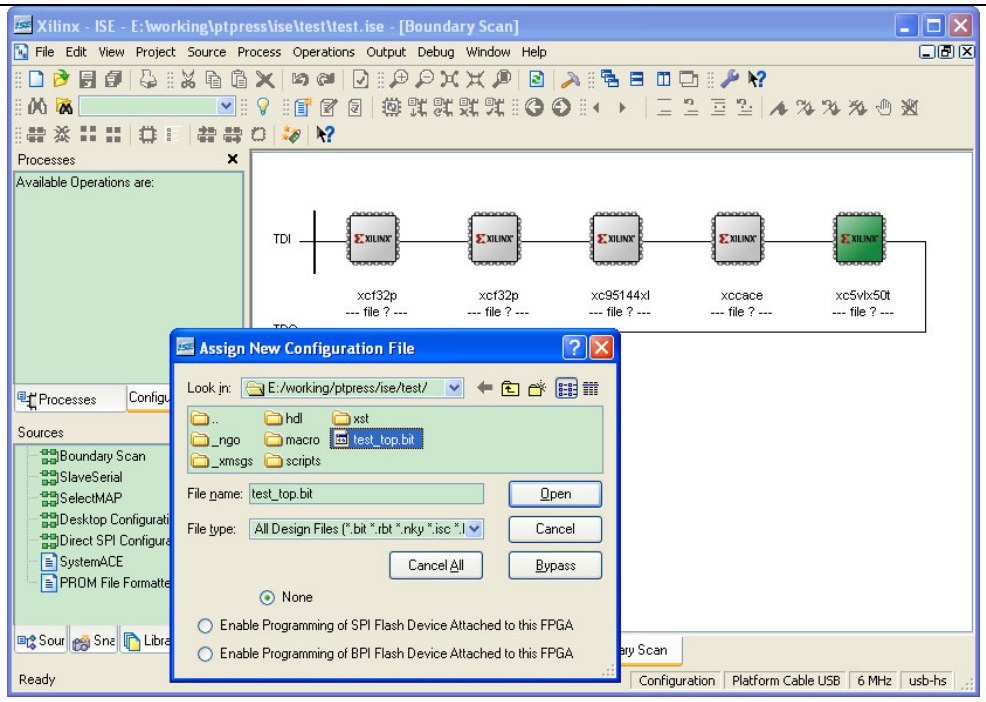

图 2.34　指定配置文件

（5）指定好配置文件后，在需要配置的器件上右键单击，在弹出的菜单中选择“Program”。
如果是配置 FPGA，在弹出的对话框中选中“Verify”，然后单击【OK】按钮。
如果是配置 PROM，在弹出的对话框中选中“Erase Before Programming”，然后单击【OK】按钮，不要选中“Verify”。
完成后会提示配置成功。

2.2　仿真验证技巧

在 ISE 中集成了测试激励生成器（HDL Bencher），在没有专业仿真工具的情况下，可以使用测试激励生成器产生激励信号，并调用 ISE 集成的仿真工具完成功能验证。

2.2.1　新建测试平台的技巧

❖ 技巧内容

使用测试平台，用户可以方便地对设计进行仿真。ISE 中可以方便地生成测试激励的文件模板，用户只需要在模板中添加激励代码。

❖ 技巧详解

新建测试平台的方法如下。
（1）在 ISE 的 Project Navigator 中单击【Project】/【New Source】，在弹出的“Select Source Type”中选择“Testbench WaveForm”，输入文件名，单击【Next】。
（2）选定需要进行仿真的文件，如图 2.35 所示，单击【Next】按钮，然后单击【Finish】按钮。

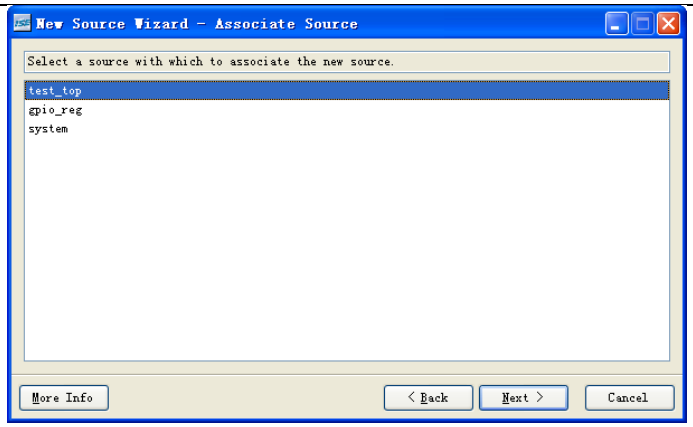

图 2.35　指定 Testbench 需要仿真的文件

（3）在弹出的“Initialize Timing”窗口中，对时钟进行设置，可以指定时钟频率、占空比、指定时钟信号。如图 2.36 所示，测试激励生成器使用图形化界面，设计者从图形界面就可以知道设置的内容，非常方便。

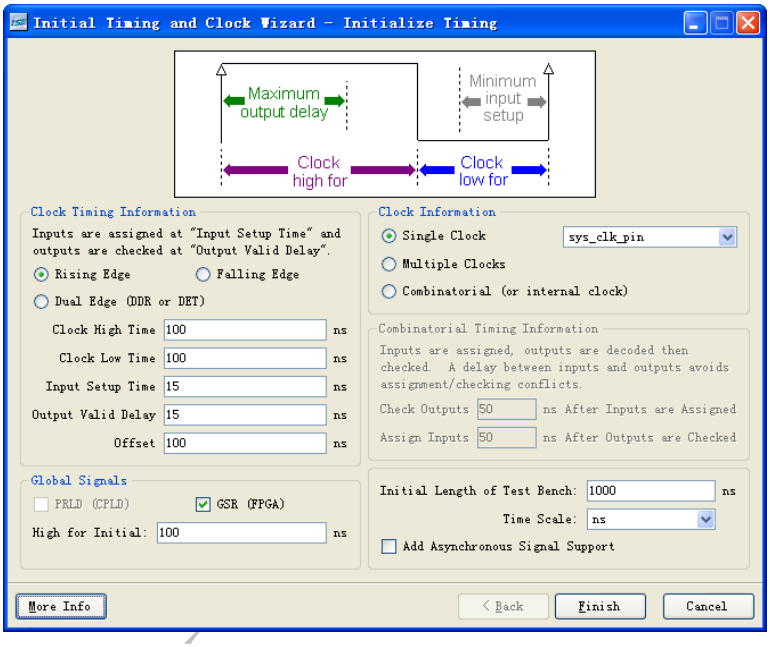

图 2.36　测试激励生成器初始化时钟

（4）设置好时钟后，单击【Finish】按钮完成，可以看到在 Project Navigator 中添加了测试平台的顶层文件，如图 2.37 所示。双击该文件，打开可以看到已经例化了需要仿真的模块，并且定义好了内部信号，只需要添加用户激励即可进行仿真。

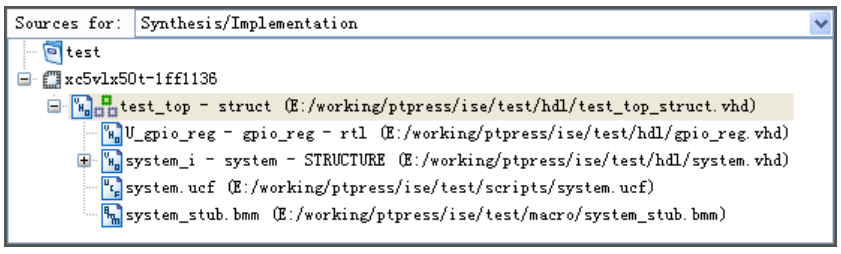

图 2.37　Project Navigator 中的测试平台顶层文件

2.2.2　图形化编辑激励信号的技巧

❖ 技巧内容

用户建立好测试平台后，ISE 中会自动出现一个后缀名为 TBW 的窗口，这是波形编辑器的窗口，可以编辑输入信号。

❖ 技巧详解

波形编辑器窗口如图 2.38 所示，使用方法非常简单，在需要变化信号的地方单击鼠标左键，信号值就会翻转。按照这个方法，可以设置激励的值。还可以在信号上单击鼠标右键，在弹出的菜单中选择“Set Value”设置信号值。

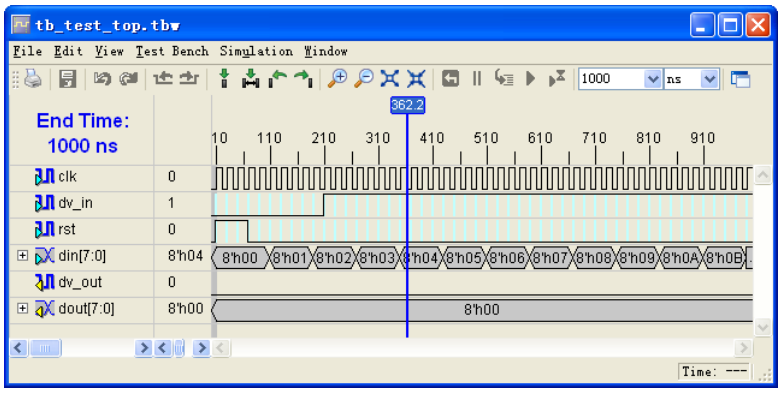

图 2.38　波形编辑器窗口

使用波形向导编辑测试激励波形是一种很方便的方法。双击信号，出现“Set Value”窗口，如图 2.39 所示。单击【Pattern Wizard】按钮，出现“Pattern Wizard”窗口。

波形向导有选择波形、波形说明、进制选择和参数设置等项目，如图 2.40 所示。

- Pattern Type 用于选择生成的波形的类型，这里可以选择脉冲、随机信号、翻转信号。选择不同的类型，窗口中对应的波形图形会变化，可以根据图形了解所选波形的特点。
- Number of Cycles 表示波形持续的周期。
- Radix 表示信号显示的进制，可以是二进制、十进制和十六进制。
- Pattern Parameters 是波形的参数，根据 Pattern Type 不同，参数也会不同。中间的图形详细说明了参数的具体信号，设计者可以根据图形来进行设置。

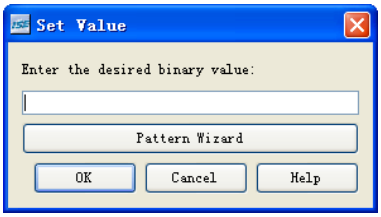

图 2.39　波形编辑器 Set Value 窗口

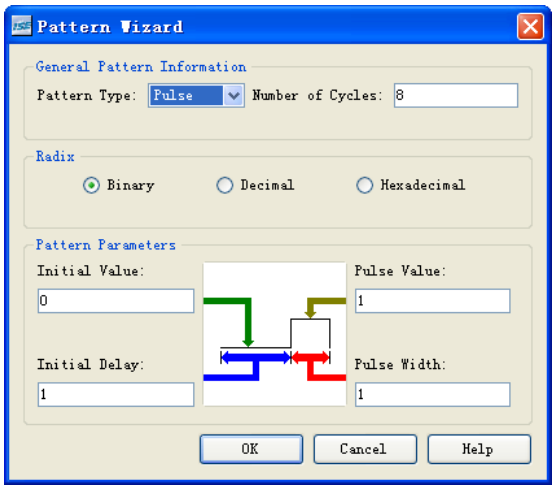

图 2.40　波形编辑器 PatternWizard 窗口

2.2.3　在 ISE 中仿真的技巧

❖ 技巧内容

用户建立好测试平台后，ISE 中会自动出现一个后缀名为 TBW 的窗口，这是波形编辑器的窗口，可以在这里编辑输入信号。编辑好激励波形后，就可以仿真了。

❖ 技巧详解

在 ISE 中仿真的具体步骤如下。

（1）新建 ISE 项目的时候需要选择仿真器为“ISE Simulator”，如图 2.41 所示。

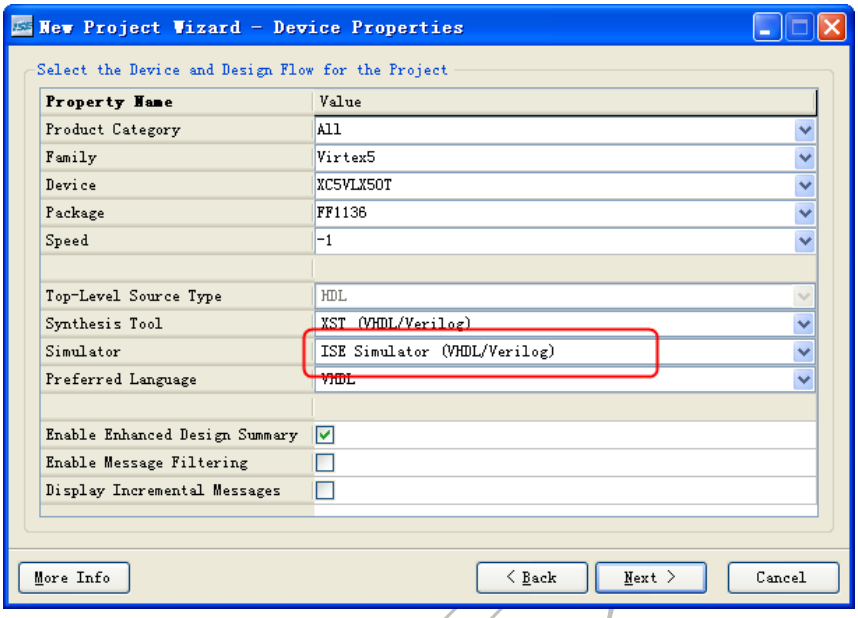

图 2.41　指定 ISE 项目的仿真器

（2）编辑好测试激励波形后，在“Project Navigator”的“Sources for”下拉菜单中选择“Behavioral Simulation”，选择编辑的波形文件（后缀名为 TBW），在“Processes”窗口中选择“Simulate Behavioral Model”，如图 2.42 所示。

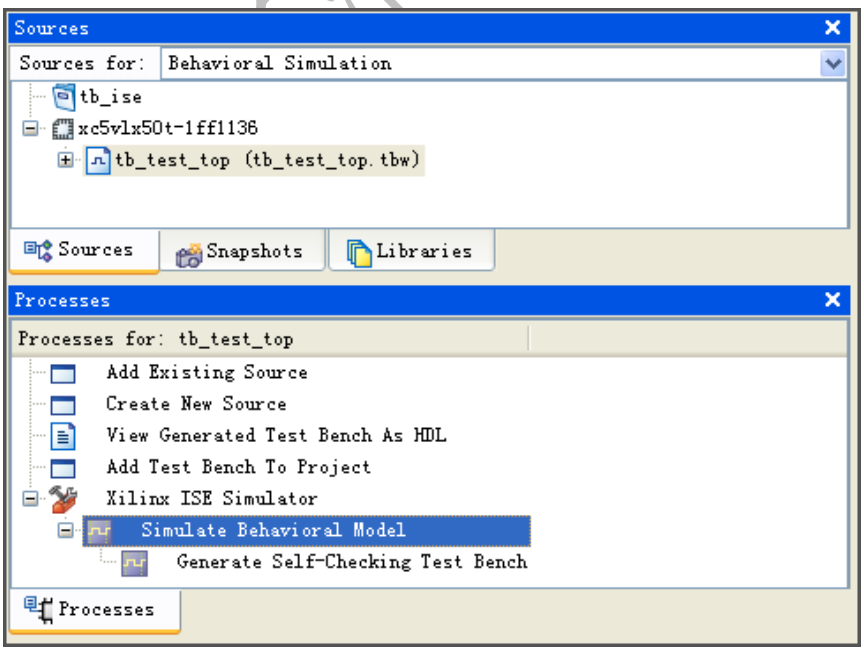

图 2.42　启动 ISE 仿真器

（3）在弹出的“Simulation”窗口中，可以进行仿真。在“Simulation”菜单下，有执行仿真的操作，包括运行仿真、运行一段时间、单步执行等。仿真的结果如图 2.43 所示。

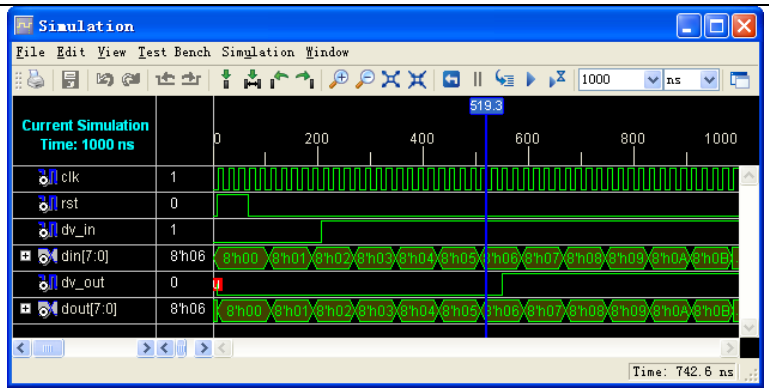

图 2.43　ISE 仿真器运行结果

2.2.4　在 ISE 中调用 ModelSim 的技巧

❖ 技巧内容

ModelSim 是目前最流行的仿真工具，在 ISE 集成环境中有 ModelSim 的接口，可以调用 ModelSim 进行仿真。

❖ 技巧详解

在 ISE 中调用 ModelSim 的方法如下。

（1）启动 ISE 集成环境，打开已有的项目，建立好测试平台。

（2）在资源管理器窗口中“Source for”下拉菜单中选择“Behavioral Simulation”，可以看到测试平台文件出现在“Source”窗口中，如图 2.44 所示。

同时在“Processes”窗口中会出现对应的操作，可以从这里启动 ModelSim。如图 2.45 所示，双击“Simulate Behavioral Model”即可调用 ModelSim 进行行为仿真。

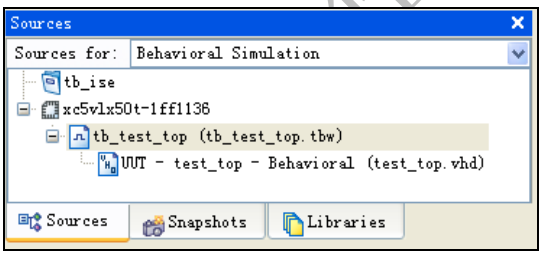

图 2.44　在资源管理器中指定仿真平台

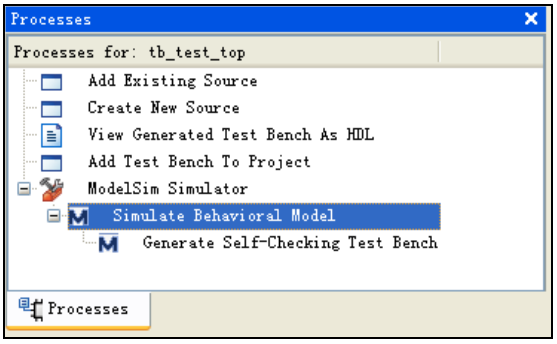

图 2.45　启动 ModelSim 行为仿真

（3）在资源管理器窗口中的“Source for”下拉菜单中，还可以选择翻译后仿真（Post- Translate Simulation）、映射后仿真（Post-Map Simulation）或者布局布线后仿真（Post-Route Simulation），如图 2.46 所示。选择后，在“Processes”窗口中会出现对应的操作以调用 ModelSim。

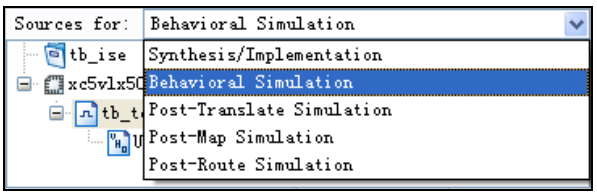

图 2.46　选择不同的仿真模式

（4）选择任何一种仿真操作，需要使用不同的仿真文件。如果相应的仿真文件不存在，ISE 会自动生成仿真文件。

2.2.5 使用 ModelSim 行为仿真的技巧

❖ 技巧内容

通过行为仿真，设计者可以验证设计的功能行为。一般来说，在进行布局布线之前，都必需通过行为仿真来验证电路的功能。本节介绍在 ISE 中使用 ModelSim 进行行为仿真的一些技巧。

❖ 技巧详解

在 ISE 中调用 ModelSim 行为仿真的技巧如下。

（1）启动 ISE 集成环境，打开已有的项目，建立好测试平台。

（2）在资源管理器窗口中“Source for”下拉菜单中选择“Behavioral Simulation”，可以看到测试平台文件出现在“Source”窗口中，如图 2.44 所示。

（3）选择仿真顶层文件“tb_test_top”，该文件是仿真顶层文件。其中例化了仿真顶层文件“test_top”，并对输入信号产生激励。例化仿真顶层文件和编写测试激励可以参考如下的例子。

在编写测试平台的时候，测试平台顶层可以不需要任何接口，测试平台内部需要例化被测试的模块，同时对该模块产生激励信号。

```
ENTITY tb_test_top IS
END tb_test_top;

ARCHITECTURE Testbench_arch OF tb_test_top IS
    --声明元件
    COMPONENT test_top
        PORT (
            clk : In std_logic;
            rst : In std_logic;
            dv_in : In std_logic;
            din : In std_logic_vector (7 DOwnTo 0);
            dv_out : Out std_logic;
            DOut : Out std_logic_vector (7 DOwnTo 0)
        );
    END COMPONENT;
--内部信号定义
    SIGNAL clk : std_logic := '0';
    SIGNAL rst : std_logic := '0';
    SIGNAL dv_in : std_logic := '0';
    SIGNAL din : std_logic_vector (7 DOwnTo 0) := "00000000";
    SIGNAL dv_out : std_logic := '0';
    SIGNAL DOut : std_logic_vector (7 DOwnTo 0) := "00000000";
--常数定义，主要用来定义时钟周期
    constant PERIOD : time := 20 ns;
    constant DUTY_CYCLE : real := 0.5;
    constant OFFSET : time := 0 ns;
```

```
BEGIN
--例化仿真元件
   UUT : test_top
   PORT MAP (
      clk => clk,
      rst => rst,
      dv_in => dv_in,
      din => din,
      dv_out => dv_out,
      DOut => DOut
   );
   --产生时钟信号
   PROCESS   -- clock process for clk
   BEGIN
      WAIT for OFFSET;
      CLOCK_LOOP : LOOP
         clk <= '0';
         WAIT FOR (PERIOD - (PERIOD * DUTY_CYCLE));
         clk <= '1';
         WAIT FOR (PERIOD * DUTY_CYCLE);
      END LOOP CLOCK_LOOP;
   END PROCESS;
   --产生测试激励信号
   PROCESS
      BEGIN
         -- -------------  Current Time:  7ns
         WAIT FOR 7 ns;
         rst <= '1';
         -- -------------  Current Time:  67ns
         WAIT FOR 60 ns;
         rst <= '0';
         -- -------------  Current Time:  107ns
         WAIT FOR 40 ns;
         din <= "00000001";
         ......
      END PROCESS;

END Testbench_arch;
```

（4）选择“Processes”窗口中的“Behavioral Simulation”，单击鼠标右键，在弹出的菜单中选择“Properties”，弹出属性设置对话框，如图 2.47 所示。

选择“Simulation Properties”选项，在界面右侧可以进行仿真属性设置，可以对如下内容进行设置。

- 测试平台使用的语言，可以为 VHDL 或者 Verilog。
- 是否使用 DO 文件，如果使用，需要指定 DO 文件的目录和文件名。
- 进行一次仿真的时间，该时间表示用户在 ModelSim 中使用一次“RUN”命令仿真的时间。
- 仿真的时间精度，默认值为 1ps（皮秒）。
- 指定仿真命令的参数，如 VSIM、VLOG 和 VCOM 的参数。

选择“Display Properties”，可以指定打开 ModelSim 后看到的窗口，设计者也可以在进入 ModelSim 软件后自行打开这些窗口。如图 2.48 所示，选择了“Signal window”、“Wave window”和“Structure window”，打开 ModelSim 后会显示这几个窗口。

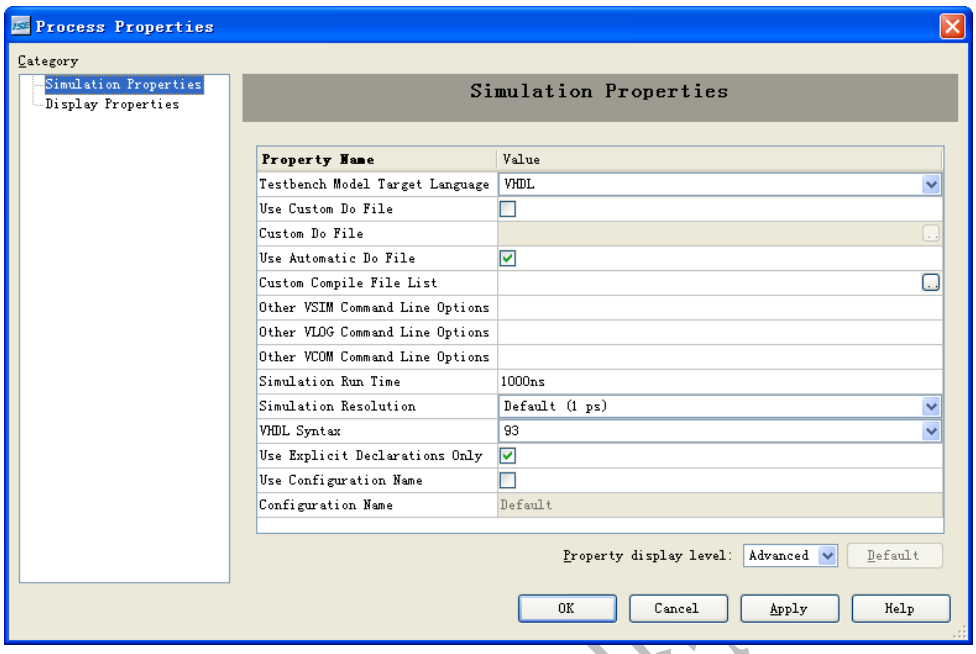

图 2.47　设置行为仿真属性

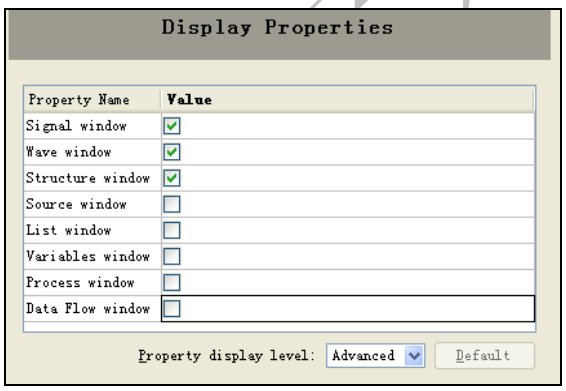

图 2.48　设置 ModelSim 中打开的窗口

（5）设置好以后单击【OK】按钮，这时会在当前目录下生成一个“.uDO”文件，包含了 ModelSim 会使用的一些命令，具体内容如下。

```
vlib work
vcom -explicit  -93 "test_top.vhd"
vcom -explicit  -93 "tb_test_top.vhw"
vsim -t 1ps   -lib work tb_test_top
view wave
add wave *
DO {tb_test_top.uDO}
view structure
view signals
run 1000ns
```

以上文件是将 ModelSim 的命令集合在一起的一个命令列表文件，对以上的命令解释如下。

- vlib work：建立一个工作目录，名称为 work。

- vcom -explicit -93 "test_top.vhd"：编译仿真文件，这里因为有两个仿真文件，所以执行了两条编译命令。
- vsim -t 1ps -lib work tb_test_top：仿真命令，指定了仿真的文件、仿真库。
- view wave *：打开波形窗口查看波形。
- add wave：添加波形，这里使用通配符“*”添加所有波形。
- DO {tb_test_top.uDO}：执行指定的 DO 文件。
- view structure：打开结构窗口。
- view signals：打开信号窗口。
- run 1000ns：执行时间为 1000ns。

（6）双击“Simulate Behavioral Model”即可进入 ModelSim 软件进行仿真，ModelSim 的使用方法会在专门的章节讲解。

2.3 命令行方式使用 ISE 的技巧

ISE 中提供了许多命令用于设计实现和验证读者的设计。使用命令行的方式可以让操作自动执行，在很多时候可以提高设计效率，尤其是进行综合、布局布线和生成下载文件的时候。

2.3.1 调用命令行的技巧

❖ 技巧内容

用户可以通过不同的方式调用命令行，运行设计者要求的操作，使用命令行执行 ISE 操作，实现自动化操作。

❖ 技巧详解

（1）设计者可以直接在 DOS 或者 Linux 界面下输入命令行，在 DOS 下使用命令行，需要在环境变量中指定 ISE 的路径。

（2）设计者还可以在“Project Navigator”界面下的“Transcript”窗口中选择“Tcl Shell”选项，然后在出现的窗口中输入命令行，如图 2.49 所示。事实上在图形界面下调用 ISE 的时候，在“Transcript”窗口中也会显示相应的命令行，设计者可以参考这些系统调用的命令行语句。

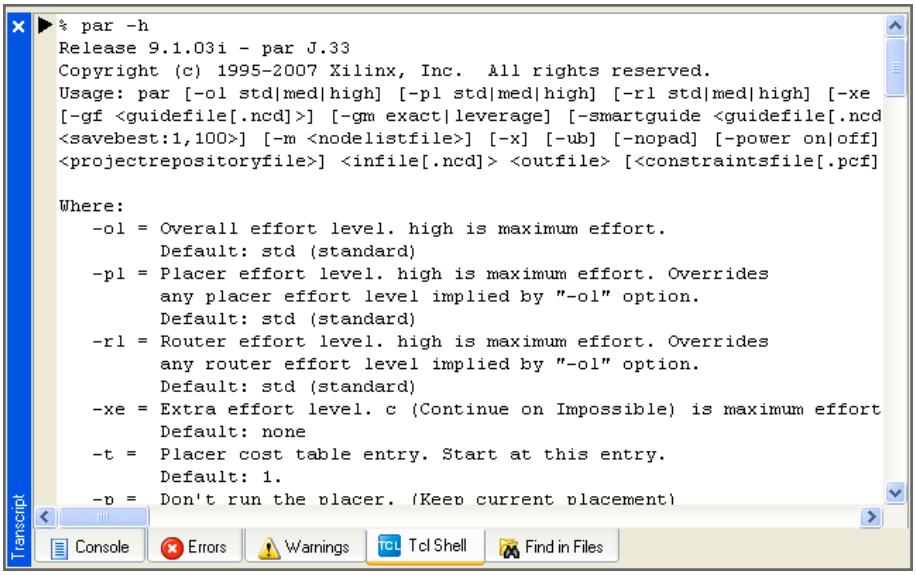

图 2.49 ISE 中的 TCL Shell 窗口

2.3.2 命令行的语法技巧

❖ 技巧内容

命令行总是以程序命令名称开头，后面跟随相应的参数。只有遵循正确的语法规则，才能保证命令行正确执行。

❖ 技巧详解

使用命令行方式的时候，需要注意以下语法。

- 命令行总是以命令作为开头。
- 对于命令选项的顺序没有强制要求，但是命令选项之前需要加上中划线“-”，各个选项之间要用空格进行分隔。
- 命令选项后如果有参数，需要用空格来分隔选项和参数。
- 命令行方式中指定文件路径时可以使用绝对路径，也可以使用相对路径。

比如设计者要使用 PAR 命令，代码如下。

```
    par -w -intstyle ise -ol high -pl high -rl high d:/test/top_map.ncd d:/test/ top_par.ncd d:/test/top_map.pcf
```

本书会在接下来的章节中对各个命令以及相应的参数进行说明，这里只是给读者一个初步印象。

有的选项对于所有命令都是适用的，这些常用的选项在设计者使用命令行方式的时候能够提供很大的帮助，常用的选项如下所示。

- -f（执行命令文件）：设计者可以把某些命令的选项保存为一个文件，这样在每次输入命令的时候，不用输入很长的选项。比如上面的例子中，可以把选项-w -intstyle ise -ol high -pl high -rl high d:/test/top_map.ncd d:/test/top_par.ncd d:/test/top_map.pcf 保存到文件 op.txt 中，这样只需要输入“par -f op.txt”即可。
- -h（帮助）：如果不知道命令的使用方法，可以使用该选项查看帮助信息。比如不知道 par 命令的使用方法，输入 par –h 后会出现如下信息。

```
    Usage: par [-ol std|med|high] [-pl std|med|high] [-rl std|med|high] [-xe n|c] [-t <costtable:1,100>] [-p] [-k] [-r] [-w]
    [-gf <guidefile[.ncd]>] [-gm exact|leverage] [-smartguide <guidefile[.ncd]>] [-n <iterations:0,100>] [-s
    <savebest:1,100>] [-m <nodelistfile>] [-x] [-ub] [-nopad] [-power on|off] [-intstyle ise|xflow|silent] [-ise
    <projectrepositoryfile>] <infile[.ncd]> <outfile> [<constraintsfile[.pcf]>]

    Where:
       -ol = Overall effort level. high is maximum effort.
            Default: std (standard)
       -pl = Placer effort level. high is maximum effort. Overrides
            any placer effort level implied by "-ol" option.
            Default: std (standard)
       -rl = Router effort level. high is maximum effort. Overrides
            any router effort level implied by "-ol" option.
            Default: std (standard)
       -xe = Extra effort level. c (Continue on Impossible) is maximum effort.
            Default: none
    …
```

- -p（Part Number）：用于指定器件型号。

2.3.3 NGDBUILD 命令使用技巧

❖ 技巧内容

NGDBUILD 命令用于生成 Xilinx 标准的 NGD 网表文件，用于产生 NGD 网表文件的原始文件为 EDIF 和 NGC 格式的网表文件。

NGD 文件是以逻辑门、触发器、RAM、复用器等形式的逻辑模块来描述设计的网表文件，描述了数字电路的逻辑层次。NGD 文件可以映射到 Xilinx 的器件中，进行布局布线。

❖ 技巧详解

NGDBUILD 生成 NGD 文件的技巧如下。

（1）命令的使用语法为：

```
ngdbuild [options] design_name [ngd_file[.ngd]]
```

- options：命令行的参数。
- design_name：需要进行处理的设计顶层文件的名字，这里就是输入的 EDIF 或者 NGC 网表文件的名字，可以为设计指定输入文件的路径。路径可以为绝对路径或者相对路径。
- ngd_file.ngd 是输出的 NGD 网表文件，可以为其指定路径。

（2）在执行该命令的时候，设计者可以根据需要指定参数，NGDBUILD 提供了很多参数供设计者使用，常用的参数包括以下几种。

- -p（指定器件型号）：这个参数用于指定器件的类型，NGDBUILD 可以针对指定的器件进行优化。可以指定器件的系列，也可以指定器件的具体型号（型号和封装），还可以包含器件的速度（型号、封装和速度）。

```
-p xc5vlx50t：指定器件系列为 Virtex-5
-p xc5vlx50t-ff1136：指定器件系列为 Virtex-5，型号为 LX50T，封装为 FF1136
-p xc5vlx50t-ff1136-1：指定器件系列为 Virtex-5，型号为 LX50T，封装为 FF1136，速度等级-1
```

- -dd（指定临时文件夹）：指定命令执行过程中产生的中间文件的存放位置。如果不设置该参数，中间文件会存放在当前目录下。比如设置中间文件为当前文件夹下的 temp 目录，可以指定为“–dd ./temp”，“./”表示当前目录。
- -sd（指定搜索路径）：指定设计中用到的其他文件，比如 EDIF 网表文件、NGO 文件、MEM 文件等。如果这些引用的文件存放在设计顶层目录，则不需要指定。
- -uc（指定 UCF 文件）：指定 UCF 约束文件。
- -bm（指定 BMM 文件）：BMM 文件是在使用 Xilinx 的 PowerPC 或者 Microblaze 嵌入式系统时生成的内存指定的文件。
- -f（执行命令文件）：设计者可以把某些命令的选项保存为一个文件，这样在每次输入命令的时候，不用输入很长的选项。

（3）设计者可以参考以下的例子来使用 NGDBUILD 命令，只需要修改相应的参数即可。

- 指定了临时文件夹、UCF 约束文件、只有一个搜索路径、输入和输出文件在当前目录的使用方法。

```
在本例中，选用的器件是 Virtex-5 系列的 xc5vlx50t-ff1136-1
临时目录为 d:/test/temp
搜索路径为 d:/test/macro
输入的网表文件为 test_top.edf
输出的 NGD 文件为 test_top_ngdbuild.ngd
ngdbuild -p xc5vlx50t-ff1136-1 -dd d:/test/temp -sd d:/test/macro –uc d:/test/test_top.ucf
test_top.edf test_top_ngdbuild.ngd
```

- 指定了临时文件夹、UCF 约束文件、有多个搜索路径、输入和输出文件在其他位置的使用方法。

```
    在本例中，选用的器件是 Virtex-5 系列的 xc5vlx50t-ff1136-1
    临时目录为 d:/test/temp
    搜索路径为 d:/test/macro 和 d:/test/ip
    输入的网表文件为 d:/test/synthesis/test_top.edf
    输出的 NGD 文件为 d:/test/test_top_ngdbuild.ngd
    ngdbuild -p xc5vlx50t-ff1136-1 -dd d:/test/temp -sd d:/test/macro –sd d:/test/ip -uc
d:/test/test_top.ucf d:/test/synthesis/test_top.edf d:/test/test_ top_ngdbuild.ngd
```

- 指定了临时文件夹、UCF 约束文件、输入和输出文件在其他位置、使用相对路径的使用方法。

```
    在本例中，选用的器件是 Virtex-5 系列的 xc5vlx50t-ff1136-1
    假设当前目录为 d:/test, ../表示当前目录的上一级，即 d:/
    临时目录为 d:/temp
    搜索路径为 d:/macro
    输入的网表文件为 d:/synthesis/test_top.edf
    输出的 NGD 文件为 d:/par/test_top_ngdbuild.ngd
    BMM 文件为嵌入式系统的内存分配的文件，当设计中使用了 Microblaze 或者 PowerPC 就会使用该文件。
    ngdbuild -p xc5vlx50t-ff1136-1 -dd ../temp -sd ../macro –uc ../script/test_top.ucf
–bm ../macro/system_stub.bmm ../synthesis/test_top.edf ../par/test_top_ngdbuild.ngd
```

（4）执行该命令后，会输出相应的信息。如果设计中有错，会显示出错信息，设计者需要根据这些信息来修改设计。

2.3.4 MAP 命令使用技巧

❖ 技巧内容

MAP 命令的功能是将逻辑设计映射到 Xilinx 的 FPGA 内部资源中，NGDBUILD 生成的 NGD 文件会作为 MAP 的输入。MAP 首先对 NGD 进行 DRC（Design Rule Check），然后将逻辑电路映射到 FPGA 中，比如 logic cell、I/O cell、Block RAM 等资源。
MAP 的输出文件为 NCD 格式，该文件将作为布局布线的输入文件。

❖ 技巧详解

使用 MAP 命令的技巧如下。
（1）MAP 命令的语法为：

```
    map [options] infile[.ngd] [pcf_file[.pcf]]
```

- options：命令行的参数。
- infile.ngd： MAP 命令的输入文件。可以不必输入后缀名，因为 MAP 会自动搜寻后缀名为“ngd”的文件作为输入文件。
- pcf_file.pcf：物理约束文件。

（2）Xilinx 为 MAP 提供了很多参数，这里介绍常用参数。

- –c（Pack CLBs）： 指定用于实现 MAP 命令操作的 CLB 的百分比，数值范围从 1 到 100。
- -cm（Cover Mode）：指定 MAP 命令执行覆盖阶段的模式：area 指定 MAP 首先要考虑的是占用最少的 LUT，speed 指定 MAP 操作优先考虑速度，balance 指定 MAP 命令要在速度和面积之间平衡。
- -gf（Guide NCD File）：-gf 参数指定一个 NCD 文件作为 Guide 文件，NCD 文件是由前一次 MAP 操作生成的，MAP 使用已有的 NCD 文件作为当前操作的 Guide 文件，以缩短 MAP 的时间，实现增量操作。

- -gm（Guide Mode）：指定 MAP 使用 Guide 文件的模式，包括两种模式：exact 表示完全按照 Guide 文件的信息来操作，leverage 表示 Guide 文件只是作为一个起始点来完成操作。-gm 需要和-gf 一起使用。
- －ignore_keep_hierarchy（Ignore KEEP_HIERARCHY Properties）：使用该参数，MAP 操作可以忽略代码中的 KEEP_HIERARCHY 属性，重新安排逻辑设计的层次达到优化效果。
- -intstyle（Integration Style）：这个参数可以用来减少屏幕输出。根据指定的值，有 3 种模式可以用来指定屏幕输出，ise 表示命令是运行于集成环境下，xflow 表示命令是运行于命令行方式下，silent 表示只输出 warning 和 error 信息。
- -o（输出文件名）：指定输出文件的路径和名称。
- -ol（Overall Effort Level）：指定总的优化难度，有 3 个级别的难度，STD 表示最低级别难度，运行时间最少，但是优化效果不好；MEDIUM 表示在运行时间和优化效果之间平衡；HIGH 表示最好的优化效果，但是最费时。
- -p（指定器件型号）：这个参数用于指定器件的类型。MAP 可以针对指定的器件进行优化，可以指定器件的系列，也可以指定器件的具体型号（型号和封装），还可以包含器件的速度（型号、封装和速度）。
- -smartguid（SmartGuide）：使用该参数后，MAP 命令就会利用上次 MAP 操作的结果来指导当前操作，实现增量编译。使用-smartguide 可以替代-gm 和-gf 参数。如果没有指定 Guide 文件，该参数不会起作用。
- -w（Overwrite Existing Files）：使用该参数，MAP 会用新生成的文件去覆盖之前的文件。当执行 MAP 命令的文件夹包含或指定输出同名的文件，需要指定该参数，否则会提示出错。

（3）下面给出部分例子，读者可以参考这些例子来执行 MAP 命令。

- -intstyle silent：设计者指定集成类型为“Silent”，这样在命令行模式下只会输出“warning”和“error”信息。
- -ol high：优化级别为高。这样会花费更长的时间来进行 MAP 操作，以便达到最优化的结果。
- -cm speed：MAP 过程中优先考虑速度。因为现在的 FPGA 容量都比较大，所以在容量足够的情况下，建议设计者以速度优先。
- -c 100：表示使用全部的 CLB 资源。
- -o d:/test/test_top_map.ncd：指定输出文件的路径和文件名。
- 输入文件为“d:/test/test_top_ngdbuild.ngd”。
- 产生的 PCF 物理约束文件为“d:/test/test_top_map.pcf”。

```
    map -intstyle silent -ol high -cm speed -ignore_keep_hierarchy -c 100 -w -o d:/ test/test_top_map.ncd
d:/test/test_top_ngdbuild.ngd d:/test/test_top_map.pcf
```

- -smartguide d:/test/test_top_map_guide.ncd：将以前进行 MAP 操作生成的 NCD 文件更名为“test_top_map_guide.ncd”，然后指定该文件为 Guide 文件进行 MAP 操作。这样 ISE 会使用以前的 MAP 信息指导当前的 MAP 操作，可以缩短命令执行时间。
- 使用“smartguide”参数时，需要注意的是生成 Guide 文件的设计和当前设计不能有太多的不同，否则效果不明显。如果是对比较小的改动使用“smartguide”参数，效果会更明显。

```
    map -intstyle silent -ol high -cm speed -ignore_keep_hierarchy -c 100 -w –o
d:/test/test_top_map.ncd d:/test/test_top_ngdbuild.ngd -smartguide d:/test/ test_ top_map_guide.ncd
d:/test/test_top_map.pcf
```

2.3.5 PAR 命令使用技巧

❖ 技巧内容

布局布线 PAR（Place And Route）命令接受映射后的 NCD 文件为输入，然后进行布局布线，生成布局布线后的 NCD 文件，以生成下载 FPGA 的 BIT 文件。

在布局阶段，布局器根据位置约束和时序约束，将元件放置到 FPGA 中，然后布线器开始布线，生成 NCD 文件。

❖ 技巧详解

使用 PAR 命令布局布线的技巧如下。

（1）PAR 的语法是：

```
par [options] infile[.ncd] outfile [pcf_file[.pcf]]
```

- options：命令的参数，各参数之间用空格分开。
- infile.ncd：MAP 命令的输出文件，作为 PAR 的输入文件，可以不必输入后缀名。
- outfile：PAR 的输出文件，后缀名为 NCD。为了和 MAP 的 NCD 区分，可以使用和 MAP 不同的文件名。
- pcf_file[.pcf]：物理约束文件（Physical Constraints File），MAP 命令会将用户的 UCF 中的约束添加到 PCF 文件中。

（2）Xilinx 为 PAR 命令提供了很多参数，常用参数包括以下几种。

- -gf（Guide NCD File）：-gf 参数指定一个 NCD 文件作为 Guide 文件。NCD 文件是由前一次 PAR 操作生成的，PAR 使用已有的 NCD 文件作为当前操作的 Guide 文件，以便缩短 PAR 的时间，实现增量式布局布线。
- -gm（Guide Mode）：指定 PAR 使用 Guide 文件的模式，包括两种模式：exact 表示完全按照 Guide 文件的信息来操作，leverage 表示 Guide 文件只是作为一个起始点来完成操作。-gm 需要和-gf 一起使用。
- -intstyle（Integration Style）：这个参数可以用来减少屏幕输出，根据指定的值，有三种模式可以用来指定屏幕输出：ise 表示命令是运行于集成环境下，xflow 表示命令是运行于命令行方式下，silent 表示只输出 warning 和 error 信息。
- -ol（Overall Effort Level）：指定总的优化难度，有 3 个级别的难度，STD 表示最低级别难度，运行时间最少，但是优化效果不好；MEDIUM 表示在运行时间和优化效果之间平衡；HIGH 表示最好的优化效果，但是最费时。
- -pl（Placer Effort Lvel）：指定布局器优化难度级别，包括 3 个级别，分别是 std、med、high。
- -rl（Router Effort Level）：指定布线器优化难度级别，包括 3 个级别，分别是 std、med、high。
- -smartguide（SmartGuide）：使用该参数后，MAP 命令就会利用上次 MAP 操作的结果来指导当前操作，实现增量编译。使用-smartguide 可以替代-gm 和-gf 参数。如果没有指定 Guide 文件，该参数不会起作用。
- -w（Overwrite Existing Files）：使用该参数，PAR 会用新生成的文件去覆盖之前的文件。当执行 PAR 命令的文件夹包含或指定的输出同名的文件，需要指定该参数，否则会提示出错。

（3）下面给出应用实例，读者可以参考这些例子来执行 PAR 命令。

指定输入文件为“d:/test/my_top_map.ncd”，输出文件为“d:/test/my_top_par.ncd”，因为 MAP 和 PAR 输出文件后缀名都为 NCD，所以可以取不同的文件名以示区分。

这里使用的 guide 文件为上次 PAR 操作产生的。

```
par -w -intstyle ise -ol high -pl high -rl high d:/test/test_top_map.ncd d:/ test/test_ top_par.ncd
d:/test/test_top_map.pcf -smartguide d:/test/test_top_par_guide.ncd
```

2.3.6 BITGEN 命令使用技巧

❖ 技巧内容

完成布局布线后，需要使用 BITGEN 命令生成比特流文件，下载到 FPGA 才能实现相应的电路功能。BITGEN 的输入文件为 PAR 输出的 NCD 文件，BITGEN 的输出文件为 BIT 文件。

❖ 技巧详解

使用 BITGEN 生成比特流文件的技巧如下。

BITGEN 命令的语法是：

```
bitgen [options] infile[.ncd] [outfile] [pcf_file.pcf]
```

- options：命令的参数，各参数之间用空格分开。
- infile.ncd：PAR 命令的输出文件，作为 BITGEN 的输入文件，可以不必输入后缀名。
- outfile：BITGEN 的输出文件，后缀名为 BIT。
- pcf_file[.pcf]：物理约束文件（Physical Constraints File），MAP 命令会将用户的 UCF 中的约束添加到 PCF 文件中。

Xilinx 为 BITGEN 命令提供了很多参数，常用参数包括以下几种。

- -w（Overwrite Existing Files）：使用该参数，BITGEN 会用新生成的文件去覆盖之前的文件。当执行 PAR 命令的文件夹包含或指定的输出同名的文件，需要指定该参数，否则会提示出错。
- -bd（Update Block Rams）：该参数指定用于更新 Block RAM 的 ELF 或者 MEM 文件，ELF 文件是为 Microblaze 或 PowerPC 生成的可执行文件。

下面给出应用实例，读者可以参考例子来使用 BITGEN 命令。

在这个例子中指定了用于更新 Block RAM 的 ELF 文件，该文件用于在系统设计中包含 Microblaze 或者 PowerPC 嵌入式系统的情况。

```
bitgen -w -bd d:/test/soft_test.elf d:/test/test_top_par.ncd d:/test/test_top.bit d:/test/test_top_map.pcf
```

2.3.7 TRACE 命令使用技巧

❖ 技巧内容

TRACE 命令用于执行静态时序分析，产生分析报告。TRACE 可以对 MAP 映射后的结果、布局之后的结果和布线之后的结果进行分析。

❖ 技巧详解

（1）TRACE 命令的语法为：

```
trace [options] design[.ncd] [constraint[.pcf]]
```

- options：为命令的参数，各参数之间用空格分开。
- design.ncd：为需要分析的文件，可以是 MAP 映射后的结果、布局之后的结果和布线之后的结果。
- constraint[.pcf]：物理约束文件（Physical Constraints File），MAP 命令会将用户的 UCF 中的约束添加到 PCF 文件中。

（2）TRACE 命令的常用参数如下所示。

- -a（Advanced Analysis）：如果不指定约束文件，需要指定该参数。
- -e（Generate an Error Report）：指定 TRACE 产生的报告为错误信息报告，而不包含其他的时序分析信息。
- -intstyle（Integration Style）：这个参数可以用来减少屏幕输出，根据指定的值，有 3 种模式可以用来指定屏幕输出，ise 表示命令是运行于集成环境下，xflow 表示命令是运行于命令行方式下，silent 表示只输出 warning 和 error 信息。

下面给出一个例子，读者可以参考例子来使用 TRACE 命令。

如下的例子中，指定 TRACE 只产生错误报告，并且指定了物理约束文件，输入文件为布局布线之后的输出结果。

```
trce -e -o d:/test/test_top.twr d:/test/test_top_par.ncd d:/test/test_top_map.pcf
```

2.3.8 DATA2MEM 命令使用技巧

❖ 技巧内容

DATA2MEM 命令用于将 CPU 的可执行代码或者数据文件初始化到 Block RAM 中，比如将用户生成的 ELF 可执行文件更新到 Block RAM 中，就可以执行新的软件程序。

❖ 技巧详解

（1）DATA2MEM 命令的语法为：

```
data2mem -bm|bd infile.[bmm|elf|mem][options]
```

- options：为命令的参数，各参数之间用空格分开。
- infile：指定输入文件的名称。

（2）DATA2MEM 命令的常用参数如下所示。

- -bm：指定输入文件的类型为 BMM 类型。
- -bd：表示输入文件为 MEM 或者 ELF 类型。
- -bt：指定输入的 BIT 文件。
- -intstyle：这个参数可以用来减少屏幕输出，根据指定的值，有 3 种模式可以用来指定屏幕输出，ise 表示命令是运行于集成环境下，xflow 表示命令是运行于命令行方式下，silent 表示只输出 warning 和 error 信息。
- -log：产生日志文件以记录相应的信息。
- -o：指定输出文件的位置和名称。
- -p：指定使用的器件型号。
- -w：用新生成的文件去覆盖之前的文件。如果输出文件夹包含和指定的输出同名的文件，需要指定该参数，否则会提示出错。

（3）下面给出一个例子，读者可以参考例子来使用 DATA2MEM 命令。

```
    器件为 xc5vlx50t-ff1136-3
    输入的 ELF 文件为 d:/test/test.elf
    输入的 BMM 文件为 d:/test/system_stub.bmm
    输入的 BIT 文件为 d:/test/sopc_test.bit
    将可执行程序初始化到 Block RAM 后生成的 BIT 文件为 d:/test/sopc_test_DOwnload.bit
    日志文件为 d:/test/sopc_test.dmr
    data2mem -p xc5vlx50t-ff1136-3 -bd d:/test/test.elf -bm d:/test/system_stub.bmm -bt
d:/test/sopc_test.bit -o d:/test/sopc_test_DOwnload.bit -log d:/test/sopc_test.dmr
```

2.3.9 自动执行命令行的技巧

❖ 技巧内容

使用命令行最大的优势就是可以让许多操作自动运行，设计者不需要关注中间的过程，只需要得到最后的结果，分析最终的报告。

❖ 技巧详解

设计者完成综合后，可以利用批处理文件，自动完成布局布线直到生成 BIT 文件的操作，具体步骤如下。

（1）新建一个文件，后缀名为 BAT，表示是一个批处理文件，在 Windows 操作系统中，可以直接运行以 BAT 为后缀名的文件。这里假设为“test.bat”。

（2）在批处理文件中，首先输出 NGDBUILD 命令，然后是 MAP 命令，接下来输入 PAR 命令。
（3）如果需要生成 BIT 文件，输入 BITGEN 命令。
（4）最后是 TRACE 命令，生成相应的报告。
（5）具体的实例如下，读者可以将此例子复制到 BAT 文件中，根据具体情况修改后直接运行。

```
    ::------------------------------------------------
    ::自动执行 ISE 命令,完成从 NGDBUILD 直到生成下载文件的操作
    ::------------------------------------------------

    ::表示当前目录下如果不存在 TEMP 文件夹,则新建 TEMP 文件夹
    ::临时文件夹中将存放执行过程的中间文件
    ::"./"表示当前目录, @表示在命令窗口中不回显该命令
    @if not exist ./temp md ./temp
    ::PAR 目录用于存放布局布线的结果
    @if not exist ./par md ./par
    @if not exist ./marco md ./macro

    ::复制 BMM 文件到 MACRO 目录,方便操作,这里的目录分隔符号为'\'
    ::也可以不复制,在命令中直接指定输入文件的位置
    copy ..\microblaze\system_stub.bmm .\macro\system_stub.bmm
    ::复制 NGC 文件,NGC 文件为 Xilinx 嵌入式处理器生成的网表文件
    copy ..\ microblaze\*.ngc .\macro\
    ::复制 ELF 文件, 为可执行的代码
    copy ..\ microblaze\*.elf .\macro\
    ::复制所有的 EDF 网表文件到 MACRO 目录
    copy ..\synthesis\*.edf .\macro\

    ::临时文件夹为 TEMP, 使用到了嵌入式处理器, 所以需要指定 BMM 文件
    ngdbuild -p xc5vlx50t-ff1136-1 -dd ./temp -sd ./macro -uc ./script/system.ucf
-bm ./macro/system_stub.bmm ./macro/test_top.edf ./par/test_top_ngdbuild.ngd
    ::NGDBUILD 正确执行后, 会给系统返回一个值, 为 0, 否则为 1
    ::当没有正确执行操作, 就结束操作, 输出错误信息
    @if not errorlevel==0 goto lab_fail

    ::MAP 操作, 生成的文件都存放到 PAR 目录下
    map -intstyle ise -ol high -cm speed -ignore_keep_hierarchy -c 100 -tx on –w
-o ./par/test_top_map.ncd ./par/test_top_ngdbuild.ngd ./par/test_top_map.pcf
    ::当没有正确执行操作, 就结束操作, 输出错误信息
    @if not errorlevel==0 goto lab_fail

    ::PAR 操作, 为了和 MAP 区分, 为 NCD 文件取不同的名字
    par -w -intstyle ise -ol high -pl high -rl high ./par/test_top_ map.ncd ./par/
test_top_par.ncd ./par/test_top_map.pcf
    ::当没有正确执行操作, 就结束操作, 输出错误信息
    @if not errorlevel==0 goto lab_fail

    ::生成 bit 文件, 需要将 ELF 文件也初始化到 BIT 文件中
    bitgen -w -g unusedpin:pullnone -bd./macro/software_test.elf./par/test_ top_
par.ncd ./par/test_top.bit ./par/test_top_map.pcf
    ::当没有正确执行操作, 就结束操作, 输出错误信息
    @if not errorlevel==0 goto lab_fail
```

```
::生成时序分析报告
trce -e -o ./par/test_top.twr ./par/test_top_par.ncd ./par/test_top_map.pcf

::提示正确完成操作, ECHO 为回显命令
@echo ----------------------------------
@echo -- Successful ……
@echo ----------------------------------

::跳转到标签"lab_end", 避免出现不必要的信息
@goto lab_end

::标签"lab_fail", 可以在批处理中引用
:lab_fail
@echo !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
@echo !! FAILED, CHECK REPORT FILE
@echo !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

@echo press any key to end
@pause

:lab_end
```

（6）执行批处理文件后，会出现如图 2.50 所示窗口信息。

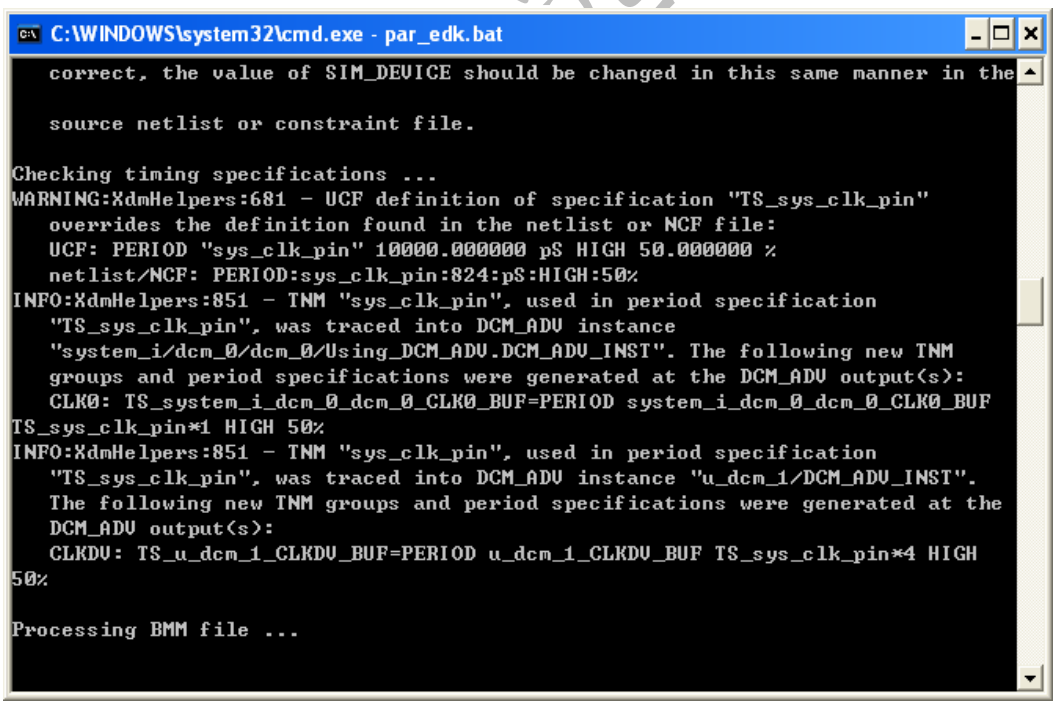

图 2.50　自动运行命令行的提示窗口

（7）设计者也可以利用输出重定向的方式将屏幕输出的信息保存到文件中，比如批处理文件为“test.bat”，在 CMD 窗口中输入“test.bat >> test.log”，就可以将窗口信息保存到 test.log 文件中。

2.4　CORE Generator 使用技巧

Xilinx 针对其 FPGA 提供大量成熟、高效、稳定的 IP 核为用户所用，使用 CORE Generator 可以生成 Xilinx 提供的各种 IP。本节介绍 CORE Generator 的使用技巧。

2.4.1 新建 CORE Generator 项目的技巧

❖ 技巧内容

CORE Generator 可新建和管理 Xilinx 的 IP，提供图形化的界面方便用户设置 IP 参数，管理生成的 IP。为了能对用户的 IP 进行统一管理，首先需要新建一个 CORE Generator 项目，在项目中包含了用户 IP 的 HDL 文件、网表文件以及 IP 的参数配置信息。

❖ 技巧详解

新建 CORE Generator 项目的方法非常简单，具体步骤如下。

（1）首先需要启动 CORE Generator，在开始菜单中，ISE 菜单下单击【Accessories】/【CORE Generator】，出现如图 2.51 所示的窗口。

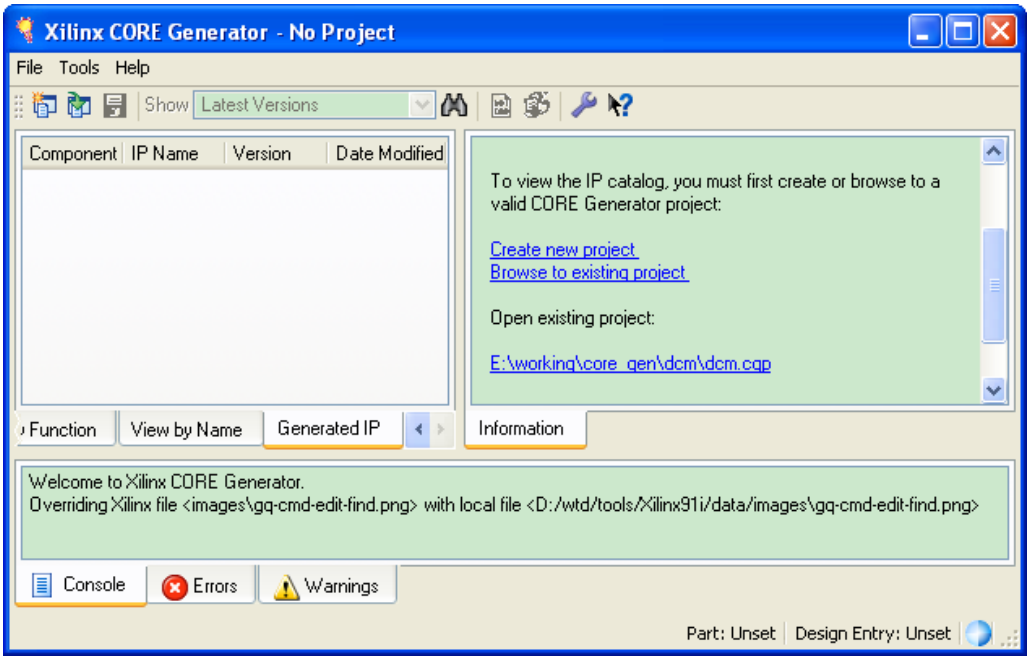

图 2.51　启动 CORE Generator 窗口

（2）在图 2.51 所示的窗口中，可以单击【File】/【New Project】新建项目，也可以在右边的“Information”窗口中单击“Create new project”新建项目。

（3）如果已经有了 CORE Generator 项目，还可以打开已有项目。可以通过“Browse to existing project”找到已有的项目，还可以通过“Open existing project”打开 Core Generator 已经识别的项目。

（4）单击新建项目后，在弹出的“New Project”对话框中输入项目名称，指定项目存放路径，单击【OK】按钮，如图 2.52 所示。

（5）如果指定的目录不存在，会提示新建目录，单击【OK】按钮新建目录。

（6）接下来会出现项目属性设置对话框，设计者需要指定器件类型等属性。如图 2.53 所示为指定器件类型对话框。这里指定器件为 Virtex5 系列的 xc5lvx30，封装类型为 ff324，速度等级为 3。

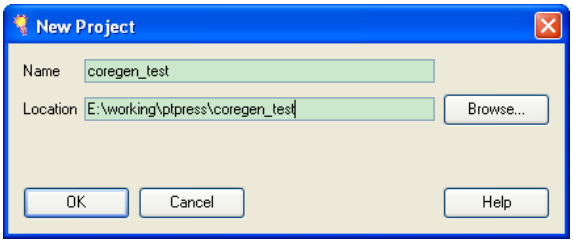

图 2.52　新建 CORE Generator 项目

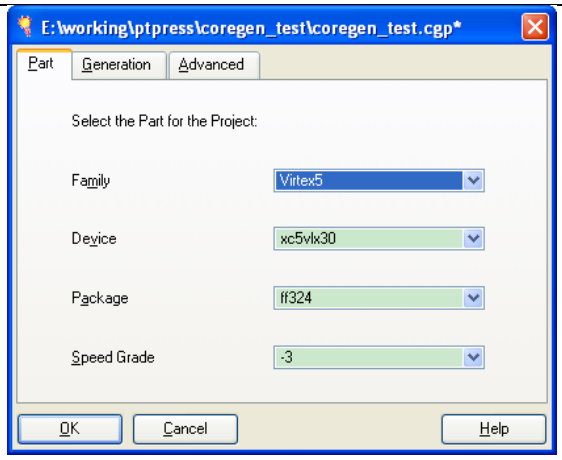

图 2.53　为 CORE Generator 新项目指定器件类型

（7）单击“Generation”选项卡，设置生成 IP 的属性，如图 2.54 所示。设置生成的文件为 VHDL 类型，相应的网表文件为 EDIF 类型，仿真文件为行为级仿真，设计使用的综合工具为 ISE，设置完后单击【OK】按钮。

（8）单击“Advanced”选项卡，可以设置高级选项，如图 2.55 所示。

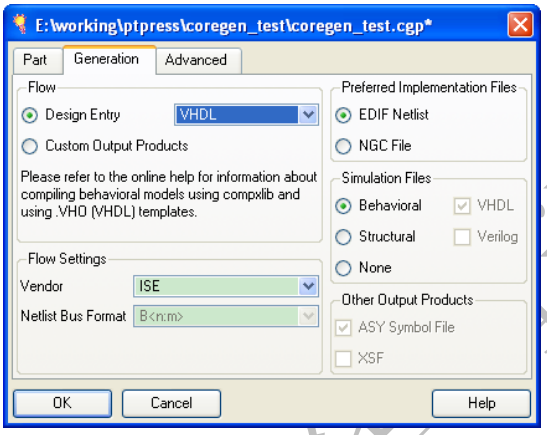

图 2.54　为 CORE Generator 设置生成文件属性

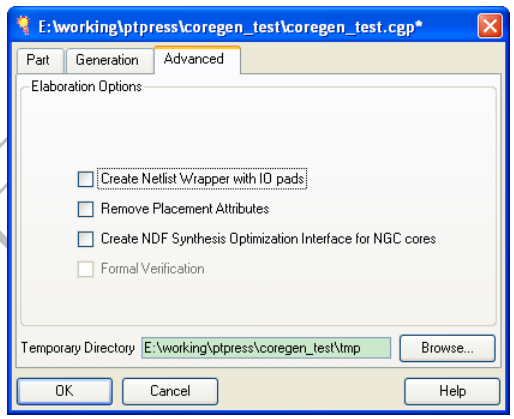

图 2.55　为 CORE Generator 设置高级参数

- 如果需要将生成的 IP 的输入/输出信号添加 IO 引脚，则勾选“Create Netlist Wrapper with IO pads”。
- 如果需要删除生成的关于布局方面的属性，就勾选“Remove Placement Attributes”。
- 如果生成的 IP 网表文件为 NGC 格式的文件，可以生成 NDF 综合优化文件。
- 还可以指定项目的临时目录。

（9）单击【OK】按钮完成新建项目，接下来用户可以看到 IP 列表窗口，设计者可以选择相应的 IP 设置参数，生成希望的 IP，如图 2.56 所示。

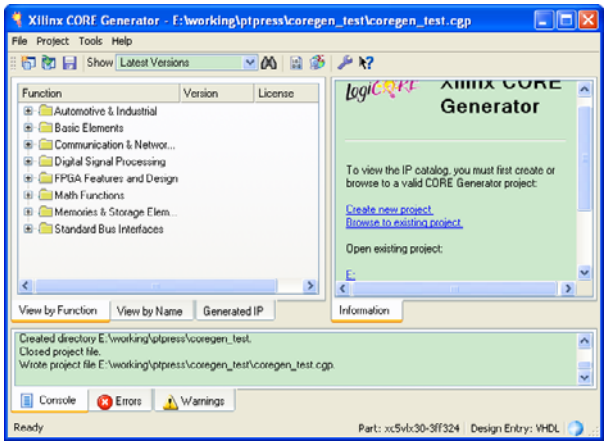

图 2.56　CORE Generator IP 分类列表

2.4.2　新建 IP 的技巧

❖ 技巧内容

建立好 CORE Generator 项目后，就可以新建 IP 了。在 CORE Generator 中对 IP 进行了分类，方便选择。

❖ 技巧详解

新建 IP 的操作步骤如下。

（1）启动 CORE Generator，打开已有项目。

（2）在打开的项目窗口中选择“View by Function”选项卡，找到需要的 IP，如图 2.57 所示。选择“Basic Elements”目录下的“Comparators”分支，然后选择“Comparator”。

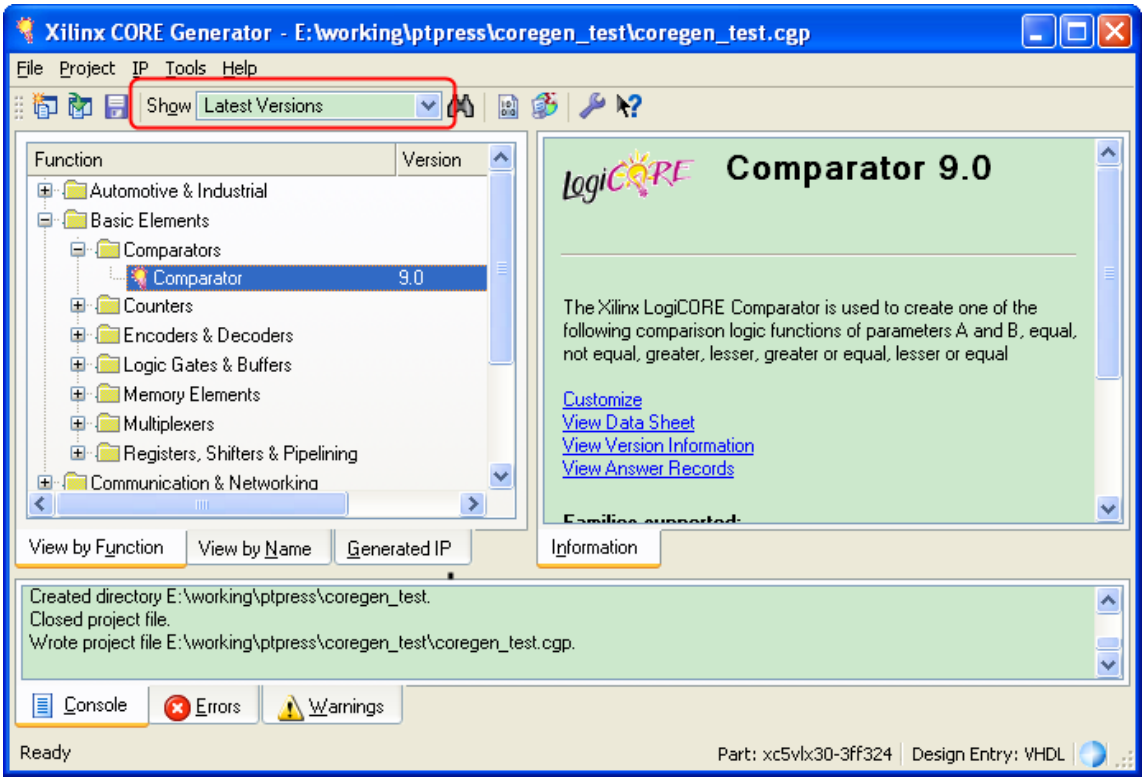

图 2.57　新建 IP 窗口

（3）新建 IP 的窗口中会显示最新版本的 IP，如果设计者希望使用其他版本的 IP，可以在窗口顶端的版本下拉菜单中选择。

（4）如果设计者希望了解 IP 的使用方法，可以在右边窗口中单击“View Data Sheet”，打开相应 IP 的数据手册。

（5）当不知道 IP 所属的分类的时候，可以单击“View by Name”选项卡，IP 会以名称为顺序排列。

（6）找到 IP 后，可以双击 IP 名称开始设置 IP，也可以单击“Customize”开始设置 IP 参数，如图 2.58 所示。

（7）定义好 IP 元件名称，设置好 IP 参数后，单击【Finish】按钮，新的 IP 的文件生成在项目所在的目录中，同时会给出相应的报告，如图 2.59 所示。

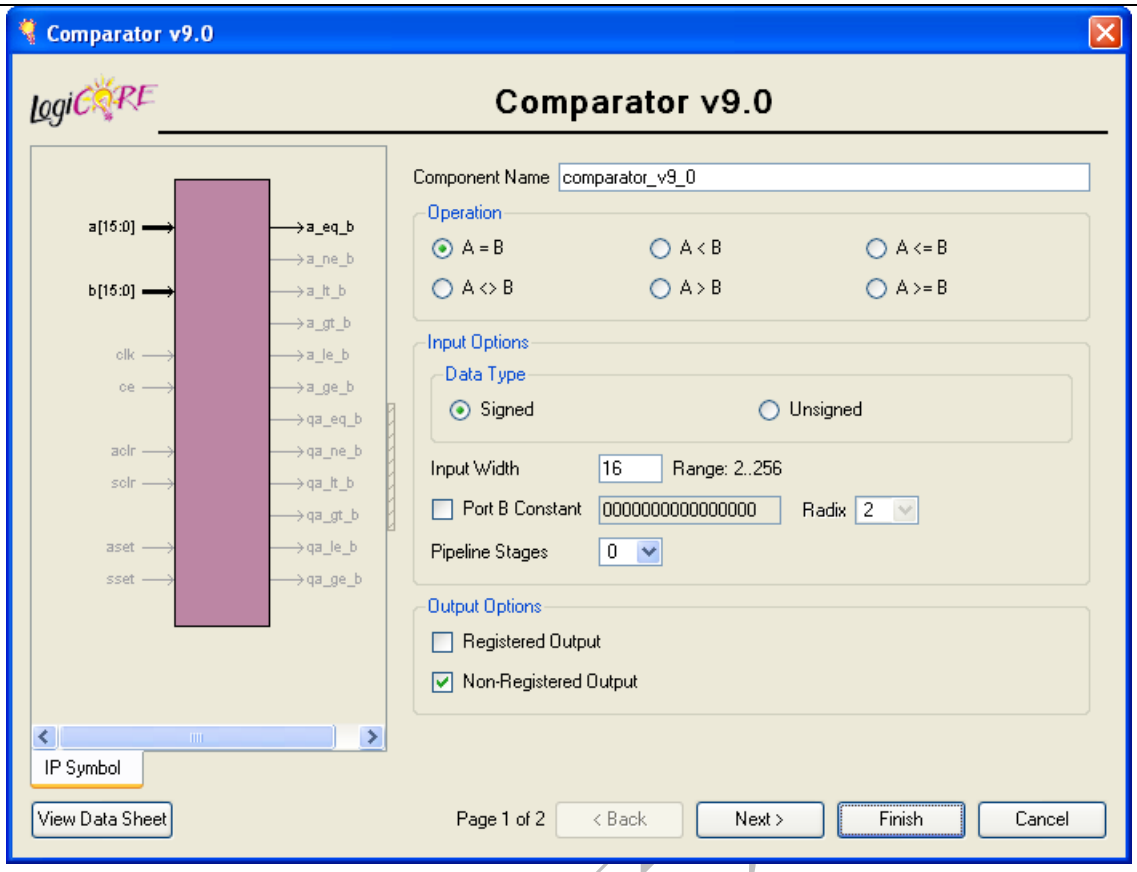

图 2.58　设置 IP 参数

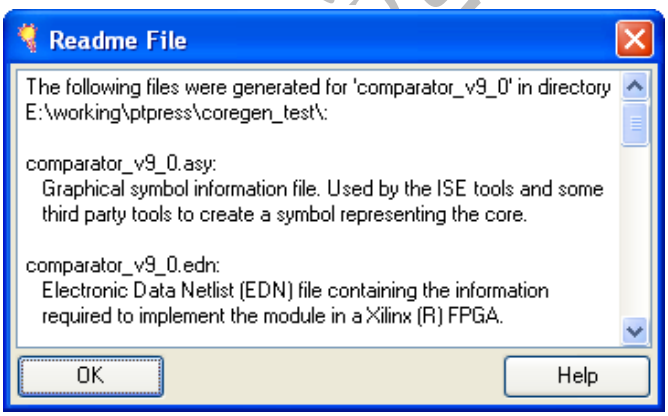

图 2.59　新建 IP 后报告信息

2.4.3　修改已有 IP 的参数的技巧

❖ 技巧内容

每个设计中使用的 IP 都会不同，设计者可以为每个设计新建 IP，也可以在已有 IP 的基础上修改参数而满足特定设计的需求。

❖ 技巧详解

修改已有 IP 的参数的技巧如下。

（1）启动 CORE Generator，打开已有项目。

（2）在打开的项目窗口中选择“Generated IP”选项卡，会出现用户 IP 列表，如图 2.60 所示。选择需要改变参数的 IP，然后单击右边窗口中的“Recustomize”，会弹出和新建 IP 的时候一样的参数设置对话框。

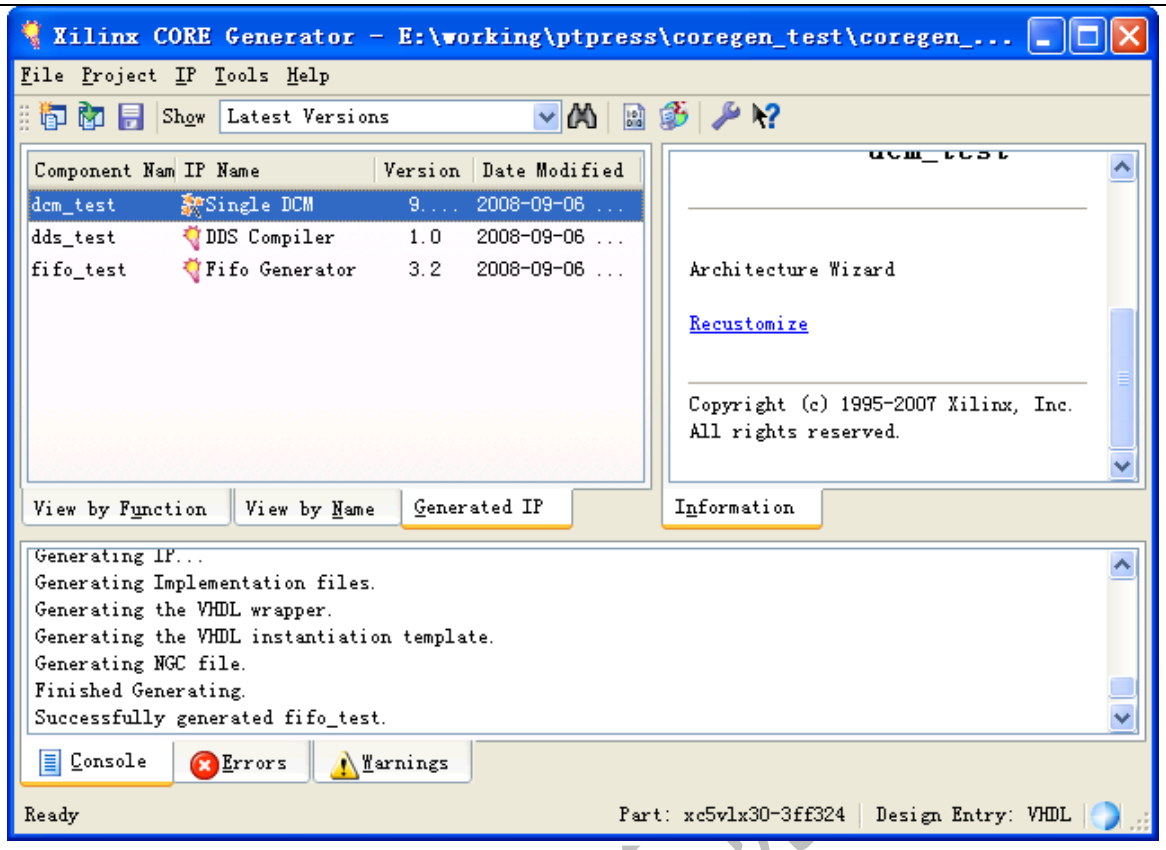

图 2.60　修改 IP 参数

（3）修改 IP 参数的方法和新建 IP 时设置 IP 参数的方法一样，设置好以后单击【Finish】按钮就可以了。

2.4.4　Architecture Wizard 使用技巧

❖ 技巧内容

Architecture Wizard 是 Xilinx 提供的另外一个用于生成 IP 的工具，主要用于生成时钟模块和 DSP 模块的 IP。

❖ 技巧详解

使用 Architecture Wizard 新建 IP 的技巧如下。

（1）单击“开始”菜单，单击 ISE 启动菜单下的【Accessories】/【Architecture Wizard】启动“Architecture Wizard”。

（2）在弹出的“Setup”对话框中输入 XAW 文件的路径和名称，指定综合工具和 FPGA 器件型号，如图 2.61 所示，设置好以后单击【OK】按钮。

（3）接下来会弹出“Selection”对话框，设计者可以选择需要新建的 IP。这里以 DCM 为例，选择“Single DCM_ADV”，单击【OK】按钮如图 2.62 所示。

（4） 接下来会弹出 IP 参数设置窗口，与从 CORE Generator 中启动参数设置的界面相同。完成后会生成相应的 HDL 文件。

（5） Architecture Wizard 是 ISE 中老版本中保留的 IP 设置工具，推荐在 CORE Generator 中新建 IP，这里只是简单介绍 Architecture Wizard 以便设计者在需要使用的时候能够正确使用。

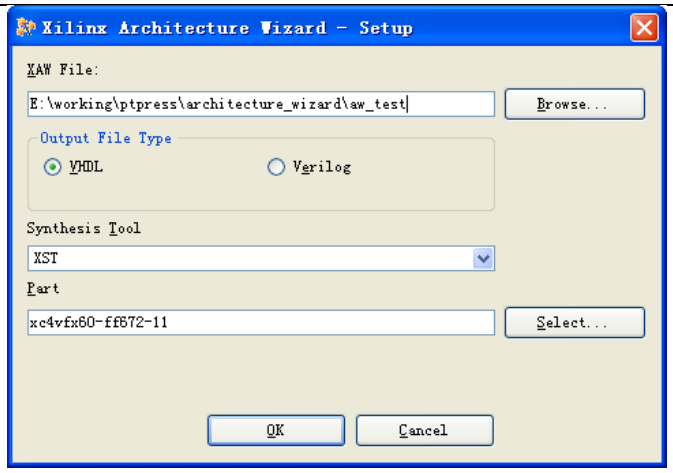

图 2.61　Architecture Wizard 设置对话框

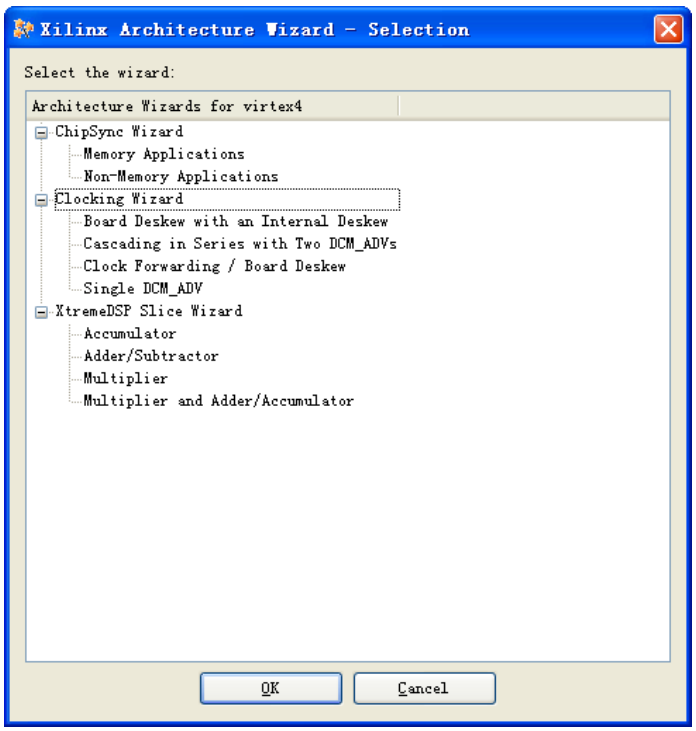

图 2.62　Architecture Wizard 选择 IP 类型

2.4.5　在设计中例化 IP 的技巧

❖ 技巧内容

新建好 IP 后，就需要在设计中使用 IP，设计者可以调用 IP 的 HDL 文件，如果 IP 为 FPGA 的硬核，ISE 在布局布线的时候能够识别，则生成的 IP 不包含网表文件，否则生成的 IP 就包含网表文件。

❖ 技巧详解

在设计中例化 IP 的技巧如下。

（1）在 CORE Generator 项目所在的目录下，会包含与用户新建 IP CORE 名字对应的 HDL 文件。如果是包含网表文件，则包含“NGC”格式的网表文件。

（2）如图 2.63 所示，DCM CORE 是 FPGA 的硬核，在布局布线的时候只需要知道 HDL 的接口就可以了，所以只有 HDL 文件，没有网表文件。而 DDS CORE 则需要知道 IP CORE 的内部实现的信息，所以包含了 NGC 格式的网表文件。

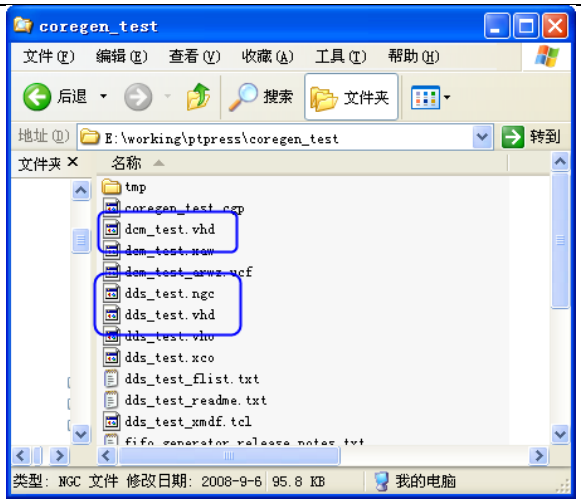

图 2.63　CORE Generator 生成的文件

（3）设计者只需要在自己的设计中例化生成的 IP CORE 的 HDL 文件就可以了，CORE Generator 还会生成一个例子文件，方便设计者例化，比如图 2.63 中所示的“dds_tset.vho”，只需要将该文件中相应的部分复制到设计文件中修改即可。

（4）VHO 文件的内容如下所示，设计者可以把元件声明部分的代码复制到自己的代码中，然后将端口匹配的文件复制到自己的代码中，做相应的修改即可。

```
-- The following code must appear in the VHDL architecture header:

------------ Begin Cut here for COMPONENT Declaration ------ COMP_TAG
component dds_test
    port (
    clk: IN std_logic;
    cosine: OUT std_logic_VECTOR(5 DOwnto 0);
    sine: OUT std_logic_VECTOR(5 DOwnto 0));
end component;

-- COMP_TAG_END ------ End COMPONENT Declaration ------------

-- The following code must appear in the VHDL architecture
-- body. Substitute your own instance name and net names.

------------ Begin Cut here for INSTANTIATION Template ----- INST_TAG
your_instance_name : dds_test
        port map (
            clk => clk,
            cosine => cosine,
            sine => sine);
-- INST_TAG_END ------ End INSTANTIATION Template ------------

-- You must compile the wrapper file dds_test.vhd when simulating
-- the core, dds_test. When compiling the wrapper file, be sure to
-- reference the XilinxCoreLib VHDL simulation library. For detailed
-- instructions, please refer to the "CORE Generator Help".
```

（5）在代码中例化好以后，需要将 IP CORE 对应的 HDL 文件和 NGC 网表复制到 ISE 工程目录下，以便进行综合与布局布线。

（6）也可以在 ISE 工程属性中指定网表文件的位置，如图 2.27 所示。

（7）最后进行综合、布局布线，生成下载文件。

2.4.6 选择不同版本 IP 的技巧

❖ 技巧内容

通常随着软件版本的升级和 FPGA 器件的更新，IP 也会更新，版本越新的 IP CORE，对应的功能也越强，但是如果只需要部分功能，或者说老版本的 IP CORE 已经经过测试，就希望使用老版本的 IP CORE。

❖ 技巧详解

选择不同版本 IP CORE 的技巧如下。

（1）启动 CORE Generator，打开已有项目，如图 2.64 所示。

（2）在打开的项目窗口中单击“Show”下拉菜单，选择“All Versions”，在“View by Function”或者“View by Name”选项卡下，可以看到所有可以使用的版本的 IP，设计者选择相应的版本，单击“Customize”设置参数。

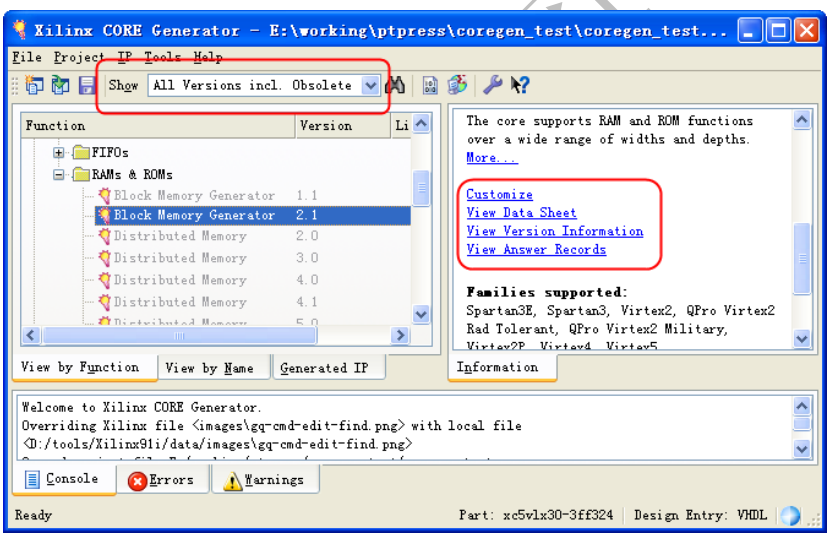

图 2.64 选择不同版本的 IP CORE 的方法

（3）接下来的设置方法和新建 IP 的方法一样。

2.5 使用 Xilinx 硬件资源的技巧

2.5.1 DCM 使用技巧

Xilinx FPGA 的时钟管理模块（CMT）提供非常灵活的高性能时钟控制。每个 CMT 包含两个 DCM（数字时钟管理器）和一个 PLL。

❖ 技巧内容

本节主要讲解 Xilinx 器件中 DCM（数字时钟管理器）的使用技巧。DCM 主要功能是对 FPGA 内的时钟进行管理，它所提供的时钟管理功能包括：时钟去歪斜、频率合成、相移和动态重配置。

在 Xilinx 的设计中，进入 FPGA 的时钟可以通过 DCM 进行相应的时钟处理再分配到各个逻辑，通过这样的时钟方案可以提高时钟的质量。

❖ 技巧详解

使用 DCM 的最简单的方法是调用 ISE 中的 Core Generator 生成相应的 IP。

（1）如图 2.65 所示，设置输入到 DCM 的时钟为 100MHz，该时钟为输入到 FPGA 的时钟，图中右边的信号为输出信号。

常用信号的详细介绍如下。

- CLKIN：源时钟输入引脚，提供 DCM 的时钟源。CLKIN 的范围必须在所使用的器件的规定范围内。
- RST：复位输入端口用于对 DCM 电路进行复位，它是高电平有效的异步复位信号。
- CLK0：输出一个频率与 DCM 的有效 CLKIN 频率相同的时钟。
- CLK90：提供一个与 DCM 的 CLK0 相移 90° 后同频的时钟。
- CLKDV：输出一个与 CLK0 相位对齐的时钟，频率是有效 CLKIN 频率的一个分频。该分频值由 CLKDV_DIVIDE 属性确定。
- CLK2X：提供一个与 CLK0 相位对齐的时钟，频率是 CLK0 的 2 倍。
- CLKFX：输出一个具有以下频率定义的时钟：CLKFX 频率=（M/D）×有效 CLKIN 频率。在此公式中，M 是倍频系数，D 是分频系数。
- LOCKED：指示 DCM 时钟输出是否有效，即是否输出正确的频率和相位，高电平有效。

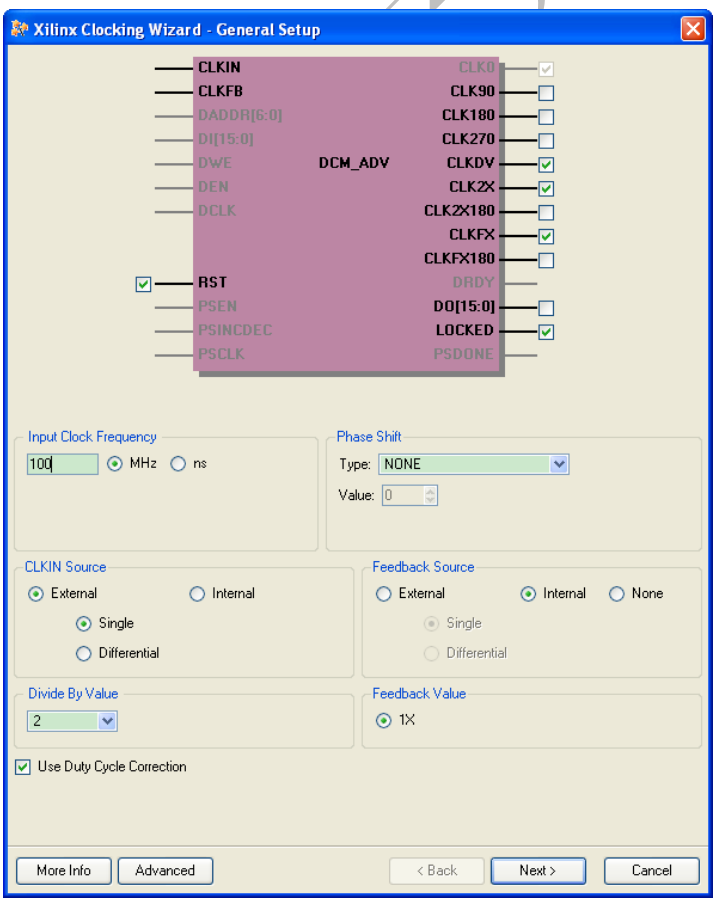

图 2.65　配置 DCM 参数的方法

（2）设置“Divide By Value”为 2，CLKDV 的输出就为 2 分频。保持其他设置为默认值，单击【Next】按钮。

（3）单击【Next】按钮直到出现图 2.66 所示的界面。在该界面中，可以直接使用需要的输出频率。如本例中需要输出 125MHz，经计算需要先 5 倍频再 4 分频；也可以计算后输入 M 参数和 D 参数。

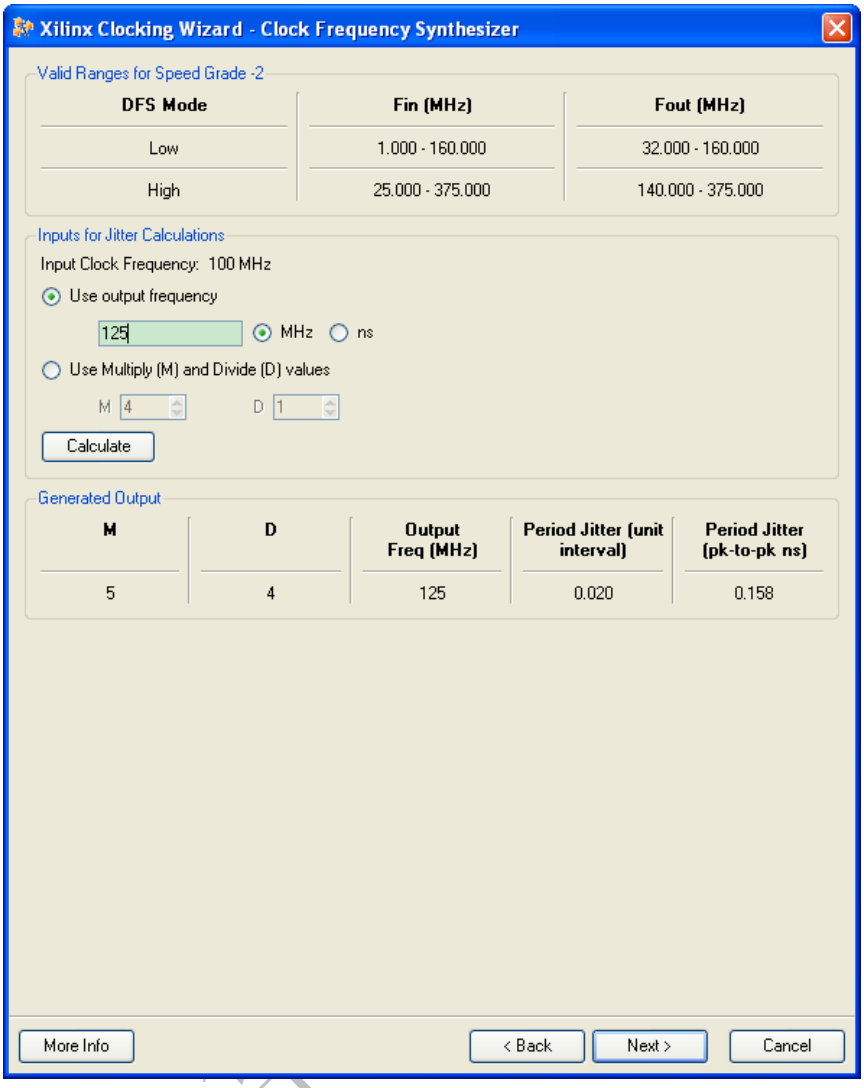

图 2.66　设置 CLKFX 输出频率界面

（4）单击【Next】按钮，然后单击【Finish】按钮完成生成 DCM。

（5）在 Core Generator 工程目录下，会生成一个 vhd 文件，在设计中可以直接例化该文件，输出用户需要的时钟。

2.5.2　PMCD 使用技巧

❖ 技巧内容

PMCD（Phase-Matched Clock Dividers）可以用来产生多个相位对齐的分频后的时钟，或者是延迟后的时钟。

❖ 技巧详解

图 2.67 是 PMCD 模块的接口模型，CLKA 是分频时钟的输入时钟，CLKB、CLKC 和 CLKD 是延迟时钟的输入时钟。

输出端的时钟 CLKA1、CLKA1D2、CLKA1D4 和 CLKA1D8 分别对应 CLKA 的一分频、二分频、四分频和八分频输出。CLKB1、CLKC1 和 CLKD1 分别为 CLKB、CLKC 和 CLKD 的延迟输出。

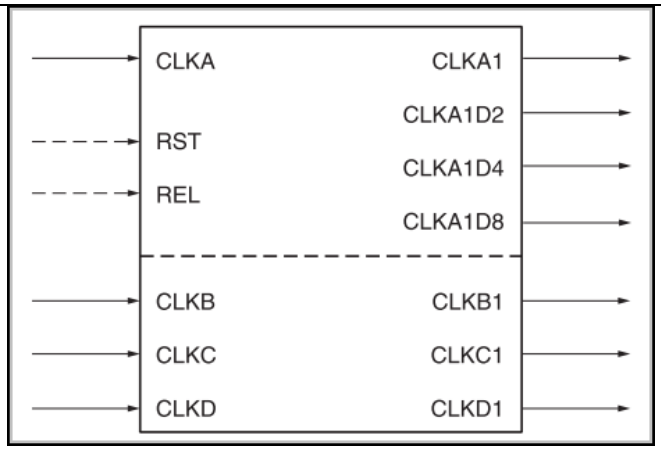

图 2.67　PMCD 模块接口

调用 PMCD 的方式如下。此处以 VHDL 代码为例，先对 PMCD 进行元件声明，然后进行类属参数对应和端口对应。

```
-- PMCD 元件声明
component PMCD
    generic(
        EN_REL          : boolean := FALSE;
        RST_DEASSERT_CLK : string  := "CLKA";
    );
    port(
        CLKA1     : out std_ulogic;
        CLKA1D2   : out std_ulogic;
        CLKA1D4   : out std_ulogic;
        CLKA1D8   : out std_ulogic;
        CLKB1     : out std_ulogic;
        CLKC1     : out std_ulogic;
        CLKD1     : out std_ulogic;
        CLKA      : in  std_ulogic;
        CLKB      : in  std_ulogic;
        CLKC      : in  std_ulogic;
        CLKD      : in  std_ulogic;
        REL       : in  std_ulogic;
        RST       : in  std_ulogic
    );
end component;

-- PMCD 元件例化
U_PMCD : PMCD
    Port map (
        CLKA1   => user_clka1,
        CLKA1D2 => user_clka1d2,
        CLKA1D4 => user_clka1d4,
        CLKA1D8 => user_clka1d8,
        CLKB1   => user_clkb1,
        CLKC1   => user_clkc1,
```

```
        CLKD1   => user_clkd1,
        CLKA    => user_clka,
        CLKB    => user_clkb,
        CLKC    => user_clkc,
        CLKD    => user_clkd,
        REL     => user_rel,
        RST     => user_rst );
```

如图 2.68 所示为 PMCD 分频端口的输出结果，可以看到输出的时钟信号相对于输入时钟都有一个相位延迟，但是输出信号之间是相位对齐的。

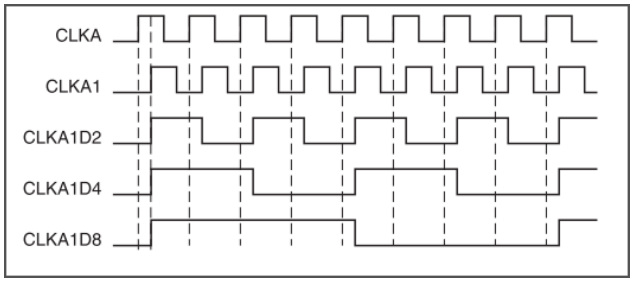

图 2.68　PMCD 分频输出结果

如图 2.69 所示为 PMCD 延迟输出的波形图，可以看到输出的 CLKB1、CLKC1 和 CLKD1 相对与 CLKA 都有一个相同的延迟，但是它们之间的相位关系保持不变。这对于需要保持相位关系的应用是非常有用的。

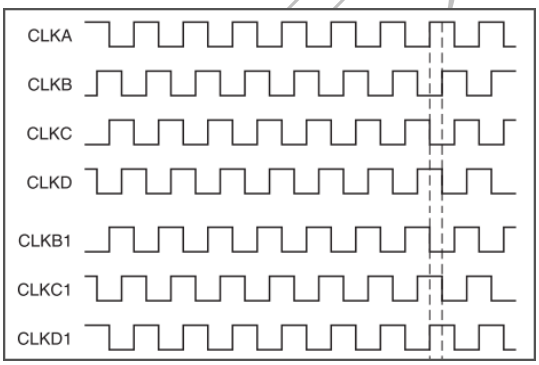

图 2.69　PMCD 延迟输出信号

2.5.3　BLOCK RAM 使用技巧

❖ 技巧内容

Xilinx 系列的 FPGA 提供了 BLOCK RAM 给用户，这些 BLOCK RAM 已经存在于 FPGA 中，使用这些专用的 BLOCK RAM 可以达到很高的读写速度，同时不会占用逻辑资源。
Virtex-4 系列 FPGA 中每个 BLOCK RAM 的大小为 18Kbit，BLOCK RAM 数量根据不同型号的 FPGA 而不同。

❖ 技巧详解

使用 BLOCK RAM 的技巧如下。
（1）可以用 CORE Generator 生成 BLOCK RAM，也可以使用推论 RAM 的方法，这里介绍使用 BLOCK RAM 的方法。
（2）启动 CORE Generator，打开已有项目，如图 2.70 所示。

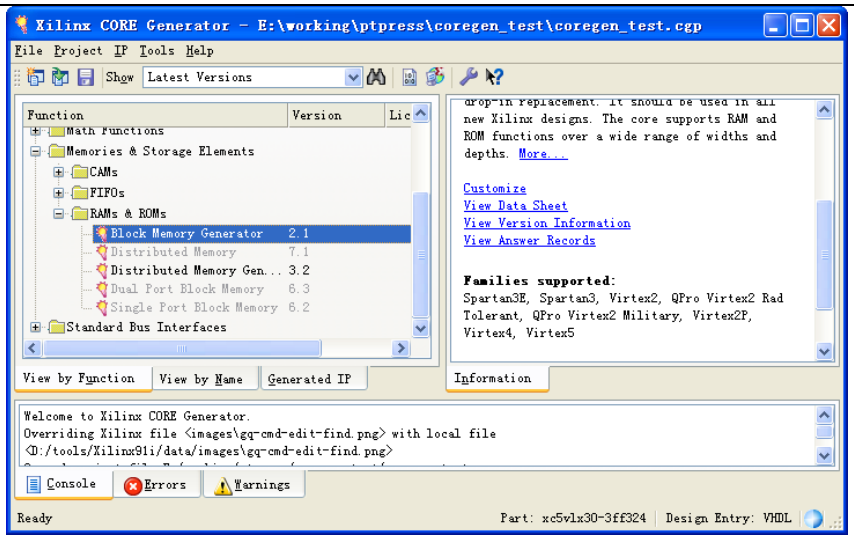

图 2.70　生成 BLOCK RAM 模块

（3） 单击“View by Function”选项卡，选择“Memories & Storage Elements”下的“RAMs & ROMs”，选择“Block Memory Generator”，单击右边“Information”窗口中的【Customize】，弹出 RAM 参数设置窗口。

（4）如图 2.71 所示为 BLOCK RAM 参数设置窗口。

- Memory Type：设计者可以选择是单口 RAM、双口 RAM 或者是 ROM。
- Write Enable：可以使用字节写使能信号来控制写操作。
- Algorithm：用来控制对 RAM 进行配置，18Kbit 的 RAM 可以配置成 16K×1、8K×2、4K×4、2K×9、1K×18、512×36 几种模式，用户根据设计中需要的数据位宽来进行配置，比如数据宽度为 8bit，就可以选择 2K×9 配置，一个 BLOCK RAM 可以存储 2KB 的　　数据。

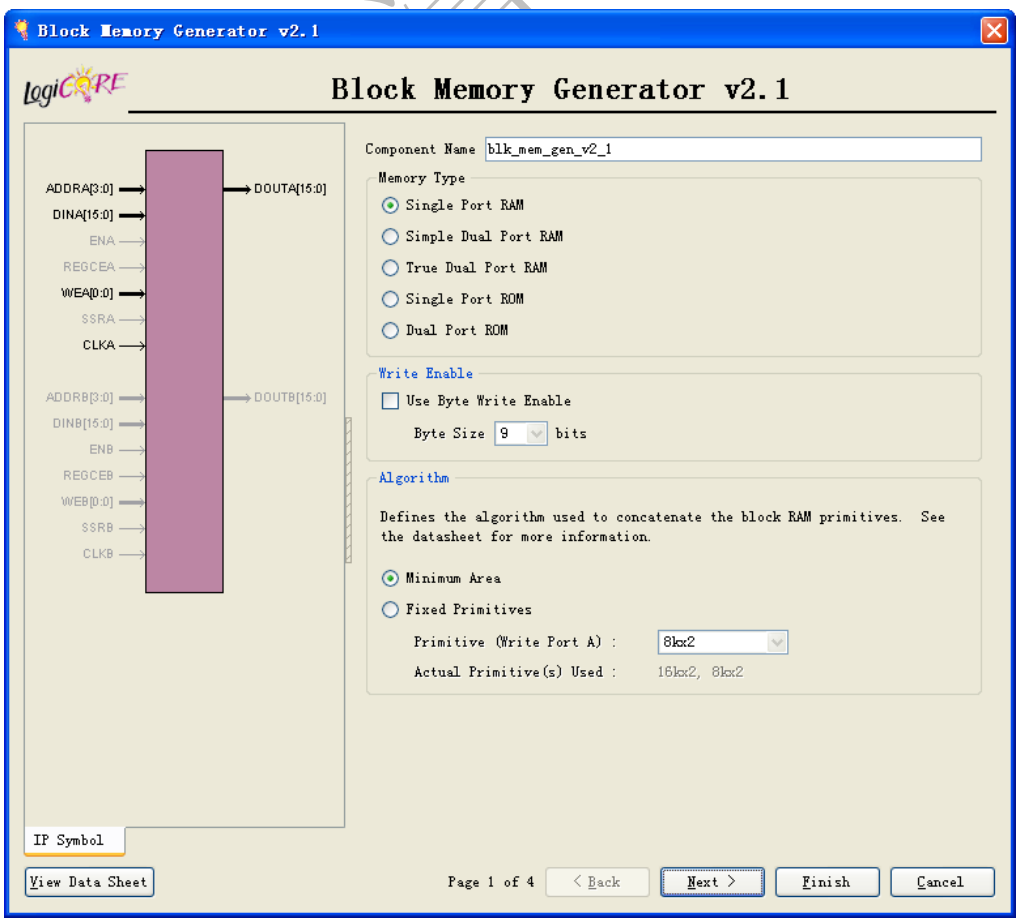

图 2.71　BLOCK RAM 参数设置窗口

（5）单击【Next】按钮，可以配置数据输入宽度和 RAM 大小，如果一个 BLOCK RAM 不够存储数据，软件会自动选择将多个 BLOCK RAM 进行级联。比如希望的 RAM 大小为 4KB，那么数据宽度为 8bit，需要两块 BLOCK RAM 级联才能存储 4KB。

（6）设置好 RAM 大小后，单击【Next】按钮，进入下一个设置窗口，如图 2.72 所示。如果选择了给输出端口增加寄存器，可以增加 RAM 的读写速度，但是输出会增加一个时钟周期的延迟。

比如说没有增加输出延迟的时候，读写时钟只能达到 100MHz。加入输出寄存器后，读写时钟能够达到 120MHz，但是输出数据相对与读地址的变化会多延迟一个时钟　　周期。

（7）单击【Next】，在出现的窗口中单击【Finish】完成 BLOCK RAM 参数设置，完成后，会弹出一个报告窗口，如图 2.73 所示。

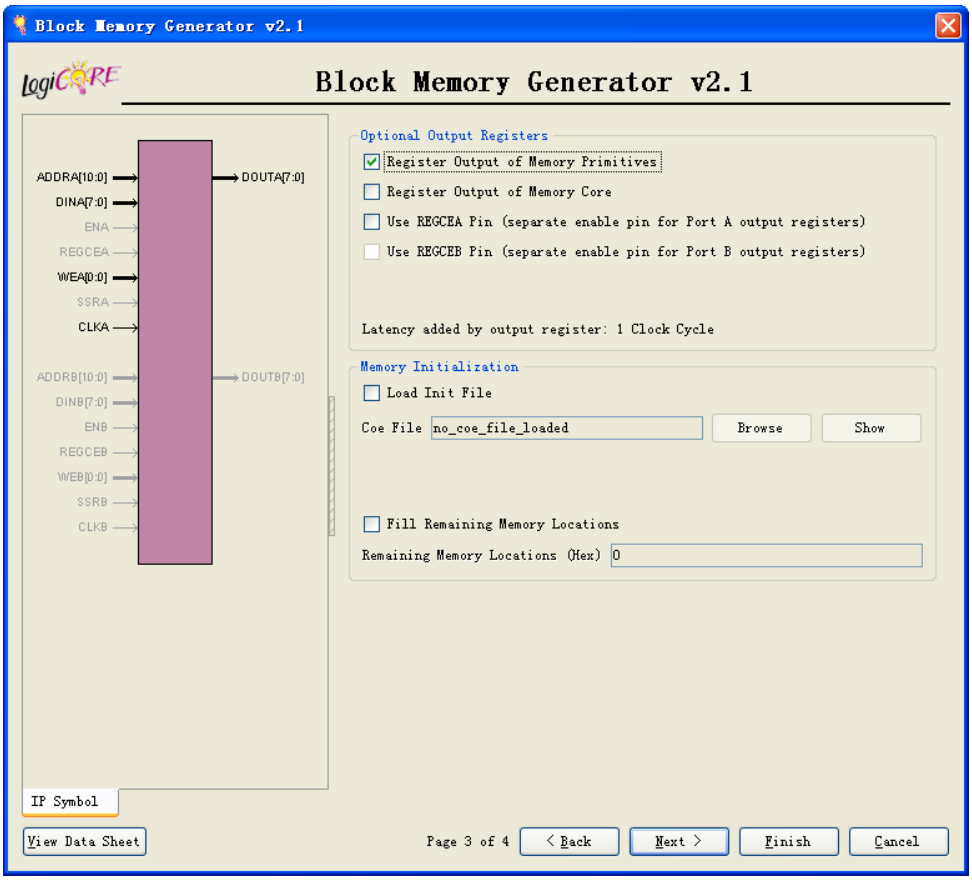

图 2.72　设置 BLOCK RAM 输出寄存器

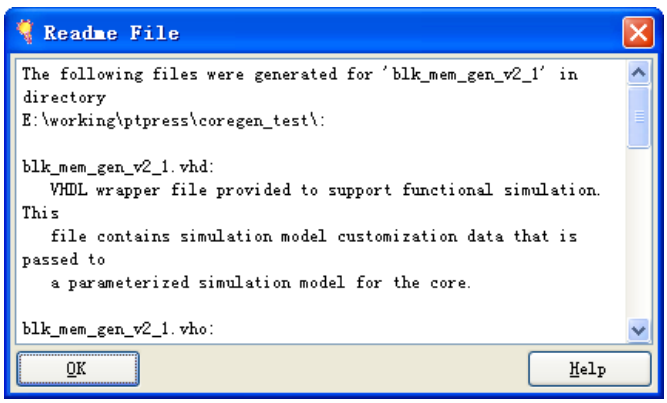

图 2.73　生成 BLOCK RAM 报告窗口

（8）在 BLOCK RAM 对应的 CORE Generator 工程目录下，会生成 HDL 文件。设计者需要在自己的设计中例化该文件，和 HDL 文件同名的 VHO 文件给出了例化元件的方法，设计者可以复制 VHO 文件中的代码到自己的代码中进行修改。

（9）在布局布线的时候，需要在 ISE 中指定 BLOCK RAM 对应的 NGC 网表文件。

2.5.4 分布式 RAM 使用技巧

❖ 技巧内容

当设计者使用的 RAM 比较小的时候，使用 BLOCK RAM 会浪费片内存储空间。因为不管用户 RAM 有多小，都会至少占用一块 BLOCK RAM。这种情况下，可以考虑使用分布式 RAM。

分布式 RAM 是利用 FPGA 内部的逻辑资源来实现 RAM 功能，所以设计者要保证片内有足够的逻辑资源来实现分布式 RAM。

❖ 技巧详解

使用分布式 RAM 的技巧如下。

（1）可以用 CORE Generator 生成分布式 RAM。启动 CORE Generator，打开已有项目，如图 2.74 所示。

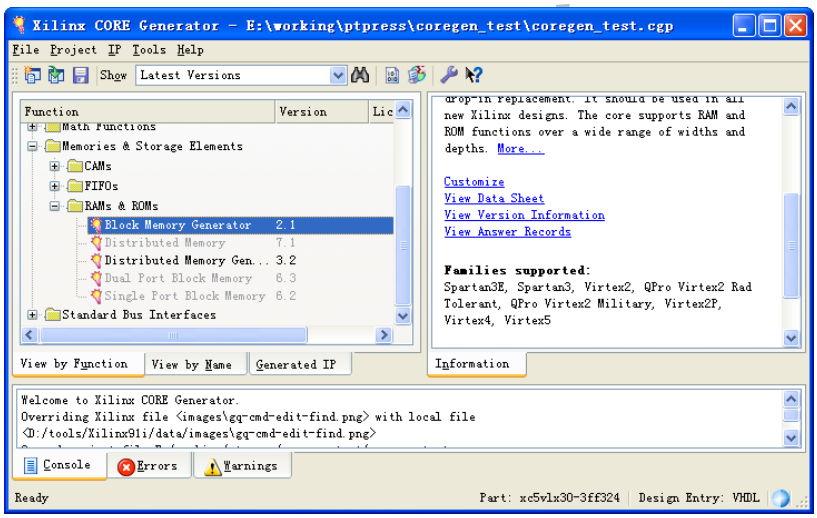

图 2.74 生成 BLOCK RAM 模块

（2）单击“View by Function”选项卡，选择“Memories & Storage Elements”下的“RAMs & ROMs”，选择“Distributed Memory Generator”，单击右边“Information”窗口中的【Customize】，弹出分布式 RAM 参数设置窗口。

（3）类似 BLOCK RAM 参数设置的方法，设计者设置好分布式 RAM 的参数后即可生成分布式 RAM 的 HDL 文件和网表文件。

2.5.5 FIFO 使用技巧

❖ 技巧内容

FIFO 为先入先出存储器，在很多应用中都会使用到 FIFO，比如缓存数据、时钟域转换。和 RAM 不同的是，FIFO 不需要关心读写地址，可以简化操作。

❖ 技巧详解

使用 FIFO 的技巧如下。

（1）启动 CORE Generator，打开已有项目。

（2）单击“View by Function”选项卡，选择“Memories & Storage Elements”下的“FIFOs”，选择“FIFO Generator”，单击右边“Information”窗口中的【Customize】按钮，弹出 FIFO 参数设置窗口。

（3）如图 2.75 所示，设置 FIFO 的参数有以下两种。

- Read/Write Clock Domains：时钟域设置，可以设置读写是在单时钟域或者读写时钟为异步时钟。
- Memory Type：选择 FIFO 中使用的存储器的类型，可以选择使用 Block RAM 或者分布式 RAM。

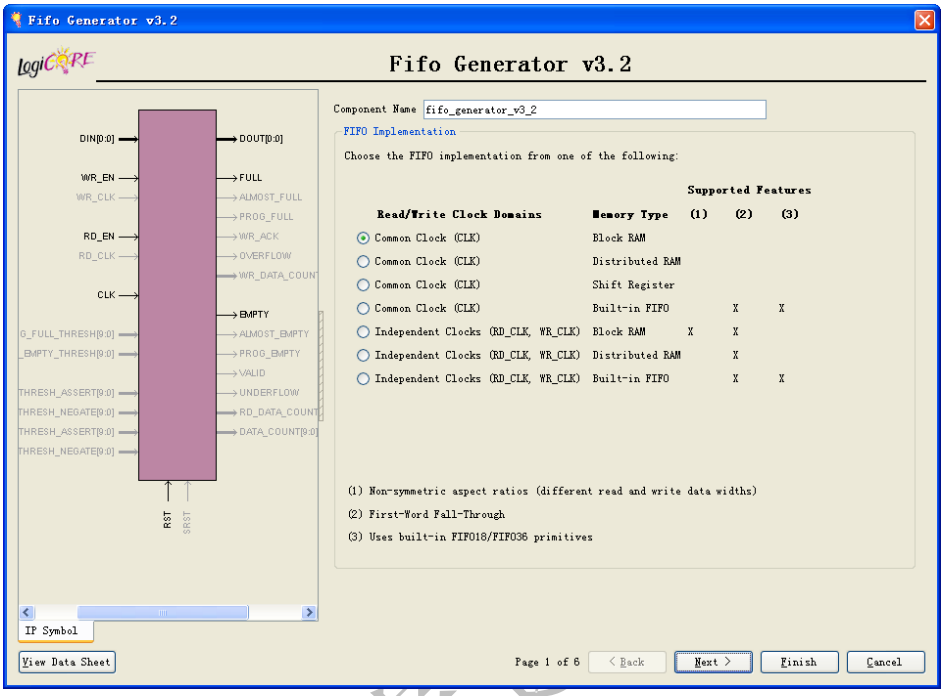

图 2.75　设置 FIFO 的时钟域和存储类型

（4）Xilinx 的 FIFO 模块提供了很多端口供用户使用，这里介绍几个常用的端口，如表 2.1 所示。如果设计者需要使用其他端口，可以参考 FIFO 的数据手册。

表 2.1　　FIFO 端口列表

端　口	方　向	功 能 介 绍
DIN	IN	数据输入
WR_EN	IN	写使能。当 WR_EN = 1 时，将数据写入存储器。当 WR_EN = 0 时，禁止写操作

续表

端　口	方　向	功 能 介 绍
WR_CLK	IN	写操作的时钟。如果读写时钟异步，写操作工作在该时钟域下
RD_EN	IN	读使能。当 RDEN = 1 时，数据读出到输出寄存器。当 RDEN = 0 时，禁止读操作
RD_CLK	IN	读操作的时钟。如果读写时钟异步，读操作工作在该时钟域下
CLK	IN	读写工作在同一时钟域的时候，读写操作都工作在该时钟域
RST	IN	所有 FIFO 功能、标记和指针的异步复位。RESET 必须置为在 3 个时钟周期内有效
DOUT	OUT	数据输出
FULL	OUT	FIFO 存储器的所有字条均已填满。不再接受写操作。与 WR_CLK 同步

ALMOST_FULL	OUT	几乎 FIFO 存储器的所有字条均已填满。与 WR_CLK 同步。此标记的偏移可由用户配置
EMPTY	OUT	FIFO 空出。不再接受读操作。与 RD_CLK 同步
ALMOST_EMPTY	OUT	几乎 FIFO 存储器的所有有效字条均已读取。与 RDCLK 同步。此标记的偏移可由用户配置

（5）设置好 FIFO 的时钟域后，单击【Next】按钮，可以设置 FIFO 的读写宽度，存储深度。
（6）接下来设置是否需要 ALMOST_FULL 和 ALMOST_EMPTY 端口、其他的参数可以使用默认值。最后单击【Finish】按钮，生成 FIFO 模块。
（7）在 FIFO 对应的 CORE Generator 工程目录下，会生成 HDL 文件，设计者需要在自己的设计中例化该文件。和 HDL 文件同名的 VHO 文件给出了例化元件的方法，设计者可以复制 VHO 文件中的代码到自己的代码中进行修改。
（8）在布局布线的时候，需要在 ISE 中指定 FIFO 对应的 NGC 网表文件。

2.5.6 IDDR 使用技巧

❖ 技巧内容

Xilinx 器件中有专用寄存器来实现输入双倍数据速率（DDR）寄存器。可以通过例化 IDDR 基元来使用此功能。
IDDR 基元只有一个时钟输入。下降沿数据由输入时钟的反转信号进行时钟控制。

❖ 技巧详解

IDDR 有以下 3 种操作模式。
- OPPOSITE_EDGE 模式；
- SAME_EDGE 模式；
- SAME_EDGE_PIPELINED 模式。

（1）OPPOSITE_EDGE 模式下，数据在时钟上升沿通过输出 Q1 以及在时钟下降沿通过输出 Q2 送至内部资源。
图 2.76 是 OPPOSITE_EDGE 模式下 IDDR 的原理图，在该模式下使用了两个触发器，输入数据 D 接到两个触发器的输入端，时钟信号及其反转信号分别接到两个触发器的时钟输入端。
因为 D 触发器是工作在时钟上升沿，因而在时钟上升沿会将数据送到 Q1，在时钟下降沿把数据送到 Q2，实现在时钟的上升沿和下降沿都采样数据。

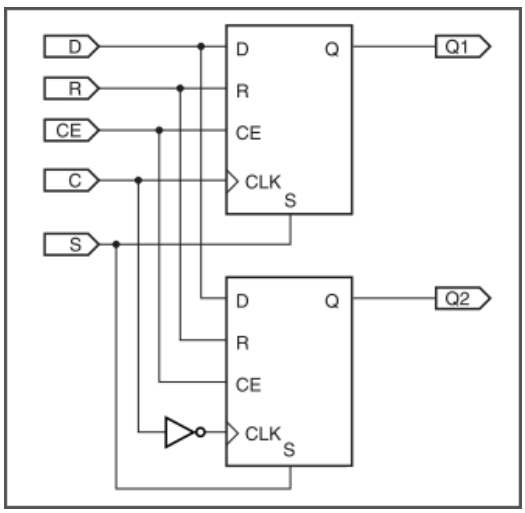

图 2.76　OPPOSITE_EDGE 模式下 IDDR 原理图

图 2.77 所示是 IDDR 的时序，可以看出在时钟的上升沿将数据 D0A 送到 Q1，在时钟的下降沿将数据 D1A 送到 Q2。Q1 和 Q2 的输出都是直接送出，没有经过任何触　　发器。

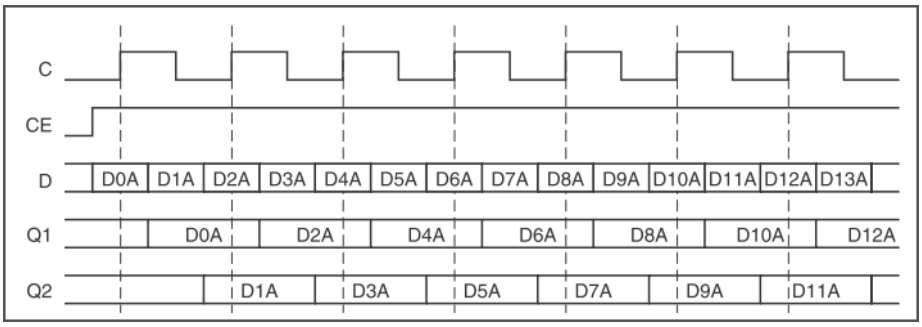

图 2.77　OPPOSITE_EDGE 模式下 IDDR 时序图

（2）SAME_EDGE 模式下，数据在同一时钟沿上送至 FPGA 内部资源。一对数据之间要有一个时钟周期的间隔。

图 2.78 所示为 SAME_EDGE 模式下 IDDR 的原理图，在 Q2 的输出端加入了一级寄存器，这样 Q2 的输出信号也会同步到时钟信号的上升沿。

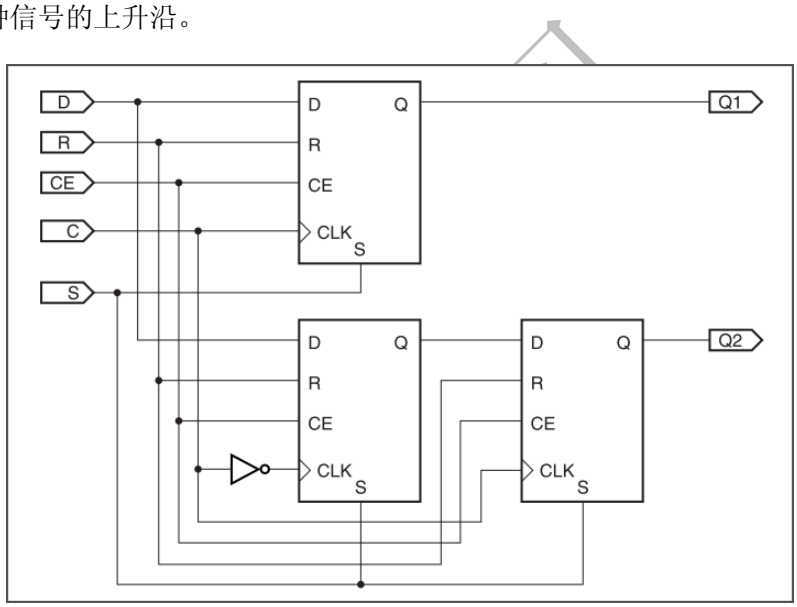

图 2.78　SAME_EDGE 模式下 IDDR 原理图

图 2.79 所示为 SAME_EDGE 模式下 IDDR 的时序图，可以看出 Q1 就是时钟上升沿采到的数据，Q2 输出为时钟下降沿采到的数据，但是被同步到了时钟的上升沿才　　输出。

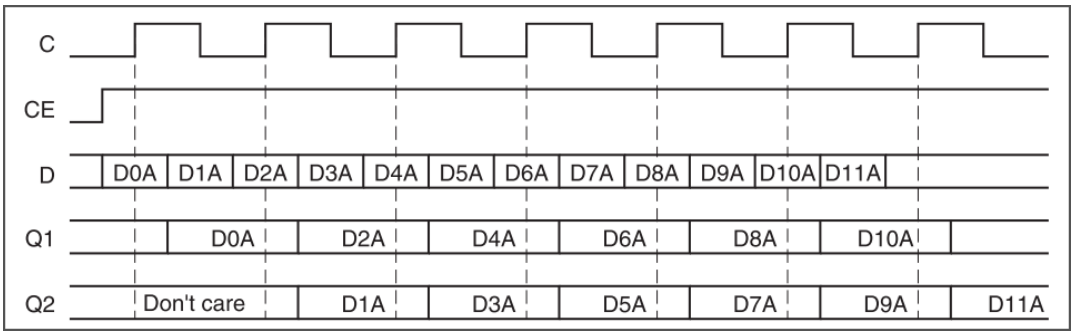

图 2.79　SAME_EDGE 模式下 IDDR 时序图

（3）SAME_EDGE_PIPELINED 模式下，数据在同一时钟沿上送至 FPGA 内部资源。与 SAME_EDGE 模式不同的是，数据没有一个时钟周期的间隔。需要增加一个时钟延迟，以消除 SAME_EDGE 模式的间隔效果。

图 2.80 所示为 SAME_EDGE_PIPELINED 模式下 IDDR 的原理图，可以看出在 Q1 和 Q2 都多加入了一级寄存器，这样 Q1 和 Q2 的输出都被同步到时钟的上升沿。

图 2.81 所示为 SAME_EDGE_PIPELINED 模式下 IDDR 的时序图，可以看到 Q1 是时钟上升沿采到的数据，Q2 输出为时钟下降沿采到的数据。Q1 和 Q2 的输出经过触发器后都被同步到时钟上升沿。

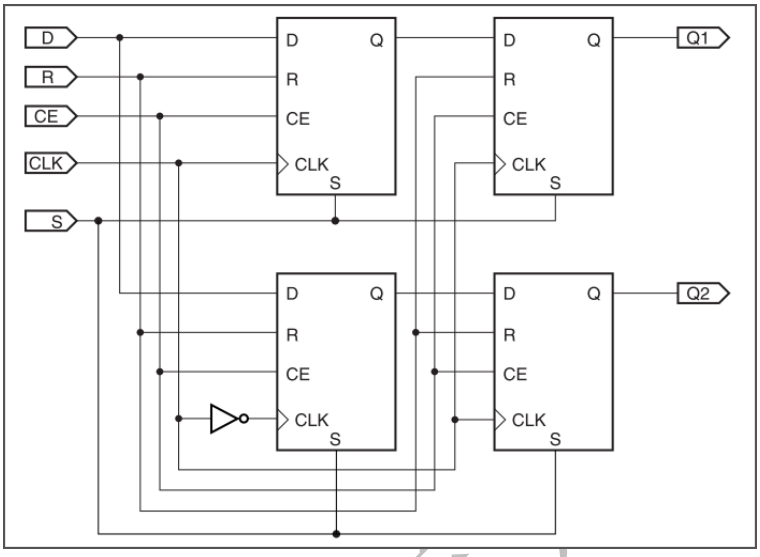

图 2.80　SAME_EDGE_PIPELINED 模式下 IDDR 的原理图

该方法对于 FPGA 内部逻辑使用 Q1 和 Q2 的数据而言比较方便，因为同一个时钟上升沿可以采到输入端 D 的上升沿数据和下降沿数据。

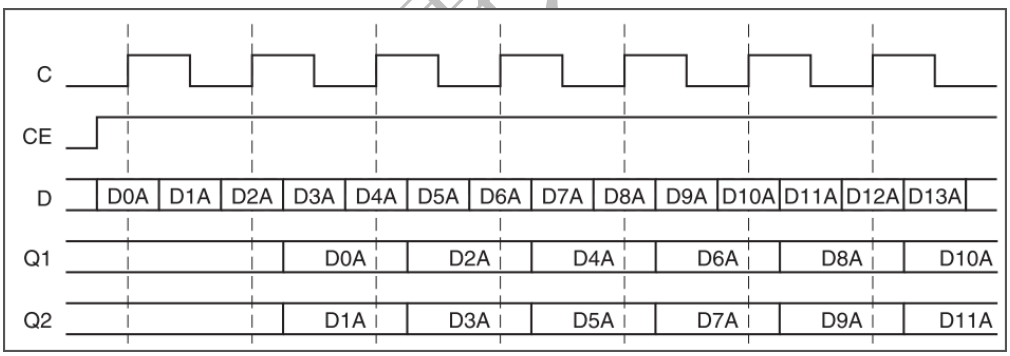

图 2.81　SAME_EDGE_PIPELINED 模式下 IDDR 的时序图

表 2.2 和表 2.3 给出了 IDDR 模板的端口定义和类属定义。

表 2.2　　IDDR 模板的端口定义

端　口	方　向	功 能 介 绍
Q1	OUT	数据输出，为上升沿数据
Q2	OUT	数据输出，为下降沿数据
C	IN	时钟输入引脚
CE	IN	时钟使能，高电平有效
D	IN	IDDR 的数据输入，时钟上升沿下降沿都有数据
R	IN	复位引脚，复位置为高电平有效
S	IN	置引脚，设置置为高电平有效

表 2.3 IDDR 模板的 Generic 类属定义

类 属 名 称	功 能 介 绍	赋 值
DDR_CLK_EDGE	设置相对于时钟沿的 IDDR 操作模式	OPPOSITE_EDGE（默认）、SAME_EDGE、SAME_EDGE_PIPELINED
INIT_Q1	设置 Q1 端口的初始值	0（默认）、1
INIT_Q2	设置 Q2 端口的初始值	0（默认）、1
SRTYPE	相对于时钟（C）的置位/复位类型	ASYNC（默认，异步复位）、SYNC（同步复位）

下面给出 IDDR 的 VHDL 模板代码，Verilog HDL 的模板代码可以根据 VHDL 代码修改。设计者只需要在自己的代码中加入以下的元件声明语句和元件例化语句就可以了。

```
-- IDDR 的 VHDL 模板代码
-- IDDR 元件声明
component IDDR
  generic(
     DDR_CLK_EDGE : string := "OPPOSITE_EDGE";
     INIT_Q1      : bit    := '0';
     INIT_Q2      : bit    := '0';
     SRTYPE       : string := "SYNC";
    );
  port(
     Q1         : out std_ulogic;
     Q2         : out std_ulogic;
     C          : in  std_ulogic;
     CE         : in  std_ulogic;
     D          : in  std_ulogic;
     R          : in  std_ulogic;
     S          : in  std_ulogic
    );
  end component;
-- IDDR 元件例化
U_IDDR : IDDR
Port map(
     Q1 => user_q1,
     Q2 => user_q2,
     C => user_c,
     CE => user_ce,
     D => user_d,
     R => user_r,
     S => user_s
    );
```

2.5.7 ODDR 使用技巧

❖ 技巧内容

Xilinx 器件中有专用寄存器来实现 DDR 双倍数据速率输出寄存器。例化 ODDR 基元时可使用此功能。当使用 OLOGIC 时，DDR 自动为多路复用。无需手动控制多路器的选择。ODDR 基元只有一个时钟输入。下降沿数据由输入时钟局部反转后进行时钟控制。

❖ 技巧详解

ODDR 有以下两种操作模式：

- OPPOSITE_EDGE 模式；
- SAME_EDGE 模式。

（1）在 OPPOSITE_EDGE 模式下，使用时钟（CLK）的两个沿以两倍吞吐量从 FPGA 内部资源采集数据。图 2.82 所示为 OPPOSITE_EDGE 模式下 ODDR 的原理图，时钟 CLK 直接连接到 D1 的时钟输入，时钟 CLK 的反相信号连接到 D2 的时钟输入。

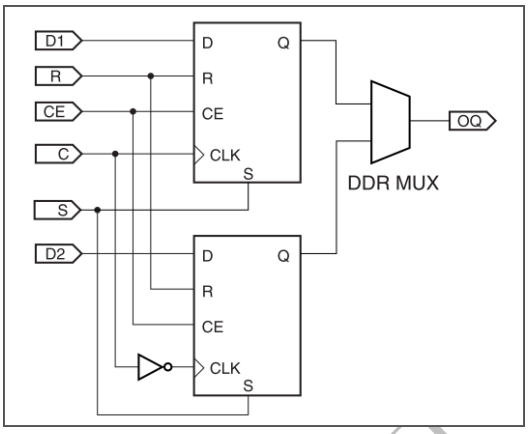

图 2.82　OPPOSITE_EDGE 模式下 ODDR 原理图

图 2.83 所示为 OPPOSITE_EDGE 模式下 ODDR 的时序图，时钟上升沿将 D1 端的数据送到多路复用器，时钟的下降沿将 D2 端的数据送到多路复用器，通过控制复用器选择端口实现双倍速率输出。

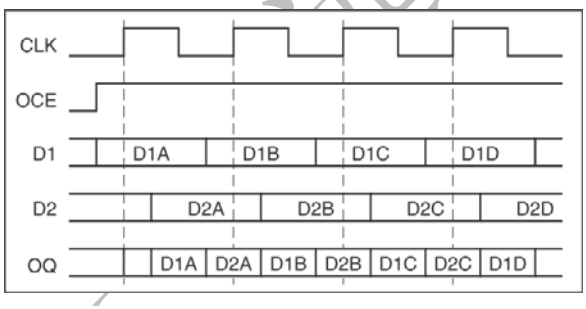

图 2.83　OPPOSITE_EDGE 模式下 ODDR 时序图

（2）在 SAME_EDGE 模式下，数据可以在同一时钟沿上送至 IOB。与使用 CLB 寄存器相比，在同一时钟沿上将数据送至 IOB 可以避免违反建立时间，并且使用户能够以尽量短的寄存器到寄存器延迟实现较高的 DDR 频率。

图 2.84 所示为 SAME_EDGE 模式下 ODDR 的原理图，采用了 3 个触发器。

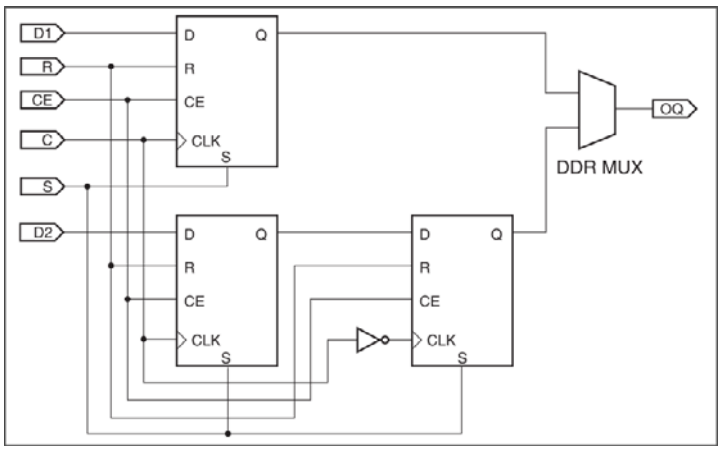

图 2.84　SAME_EDGE 模式下 ODDR 的原理图

图 2.85 所示为 SAME_EDGE 模式下 ODDR 的时序图。

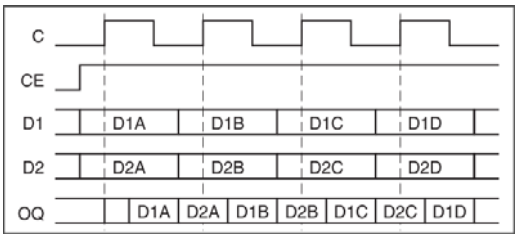

图 2.85　SAME_EDGE 模式下 ODDR 的时序图

在时钟上升沿，同时将 D1 和 D2 采样，D1 送到复用器的输入端，D2 经另外一个触发器，在时钟的下降沿采样，送到复用器的输入端。

由于采用了 3 个触发器，D1 和 D2 的数据可以同时在时钟的上升沿送到 IOB，这样用户就不用太关心 ODDR 内部的实现，只需要将数据按照正常方式送到 D1 和 D2 即可。

如下是 ODDR 的 VHDL 模板，Verilog HDL 的模板与之类似，读者可以根据 VHDL 自行修改成 Verilog HDL 模板。

读者可以将以下的元件声明语句复制到用户代码的声明部分，并且在代码的 Architecture 部分例化元件、匹配端口和类属参数即可。

```
-- ODDR 元件声明
component ODDR
  generic(
     DDR_CLK_EDGE : string := "OPPOSITE_EDGE";
     INIT        : bit    := '0';
     SRTYPE      : string := "SYNC";
    );
  port(
     Q           : out std_ulogic;
     C           : in  std_ulogic;
     CE          : in  std_ulogic;
     D1          : in  std_ulogic;
     D2          : in  std_ulogic;
     R           : in  std_ulogic;
     S           : in  std_ulogic
    );
  end component;
-- ODDR 元件例化
U_ODDR : ODDR
Port map(
     Q => user_q,
     C => user_c,
     CE => user_ce,
     D1 => user_d1,
     D2 => user_d2,
     R => user_r,
     S => user_s
    );
```

表 2.4 所示为 ODDR 的端口定义，表 2.5 所示为 ODDR 的类属定义。

表 2.4　　ODDR 模板的端口定义

端　口	方　向	功 能 介 绍

Q	OUT	数据输出，为双倍速率数据
C	IN	时钟输入引脚
CE	IN	时钟使能，高电平有效输出数据到 Q
D1	IN	ODDR 的数据输入，时钟上升沿数据
D2	IN	ODDR 的数据输入，时钟下降沿数据
R	IN	复位引脚，复位置为高电平有效
S	IN	置引脚，设置置为高电平有效

表 2.5　　ODDR 模板的类属参数定义

类 属 名 称	功 能 介 绍	赋　　值
DDR_CLK_EDGE	设置相对于时钟沿的 ODDR 操作模式	OPPOSITE_EDGE（默认）、SAME_EDGE
INIT	设置 Q 端口的初始值	0（默认）、1
SRTYPE	相对于时钟（C）的置位/复位类型	ASYN、SYNC（默认，同步复位）

2.6　设置约束的技巧

在设计数字电路时，通常需要对设计施加约束，以便控制综合，实现满足速度、面积和引脚位置等方面的要求。
设置约束常用的方法是编写约束文件，综合与实现工具会使用该约束文件指导逻辑映射和布局布线。
本节主要介绍设置约束的技巧，包括约束文件（UCF）的使用方法、常用约束属性的使用方法等。

2.6.1　使用 UCF 文件的技巧

❖ 技巧内容

用户约束文件 UCF（User Constraint File）是标准格式的约束文件，UCF 是 ASCII 格式的文件，其中指定了逻辑设计的约束。这些约束用于指导如何在 FPGA 中实现设计。

❖ 技巧详解

可以在图形界面和命令行方式下使用 UCF 文件，分别介绍如下。
（1）在图形界面下使用 UCF，需要在 ISE 中添加 UCF 文件如图 2.86 所示。选择【Project】/【Add Source】添加 UCF 文件，在布局布线的时候就会使用 UCF 中指定的约束来实现 FPGA。

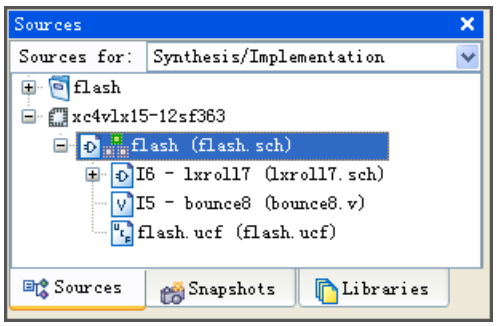

图 2.86　在 ISE 中添加 UCF 文件

（2）在命令行方式下，需要在 NGDBUILD 命令中指定 UCF 文件，以下程序给出了在命令行方式下加入 UCF 文件的方法。

```
    在本例中，选用的器件是 Virtex-5 系列的 xc5vlx50t-ff1136-1
    临时目录为 d:/test/temp
    搜索路径为 d:/test/macro
    输入的网表文件为 test_top.edf
    输出的 NGD 文件为 test_top_ngdbuild.ngd
    ngdbuild -p xc5vlx50t-ff1136-1 -dd d:/test/temp -sd d:/test/macro -uc d:/test/test_top.ucf
test_top.edf test_top_ngdbuild.ngd
```

UCF 文件是 NGDBUILD 的输入文件（在图形界面中对应 Translate）。约束文件的信息会加入到 NGD 网表文件中，在接下来的 MAP 和 PAR 操作中会用到这些约束信息，设计者只需要在 UCF 文件中设置好用户约束就可以了。

2.6.2 编写 UCF 文件的语法技巧

❖ 技巧内容

UCF 文件作为标准的约束文件，需要按照一定的语法规则来编写，这里首先介绍 UCF 的通用语法，在随后的各节中介绍具体的约束条件。

❖ 技巧详解

以下是一些 UCF 文件的常用语法规则。

- UCF 文件对用户对象名的大小写也是敏感的，也就是说，在 UCF 文件中出现的对象名称（比如信号名）必须和设计中出现的名称完全一样。
- Xilinx 定义的约束关键字不是大小写敏感的，比如位置约束关键字 LOC、周期约束关键字 PERIOD 都不是大小写敏感的，可以写成 loc、Loc。但是不推荐大小写混合使用，一个关键字要统一使用大写或者小写，方便阅读。
- 每条约束语句都必须以分号作为结束标志。
- 如果一条约束语句太长，可以分为多行书写，行于行之间不需要连接符号，因为每条约束语句是以分号作为结束标志。
- 约束文件中的注释语句以井号（#）作为开头。
- UCF 文件中的语句之间没有顺序要求，可以使用任意顺序来组织各约束语句。

下面给出一个 UCF 文件的简单例子，读者可以通过这个例子对 UCF 文件有个初步的了解，具体的约束语句会在随后的章节详细介绍。

```
    ########################################
    ## UCF 实例文件，设计者可以参考该实例编写自己的 UCF 文件
    ########################################
    Net sys_clk TNM_NET = sys_clk_pin;
    Net sys_clk LOC = AH15;
    Net sys_clk IOSTANDARD=LVCMOS33;
    Net sys_rst LOC = E9;
    Net sys_rst IOSTANDARD=LVCMOS33;
    Net sys_rst PULLUP;
    ## System level constraints
    Net sys_clk TNM_NET = sys_clk;
    TIMESPEC TS_sys_clk = PERIOD sys_clk 10000 ps;
    Net sys_rst TIG;
    ## IO Devices constraints
    #### Module Uart_1 constraints
    Net Uart_RX_pin LOC = AG15;
    Net Uart_RX_pin IOSTANDARD=LVCMOS33;
    Net Uart_TX_pin LOC = AG20;
    Net Uart_TX_pin IOSTANDARD=LVCMOS33;
```

2.6.3 设置 TNM_NET 分组约束的技巧

❖ 技巧内容

TNM_NET 用于设置分组约束，将同一网络（NET）上的元件作为命名组的一部分。

❖ 技巧详解

TNM_NET 将同一网络（NET）上的元件作为命名组的一部分，可以对该分组进行时序约束，该分组内的元件都受到同一时序约束。
TNM_NET 与 TNM 约束类似，不过 TNM 是将同一网络上的 PAD 进行分组，而不是元件。
TNM_NET 的语法如下：

```
NET "netname"TNM_Net=identifier;
```

- netname：网络的名字。
- identifier：由字母、数字和下划线组成的分组的标志符，用来区分各个分组，对该分组的约束只要加到该标志符上就可以了。

例如要对时钟网络“sys_clk”上的元件设置为命名组的一部分，如图 2.87 所示，将 DFF1 和 DFF2 设置为同一分组，可以使用如下的约束语句：

```
NET sys_clk TNM_NET = sys_clk_group;
```

这样，对 DFF1 和 DFF2 的时钟约束，只需要加在“sys_clk_group”这个分组上就可以了。

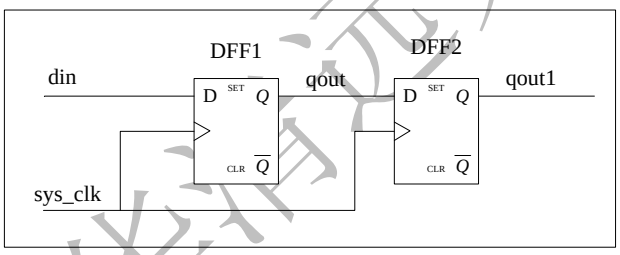

图 2.87 设置 TNM_NET 约束

2.6.4 设置 TNM 分组约束的技巧

❖ 技巧内容

使用 TNM 可以选出一组元件，构成一个分组，并赋予这个分组一个名字，以后就可以在该分组上添加时序约束。

❖ 技巧详解

TNM（pronounced tee-name）是一个标志，可以附加在指定的网络、元件引脚、基元和宏上，关键字 RISING 和 FALLING 也可以与 TNM 同时使用，所有被指定了 TNM 约束的对象被认为是一个分组。
TNM 的语法为：

```
{NET | INST | PIN} "object_name" TNM = identifier;
```

- {NET | INST | PIN}：选其中之一，表明约束是加在网络（NET）、元件（INST）或者引脚（PIN）上。
- object_name： NET、INST 或者 PIN 的名称，可以包括 FFs、RAM、Latches、Input Pads、Output Pads 或者三态 Output Pads。
- identifier：分组的名称。

当 TNM 约束加在网络（NET）上，该网络上的有效同步元件将成为分组的一部分，例如将网络 d_input 上的元件指定为一个分组，约束语句如下：

```
NET d_input TNM = d_input_group;
```

2.6.5 设置 TIMESPEC 时序约束的技巧

❖ 技巧内容

TIMESPEC 约束是基本的时序约束，作为时序约束的一个标志符用来调用 TS 属性。

❖ 技巧详解

所有的 TIMESPEC 指定的约束都需要调用 TS 属性，因此都要以 TS 开头，后边紧接一个下划线，然后是标志符。TS 属性用于指定设计中允许的路径延迟以及时钟周期。
它的语法格式为：

```
TIMESPEC "TSidentifier"=PERIOD "timegroup_name" value [units];
```

- Tsidentifier：TS 属性的名称。
- value：一个数值形式的值，为周期的值。
- units：value 的单位，可以是 ms、μs、ps、ns。

比如需要指定系统时钟为 100MHz，占空比为 1:1，可以使用如下的约束语句：

```
TIMESPEC TS_sys_clk = PERIOD sys_clk 10 ns HIGH 5;
```

2.6.6 设置 AREA_GROUP 约束的技巧

❖ 技巧内容

AREA_GROUP 用于附加区域约束，附加区域约束的逻辑只会在指定区域内布局布线。

❖ 技巧详解

使用 AREA_GROUP 约束可以将设计中的部分代码指定到 FPGA 的特定物理区域执行，也就是说，对设计实现进行位置约束。AREA_GROUP 约束可以加在 Logic Block 上，也可以加在时序分组上。
AREA_GROUP 约束的语法为：

```
#定义一个区域分组的语法
INST "X" AREA_GROUP=groupname;
#将约束加在位置分组上的语法
#定义区域分组的位置信息
AREA_GROUP "groupname" RANGE=range;
#定义区域分组的压缩比例，占用位置的比例
AREA_GROUP "groupname" COMPRESSION=percent;
#定义区域分组中的落实是否要重新实现
AREA_GROUP "groupname" IMPLEMENT={FORCE|AUTO};
#是否允许分组外的逻辑和区域约束内的逻辑进行组合
AREA_GROUP "groupname" GROUP={OPEN|CLOSED};
#是否允许分组外的逻辑在区域内布局布线
AREA_GROUP "groupname" PLACE={OPEN|CLOSED};
#是否允许重新配置分组约束
AREA_GROUP "groupname" MODE={RECONFIG};
```

首先需要定义一个分组，然后对该分组使用各种约束。在 AREA_GROUP 约束语法中，groupname 是定义的分组的名称。

对分组使用的各种约束描述如下。

- RANGE 定义了 AREA_GROUP 包含的逻辑能够使用的物理区域，定义的方法如下所示。

```
#使用的 SLICE 的区域，包含 Xm1Yn 和 xm2Yn2 为对角顶点的矩形区域
RANGE=SLICE_Xm1Yn1:SLICE_xm2Yn2
#使用的乘法器的区域，包含 Xm1Yn 和 xm2Yn2 为对角顶点的矩形区域内的乘法器
RANGE=MULT18X18_Xm1Yn1:MULT18X18_Xm2Yn2
#使用的 BLOCK RAM 的区域，包含 Xm1Yn 和 xm2Yn2 为对角顶点的矩形区域内的 BLOCK RAM
RANGE=RAMB16_Xm1Yn1:RAMB16_Xm2Yn2
#举例说明 RANGE 属性的用法
#将 block1 这个模块中的逻辑都指定到 “group1” 这个分组中
INST "block1" AREA_GROUP=group1;
#指定"group1"中的逻辑只能在以 SLICE_X1Y1 和 SLICE_X10Y10 为顶点的矩形区域内实现
AREA_GROUP "group1" RANGE=SLICE_X1Y1:SLICE_X10Y10;
```

- COMPRESSION 定义使用的区域中的压缩比例，即逻辑电路在指定的区域中占用的 SLICE 等资源的百分比，可以取的值为 0～100。COMPRESSION 只能用在 SLICE 上，对 TBUF、BLOCK RAM 和乘法器不能使用 COMPRESSION 属性。
- IMPLEMENT 属性可以指定两个值：FORCE 表示 AREA_GROUP 定义的分组中的逻辑需要重新综合、布局布线；AUTO 表示自动检测 AREA_GROUP 定义的分组中的逻辑是否发生了改变，如果发生了改变才重新执行综合与布局布线。
- GROUP 表示是否封装 AREA_GROUP 分组中的逻辑。如果取值为 CLOSED 表示不允许分组之外的逻辑包含到分组中；取值 OPEN 表示允许分组外逻辑包含到分组中，默认值是 OPEN。
- PLACE 表示是否允许分组外的逻辑在分组划分的区域内布局布线。取值 CLOSED 表示不允许分组外的逻辑在指定的区域布局布线；取值 OPEN 表示允许分组外的逻辑在指定的区域内布局布线，默认值为 OPEN。

2.6.7 设置 DRIVE 约束的技巧

❖ 技巧内容

DRIVE 约束可以指定 SelectIO 输出的驱动能力（mA），可以使用的 IO 标准包括 LVTTL、LVCMOS12、LVCMOS15、LVCMOS18，LVCMOS25 和 LVCMOS33。

❖ 技巧详解

使用 DRIVE 约束的时候，需要区分 IOB 或者 SelectIO。

（1）对 IOB 使用约束的语法为：

```
INST "instance_name" DRIVE={2|4|6|8|12|16|24};
```

其中“instance_name”为 IOB 元件名称，12mA 是默认的驱动能力。

（2）对 SelectIO 使用驱动能力约束的语法为：

```
#对于使用 LVTTL 标准的 IO 端口，默认值为 12mA
INST "instance_name" DRIVE={2|4|6|8|12|16|24};
#对于使用 LVCMOS12、LVCMOS15 和 LVCMOS18 标准的 IO 端口，默认值为 12mA
INST "instance_name" DRIVE={2|4|6|8|12|16};
#对于使用 LVCMOS25 和 LVCMOS33 标准的 IO 端口，默认值为 12mA
INST "instance_name" DRIVE={2|4|6|8|12|16|24};
```

可以看到对于 IOB 和 SelectIO 加的 DRIVE 约束，从语法上都是一样的，不同的只是 IO 端口上的驱动能力。

2.6.8 设置 IBUF_DELAY_VALUE 约束的技巧

❖ 技巧内容

IBUF_DELAY_VALUE 用来约束在输入端口上附加的延迟。当需要专门调整输入延迟以便和电路板上的延迟相匹配的时候，就需要使用该约束。

❖ 技巧详解

输入延迟的约束可以使用在任何输入或者双向端口上，但是这些端口不能用于驱动时钟或者 IOB 寄存器。IBUF_DELAY_VALUE 约束可以设置为 0～16 之间的整数值，设置的值是 IBUF 上的相对的延迟，并不是绝对的延时时间。

使用 IBUF_DELAY_VALUE 的语法为：

```
NET "top_level_port_name" IBUF_DELAY_VALUE = value;
```

其中 value 表示 IBUF 的延时设置，比如想对端口“din1”使用 6 的延迟，则可以使用以下程序：

```
NET "din1" IBUF_DELAY_VALUE = 6;
```

该约束表示在 din1 这个端口上附加了 5 个单位的延迟，具体的延迟时间可以参考使用器件的数据手册。

2.6.9 设置 IFD_DELAY_VALUE 约束的技巧

❖ 技巧内容

IFD_DELAY_VALUE 是使用在 FPGA 输入端口上的延迟。

❖ 技巧详解

该约束使用的端口需要驱动 IOB 寄存器，取值为 0～8 之间的整数，还可以取值“AUTO”。“AUTO”是默认取值，表示为了保证有足够的建立时间而自动加入适当的延迟。如果设置为 0，就表示没有任何附加的延迟。

0～8 之间的取值并不是绝对的时间，而是相对于端口延迟附加的延迟，具体的延迟时间需要参考使用的器件的手册。

IFD_DELAY_VALUE 约束的语法为：

```
NET "top_level_port_name" IFD_DELAY_VALUE = value;
```

比如需要给输入端口 din1 使用 6 的附加延迟，可以使用如下所示的约束：

```
NET "din1" IFD_DELAY_VALUE = 6;
```

2.6.10 设置 IOBDELAY 约束的技巧

❖ 技巧内容

IOBDELAY 约束用来指定输入延迟是如何实现的，支持的器件包括：Spartan-II、Spartan-IIE、Spartan-3、Virtex、Virtex-E、Virtex-II、Virtex-II Pro、Virtex-II Pro X、Virtex-4 和 Virtex-5。

❖ 技巧详解

在 Xilinx 的 FPGA 内部，输入信号可以送到两个目的位置，一个是 IOB 的输入寄存器，一个是 IBUF，通过使用 IOBDELAY 约束可以指定输入信号在 IBUF 和寄存器上的延迟方式。
IOBDELAY 的语法为：

```
INST "instance_name" IOBDELAY={NONE|BOTH|IBUF|IFD};
# instance_name 表示输入端口连接的 IO 元件名称
# NONE 是默认值，表示输入到 IBUF 和输入寄存器上的信号没有延迟
# BOTH 表示输入同时在 IBUF 和输入寄存器上延迟
# IBUF 表示输入只在 IBUF 上延迟
# IFD 表示输入只在输入寄存器 IFD 上延迟
```

例如，不希望在 inst_in 这个元件的 IBUF 和 IDF 上使用延迟，可以用如下的约束语句：

```
INST "inst_in" IOBDELAY=NONE;
```

2.6.11 设置 KEEP 约束的技巧

❖ 技巧内容

在进行 MAP 的时候，有的信号会被优化掉，可以使用 KEEP 约束来保留这些信号。

❖ 技巧详解

如果一个网络被优化掉，那么在物理实现的结果中就不能看到该网络。比如一个模块的输出被连接到该模块的输入，这个网络就可能被优化掉，如果设计者希望能够看到该网络以便检测信号，就可以使用 KEEP 约束。
KEEP 约束的语法为：

```
NET "$1I3245/$SIG_0" KEEP;
```

该约束保证了元件“1I3245”上的信号“SIG_0”不被优化掉。

2.6.12 设置 IOSTANDARD 约束的技巧

❖ 技巧内容

使用 IOSTANDARD 约束来指定 IO 端口的标准，比如 LVTTL 或者 LVCMOS。

❖ 技巧详解

在同一个 FPGA BANK 中的 IO 必须使用同一个 IOSTANDARD 标准，不同的器件支持不同的 IO 标准，设计者可以参考使用器件的数据手册来了解对应的 IO 标准。
对 Spartan-3、Spartan-3A、Spartan-3E、Virtex-II、Virtex-II Pro、Virtex-II Pro X、Virtex-4 和 Virtex-5 器件，可以将 IOSTANDARD 加在 IO 中的 BUFFER 上。例如，在 IBUF 上加 IOSTANDARD 约束。
如果是差分信号，需要将 IOSTANDARD 加在 IBUFDS、IBUFGDS、OBUFDS 和 OBUFTDS 上，不要加在 IBUF 和 OBUF 上。
IOSTANDARD 约束的语法如下：

```
# 将约束加在 IO 元件上
INST "instance_name" IOSTANDARD=iostandard_name;
# 将约束加在 IO 网络上
NET "pad_net_name" IOSTANDARD=iostandard_name;
```

iostandard_name 是 IO 标准的名称，不同的器件支持不同的 IO 标准，可以参考器件手册，常用的 IO 标准包括 LVTTL、LVCMOS 等。

2.6.13 设置 KEEP_HIERARCHY 约束的技巧

❖ 技巧内容

KEEP_HIERARCHY 是综合与布局布线的约束。使用了该约束后，在综合阶段设计的层次结构就不会被优化。在布局布线的过程中，该约束可以保持设计层次结构，这样就可以生成便于仿真的网表文件；保持设计的层次结构还便于使用 Chipscope 观察内部信号。

❖ 技巧详解

综合工具 XST 为了优化设计，有时候会重新组织设计的层次结构，以达到优化的效果。
如图 2.88 所示，设计中希望在顶层有 I1、I2、I3 三个模块，但是综合后，只保留了 I2 模块，其他模块被合并到顶层中。这往往是设计者不希望遇到的，因为在调试过程中，由于打乱了设计的层次结构，就无法定位产生问题的模块。使用 KEEP_HIERARCHY 约束后，就可以保持设计的层次结构。

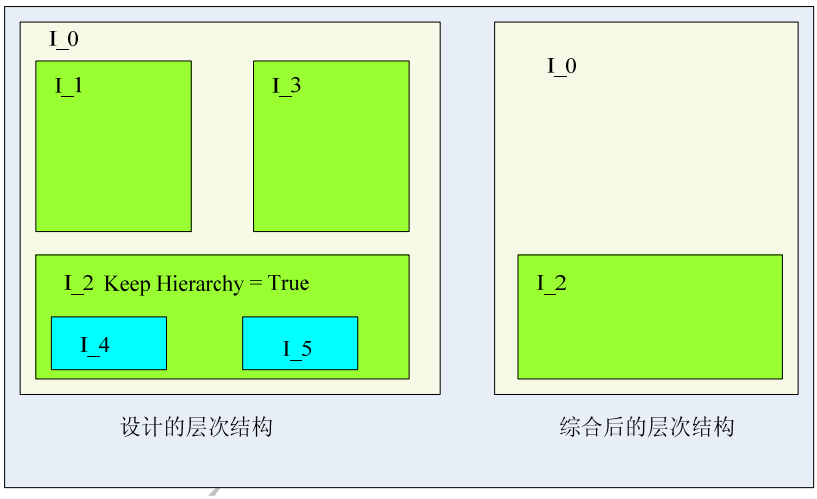

图 2.88 综合工具对设计层次结构的影响

KEEP_HIERARCHY 的语法规则如下：

```
# 在 VHDL 语言中使用
attribute keep_hierarchy : string;
attribute keep_hierarchy of architecture_name: architecture is {TRUE|FALSE|SOFT};
# 对 FPGA 默认属性是 FALSE，对 CPLD 默认属性是 TRUE
# 在 Verilog HDL 语言中使用
(* KEEP_HIERARCHY = "{TRUE|FALSE|SOFT}" *)
# 在 UCF 文件中，对例化后的元件使用约束
INST "instance_name" KEEP_HIERARCHY={TRUE|FALSE|SOFT};
```

- TRUE 表示在综合阶段保持 HDL 代码中描述的层次结构，并且在布局布线（设计实现）阶段也保持该层次结构。
- FALSE 表示层次结构可以被合并到顶层模块中。
- SOFT 表示在综合阶段保持层次结构，但是在布局布线（设计实现）阶段不受此约束的限制。

KEEP_HIERARCHY 使用在需要保持层级结构的模块上。比如在图 2.88 中，如果需要保持 I1、I2 和 I3 模块在顶层中的层次结构不被合并到顶层中，并且不关心 I1、I2 和 I3 的下一级层次，就可以使用如下所示的约束：

```
# 只希望保持 I1、I2、I3 在顶层的层次
INST "I1" KEEP_HIERARCHY=TRUE;
INST "I2" KEEP_HIERARCHY=TRUE;
INST "I3" KEEP_HIERARCHY=TRUE;
```

2.6.14 设置 IOB 约束的技巧

❖ 技巧内容

IOB 是综合与 MAP 操作的约束，可以指定某个触发器或者锁存器被放到 IOB 中，通过利用 IOB 内部的触发器而不是 SLICE 中的触发器，提高逻辑资源的利用率。

❖ 技巧详解

将 IOB 约束加在触发器或者锁存器上，MAP 操作的时候就尽可能将该触发器或者锁存器在 IOB 中实现。在综合阶段，XST 就会将 IOB 约束的信息添加在综合网表中，因此 MAP 操作就可以利用该信息。
IOB 约束的语法规则如下：

```
# 在 VHDL 代码中使用 IOB 约束
attribute iob: string;
attribute iob of component_name: component is "{TRUE|FALSE|AUTO}";
attribute iob of entity_name: entity is "{TRUE|FALSE|AUTO}";
attribute iob of label_name: label is "{TRUE|FALSE|AUTO}";
# 在 Verilog HDL 代码中使用 IOB 约束
(* IOB = "{TRUE|FALSE|AUTO}" *)
# 在 UCF 中指定 IOB 约束
INST "instance_name" IOB={TRUE|FALSE|AUTO};
```

- TRUE 表示允许将触发器和锁存器在 IOB 中实现。
- FALSE 表示不使用 IOB 中的触发器和锁存器。
- AUTO 只在 XST 综合工具中使用，表示根据时序约束自动决定是否使用 IOB 中的触发器。

2.6.15 设置 LOC 约束的技巧

❖ 技巧内容

LOC 是综合与布局的一个最基本的约束，可以指定设计元件在 FPGA 中的布局位置。LOC 可以指定一个具体位置，也可以指定一个位置区域；可以在设计源文件中指定位置约束，也可以在约束文件中指定位置约束。

❖ 技巧详解

指定位置约束的时候，首先需要知道如何定义位置，比较方便的方法就是使用 FPGA Editor、PACE 或者 Floorplanner 来查看位置信息。确定了位置，就可以使用位置约束语句 LOC 进行位置约束。
如果需要给一个模块指定多个位置，用逗号将各个位置分隔开，表明这个模块可以在多个指定的位置中任选一个。也可以将一个位置区域指定给一组设计模块。

定义位置的方法如下：

```
SLICE_X0Y0: 最左下角的 CLB 中的第一个 slice
SLICE_X0Y1: 最左下角的 CLB 中的第二个 slice
SLICE_X1Y0: 最左下角的 CLB 中的第三个 slice
SLICE_X1Y1: 最左下角的 CLB 中的第四个 slice
SLICE_X0Y2: 第一列第二行的 CLB 中的第一个 slice
SLICE_X2Y0: 第二列最底端的 CLB 中的第一个 slice
```

一个 CLB 中包含多个 slice，不同的器件中 CLB 中的 slice 分布可以参考器件手册。

LOC 约束可以加在 FPGA 内部的元件上，包括 SLICE、PIN、CLB、SLICE、Block RAM、DCM 和 PLL 等，不同的元件使用的约束定义也有一定差别，如表 2.6 所示。在指定元件位置的时候，可以使用通配符。

表 2.6　　指定 LOC 约束的元件类型

元 件 类 型	位 置 定 义	位置定义说明
IOB	P12	IOB 位置约束，对承载式封装（chip carrier）适用
	A12	IOB 位置约束，对引脚栅格封装（pin grid）适用
	B, L, T, R	指定位置边缘（底端 B、左端 L、顶端 T、右端 R）
	LB, RB, LT, RT, BR, TR, BL, TL	指定一半边缘位置，（如左底端 LB、右底端 RB）
	Bank0, Bank1…	指定 IOB 所在 BANK 位置
CLB	CLB_R4C3 (or .S0 or .S1)	CLB 的位置定义，适用于 Spartan-II、Spartan-IIE、Virtex、Virtex-E 器件

续表

元 件 类 型	位 置 定 义	位置定义说明
	CLB_R6C8.S0 (or .S1)	函数发生器或者寄存器 slice 位置定义，适用于 Spartan-II、Spartan-IIE、Virtex、Virtex-E 器件
SLICE	SLICE_X22Y3	指定 slice 的位置
TBUF	TBUF_R6C7 (or .0 or .1)	TBUF 的位置定义，适用于 Spartan-II、Spartan-IIE、Virtex、Virtex-E 器件
Block RAM	RAMB4_R3C1	Block RAM 的位置定义，适用于 Spartan-II、Spartan-IIE、Virtex、Virtex-E 器件
	RAMB16_X2Y56	Block RAM 的位置定义，适用于 Spartan-3、Spartan-3°、Spartan-3E、Virtex-II、Virtex-II Pro、Virtex-II Pro X、Virtex-4 和 Virtex-5 器件
Multiplier	MULT18X18_X55Y82	Multiplier 的位置定义
Global Clock	GCLKBUF0 (or 1, 2, or 3)	全局时钟 buffer 的位置定义，适用于 Spartan-II、Spartan-IIE、Virtex、Virtex-E
	GCLKPAD0 (or 1, 2, or 3)	全局时钟 pad 位置定义，适用于 Spartan-II、Spartan-IIE、Virtex、Virtex-E
DLL	DLL0P(or S) (or 1, 2, or 3)	DLL（Delay Locked Loop）元件的位置定义，适用于 Spartan-II、Spartan-IIE、Virtex、Virtex-E
DCM	DCM_X[A]Y[B]	DCM（Digital Clock Manager）的位置定义
PLL	PLL_X[A]Y[B]	PLL（Phase Lock Loop），适用于 Virtex-5

进行位置约束时，可以分为单个位置约束、多个位置约束与区域位置约束。

（1）进行单个位置约束的 UCF 约束语法如下：

```
INST "instance_name" LOC=location;
```

其中 instance_name 为元件名称，location 是该元件的位置。进行单个位置约束的部分例子如下所示，其中的位置都是与元件相关的，比如 SLICE_X3Y2 表示 SLICE 阵列的 X3Y2 位置，RAMB16_X0Y6 表示 Block RAM 阵列的 X0Y6 位置。

```
INST "instance_name" LOC=P12; #在 P12 位置放置 IO 引脚
INST "instance_name" LOC=CLB_R3C5; #指定逻辑电路位置在第三行、第五列的 CLB 中
INST "instance_name" LOC=CLB_R3C5.S0; #使用第三行、第五列的 CLB 中左边的 SLICE
INST "instance_name" LOC=SLICE_X3Y2; #使用 X3Y2 指定的 SLICE
INST "instance_name" LOC=TBUF_R1C2.*; #将 TBUF 放置在第一行、第二列
INST "instance_name" LOC=TBUF_X0Y6; #将 TBUF 放置在 X0Y6
INST "instance_name" LOC=RAMB4_R*C1; #第一列 Block RAM 中任何 Block RAM
INST "instance_name" LOC=RAMB16_X0Y6; #放置 BLOCK RAM 在 X0Y6 位置的 Block RAM
INST "instance_name" LOC=MULT18X18_X0Y6; #指定乘法器在 X0Y6 位置
INST "instance_name" LOC=FIFO16_X0Y15; #指定 FIFO 的位置
INST "instance_name" LOC=IDELAYCTRL_X0Y3; #指定 IDELAY 的位置
```

（2）对多个位置进行约束的 UCF 语法如下：

```
LOC=location1,location2,...,locationx;
```

用逗号将各个位置约束分隔开。如果指定了多个位置的约束，在进行布局布线的时候 ISE 就可以使用多个位置中任何一个位置。以下是一些对多个位置进行约束的例子。

```
# 将触发器放置在 CLB R4C5 的最右边的 SLICE，或者 CLB R4C6 中的任意 SLICE
INST "instance_name" LOC=clb_r4c5.s1, clb_r4c6.*;
# 使用 SLICE 阵列中的 SLICE X2Y10 或者 SLICE X1Y10
INST "instance_name" LOC=SLICE_X2Y10, SLICE_X1Y10;
```

（3）指定一个区域位置约束的 UCF 语法如下：

```
INST "instance_name" LOC=location:location {SOFT};
```

可以通过指定一个矩形区域的两个对角来指定区域约束，两个对角用分号分开，对角位置是矩形的左上角和右下角，逻辑电路会约束在指定的矩形区域内实现。

可以对约束使用“hard”和“soft”参数。“hard”要求逻辑电路只能在约束区域内实现；“soft”表示布局布线工具会根据实际效果调整，如果逻辑电路在其他位置实现能达到更好的效果，就会在其他位置布局布线。默认值为“hard”。以下是区域位置约束的例子。

```
# 指定位置是以 CLB_R1C1 和 CLB_R4C4 为顶点的矩形区域
INST "instance_name" LOC=CLB_R1C1:CLB_R4C4;
# 指定位置是以 SLICE_X3Y5 和 SLICE_X5Y20 为顶点的矩形区域
INST "instance_name" LOC=SLICE_X3Y5:SLICE_X5Y20;
```

（4）设计者也可以在 HDL 代码中指定位置约束，以下是在 VHDL 和 Verilog HDL 代码中指定位置约束的例子。

```
# 在 VHDL 代码中使用位置约束的方法
attribute loc: string;
attribute loc of {signal_name|label_name}: {signal|label} is "location";
attribute loc of bus_name : signal is "location_1 location_2 location_3... ";
# Verilog HDL 中使用位置约束的方法
```

```
    (* LOC = "location" *)
    (* LOC = "location_1 location_2 location_3... " *)
```

2.6.16 设置 OFFSET 约束的技巧

❖ 技巧内容

OFFSET 是基本的时序约束，可以指定外部时钟和内部数据输入/输出引脚之间的时间关系，只能用于和引脚相关的信号，不能用于内部信号。

❖ 技巧详解

如果触发器的数据和时钟输入来自外部的网络，OFFSET 可以用来计算建立时间是否被打破。如果触发器的输出使用外部时钟来触发，别且输出到外部网络，OFFSET 可以用来指定外部输出的延迟。
有 3 种 OFFSET 约束，分别为 Global、Net-specific、Group。
（1）Global 方法设置 OFFSET 约束的语法如下。

```
    OFFSET = {IN} "offset_time" [units] [VALID <datavalid time>] {BEFORE|AFTER} "clk_name" [TIMEGRP
"group_name"] {HIGH | LOW};
    OFFSET = {OUT} "offset_time" [units] {BEFORE|AFTER} "clk_name" [TIMEGRP "group_name"] {HIGH
| LOW};
```

- offset_time：外部时钟沿与数据之间的时间延时。
- units：延时的单位，默认值为 ns（纳秒）。
- clk_name：时钟引脚到输入 BUFFER 之间的网络名。
- TIMEGRP group_name：定义了要约束的寄存器组，默认是对所有与 clk_name 相关的寄存器都要约束，可选项。
- IN | OUT：用来设定 OFFSET 是相对于 IOB 的输入端口或者输出端口。
- BEFORE | AFTER：定义了数据是相对于当前时钟沿还是下一个时钟沿。
- VALID：定义了数据有效的时间周期。
- HIGH | LOW：将时钟的到达时间设置为 0。如果是 HIGH，时钟的上升沿到达时间为 0；如果是 LOW 则时钟的下降沿到达的时间为 0。

（2）Net-specific 用来对特定的数据网线进行约束，语法规则如下：

```
    NET "pad_net_name" OFFSET = {IN} "offset_time" [units] [VALID <datavalid time>] {BEFORE|AFTER}
"clk_name" [TIMEGRP "group_name"] {HIGH | LOW};
    NET "pad_net_name" OFFSET = {OUT} "offset_time" [units] {BEFORE|AFTER} "clk_name" [TIMEGRP
"group_name"] {HIGH | LOW};
```

图 2.89 是输入的 OFFSET 设置，IN 表示是输入，TIN_AFTER 就是数据的保持时间，TIN_BEFORE 就是数据的建立时间。

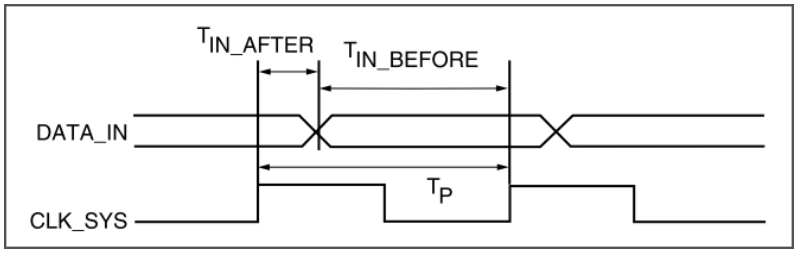

图 2.89　输入的 OFFSET 设置

图2.90是输出的OFFSET设置，输出端口的数据会送到其他的元件，所以这里的TOUT_AFTER就是使用该数据的元件的保持时间，TOUT_BEFORE就是建立时间。

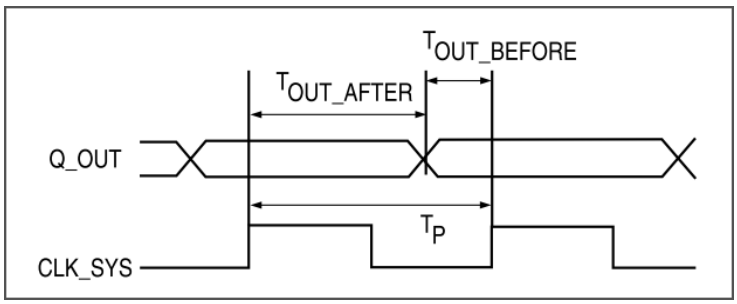

图 2.90　输出的 OFFSET 设置

（3）GROUP 的方法分为对输入/输出分组和对寄存器分组。

如果输入/输出有相同的时序要求，可以用 TIMEGRP 约束将它们指定成一个分组。使用分组的方法可以减少布局布线的时间，也可以使时序报告更简单明了。使用 TIMEGRP 的语法如下：

```
    TIMEGRP "name" OFFSET={IN | OUT} offset_time [units] {BEFORE | AFTER} "clk_net" [TIMEGRP
"reggroup"] {HIGH | LOW};
```

- name：时序分组的名字，分组包括了 IOB 元件。
- offset_time：外部的时间偏移量。
- units：表明时间偏移量的时间单位，默认是 ns（纳秒）。
- IN | OUT：表明 OFFSET 偏移量是相对于 IOB 的输入或者输出。
- BEFORE | AFTER：表示数据偏移量是在时钟之前或者之后。
- clk_net：是时钟输入引脚到 BUFFER 之间的网络的名称，代表了和这个时钟相连的整个时钟网络。

寄存器分组允许用户定义一组寄存器，这一组寄存器都受到相同的时钟约束。如图 2.91 所示，可以为 A、B 和 C 寄存器指定不同的分组，比如 AB 为一个分组，C 为一个分组。虽然它们有共同的数据和时钟输入，但是指定不同分组后，还是可以为不同的分组指定不同的时序约束。

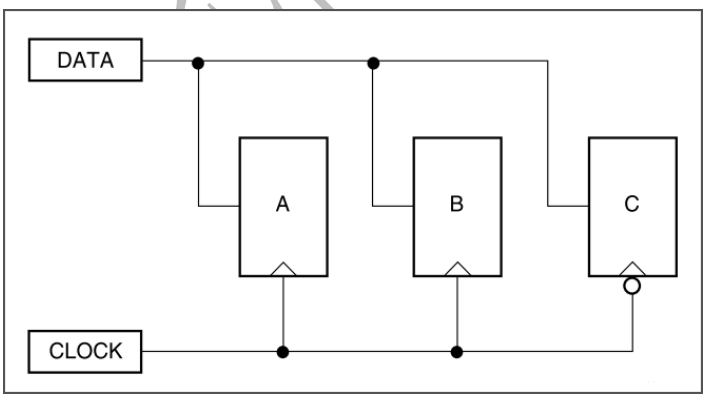

图 2.91　OFFSET 寄存器分组

寄存器分组的约束语法如下所示：

```
    NET "DATA" OFFSET=IN 10 BEFORE "CLOCK" TIMEGRP "AB";
    NET "DATA" OFFSET=IN 20 BEFORE "CLOCK" TIMEGRP "C";
```

2.6.17　设置 PERIOD 约束的技巧

❖ 技巧内容

周期约束 PERIOD 用来约束在同一个分组中的时钟周期，分组可以为同一个时钟域中的同步元件。

❖ 技巧详解

PERIOD 约束加在时钟网络上，分析工具会对时钟网络上所有的网络进行分析，包括时钟的反转信号和由时钟生成的其他时钟信号。

图 2.92 中，PERIOD 约束会加在所有和 CLK 相关的网络上，包括 CLB1 的 Q 输出端到 CLB2 的 D 输入端，EN 输入信号到 EC 使能信号端。

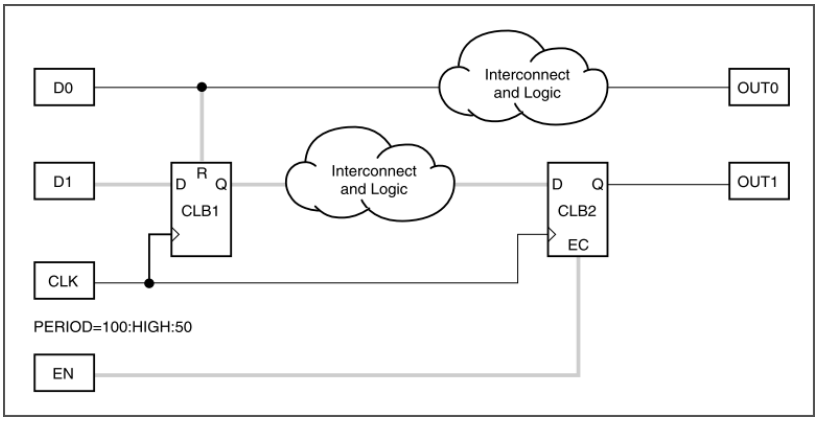

图 2.92　PERIOD 约束的作用范围

（1）首选的定义周期约束的方法是使用比较简单的方法来定义复杂的派生的时钟约束，使用 TIMESPEC 和 TNM 约束定义相关的时钟网络，语法规则如下：

```
    TIMESPEC "TSidentifier"=PERIOD "TNM_reference" period {HIGH | LOW} [high_or _low_time]
INPUT_JITTER value;
```

- TNM_reference：指定时钟约束加在哪个分组上，一般是加在时钟网络上的 TNM_NET 分组名称，也可以是 TNM 或者 TIMEGRP 定义的分组。
- HIGH | LOW：指定时钟初始相位，即第一个时钟是上升沿还是下降沿。
- high_or_low_time：可以指定占空比，即 high 或者 low 的时间，如果没有指定 high 或者 low 的时间，默认的占空比是 1:1。
- INPUT_JITTER：时钟的抖动时间，时钟会在这个值的范围内随机抖动，默认的抖动的单位是 ps（皮秒）。

以下是 UCF 中定义 PERIOD 约束的例子。

```
    TIMESPEC TS_master = PERIOD "master_clk" 50 HIGH 30 INPUT_JITTER 50;
    TIMESPEC TS_clkinA = PERIOD "clkinA" 21 ns LOW 50% INPUT_JITTER 500 ps;
    TIMESPEC TS_clkinB = PERIOD "clkinB" 21 ns HIGH 50% INPUT_JITTER 500 ps;
```

（2）另外一个定义 PERIOD 约束的方法就是直接将约束加在寄存器的时钟引脚上，这种约束的语法如下：

```
    NET "net_name" PERIOD = period {HIGH|LOW} [high_or_low_time] INPUT_JITTER value;
```

- period：要求的时钟周期，默认的单位是 ns（纳秒），可以指定其他的单位，也可以指定频率值，比如 MHz、GHz 或者 kHz。周期单位和周期值之间可以用空格分开，也可以不分开，单位是大小写敏感的。
- HIGH | LOW：指定时钟初始相位，即第一个时钟是上升沿还是下降沿。
- high_or_low_time：可以指定占空比，即 high 或者 low 的时间，如果没有指定 high 或者 low 的时间，默认的占空比是 1∶1。

（3）还可以对派生的时钟周期进行约束，首选的方法是先定义一个时钟周期，这样就允许其他时钟参考这个时钟进行约束，语法如下：

```
    TIMESPEC "TSidentifier"=PERIOD "timegroup_name" "TSidentifier" [* or /] factor PHASE [+ |-]
phase_value [units];
```

- factor：浮点数，如果约束的时钟和参照时钟只是相位不同，可以省略[* or /] factor 参数。

- phase_value：浮点数。

对派生时钟约束的语法如下。

首先定义主时钟的周期。

```
    TIMESPEC "TS01" = PERIOD "clk0" 10.0 ns;
```

定义相移 180° 的时钟，周期相同。

```
    TIMESPEC "TS02" = PERIOD "clk180" TS01 PHASE + 5.0 ns;
```

定义相位提前 90°，但是周期相同。

```
    TIMESPEC "TS03" = PERIOD "clk90" TS01 PHASE - 2.5 ns;
```

定义时钟周期是 TS01 的两倍，相位相当于两倍时钟后，再相移 180°（2.5ns）。

```
    TIMESPEC "TS04" = PERIOD "clk180" TS01 / 2 PHASE + 2.5 nS;
```

（4） 设计者在约束时钟的时候，往往只需要对主时钟进行简单约束，以下给出一些简单的例子。

在 VHDL 中约束时钟的方法如下：

```
    --首先声明约束
    attribute period: string;
    --指定约束
    attribute period of signal_name : signal is "period [units]";
```

- period：要求的时钟周期。
- units：周期的单位，默认值是 ns。

在 Verilog HDL 中约束时钟的语法如下。

```
    (* PERIOD = "period [units]" *)
```

- period：要求的时钟周期。
- units：周期的单位，默认值是 ns。

Xilinx 推荐在 UCF 中指定周期约束，语法如下：

```
    TIMESPEC "TSidentifier"=PERIOD "TNM_reference period" [units] [{HIGH | LOW} [high_or_low_time
[hi_lo_units]]] INPUT_JITTER value [units];
```

2.6.18 设置 PIN 约束的技巧

❖ 技巧内容

PIN 约束用来与 LOC 约束一起，定义引脚的位置。

❖ 技巧详解

PIN 与 LOC 约束一起使用的语法如下：

```
    PIN "module.pin" LOC="location";
```

- "module.pin"：表示 PIN 引脚的名称。
- "location"：表示引脚所在的位置。

2.6.19 设置 TIMEGRP 约束的技巧

❖ 技巧内容

TIMEGRP 是基本的分组约束，可以使用 TIMEGRP 将已有的几个分组进行组合形成新的分组。

❖ 技巧详解

使用 TIMEGRP 定义分组约束有几种方法：将多个分组组合成一个分组，使用排除的方法来定义分组，定义触发器的子分组。

可以使用 3 种方法来创建分组，分别介绍如下。

（1）Combining Multiple Groups into One。

使用这种方法可以将多个分组合并，形成一个新的分组，约束的语法如下：

```
TIMEGRP "big_group"="small_group" "medium_group";
```

这里“small_group”和 “medium_group” 是已经存在的分组，可以是 TIMEGRP 定义的分组，也可以是 TNM 约束定义的分组。

这种方式定义的分组不能嵌套定义，否则在运行 Translate（NGDBuild）的时候会提示出错。比如定义分组“ffs_total”是由分组“ffs1”和分组“ffs2”合并形成的，如果再定义分组“ffs1”是由分组“ffs_total”和分组“ffs2”合并而成，就会出错。

```
# 错误，嵌套定义
TIMEGRP "ffs_total"="ffs1" "ffs2";
TIMEGRP "ffs1"="ffs_total" "ffs3";
```

（2）Creating Groups by Exclusion。

使用这种方法，设计者可以定义一个分组，包含了一个分组中的元件，除了那些同时还属于其他分组的元件。例如，分组“group2”中的部分元件同时也是分组“group3”中的部分元件，可以定义分组“group1”，它包含了“group2”中的元件，但是要排除“group2”中同时属于“group3”的元件，语法如下：

```
# group2 和 group3 可以是 TNM 或者 TIMEGRP 定义的分组
TIMEGRP "group1"="group2" EXCEPT "group3";
```

对于多个分组，也可以使用这种方法。可以定义一个分组“group1”，它包含了分组“group2”和“group3”中的元件，但是要排除属于分组“group4”或者“group5”的元件，语法如下：

```
TIMEGRP "group1"="group2" "group3" EXCEPT "group4" "group5";
```

位于关键字 EXCEPT 之前的分组中的元件是要被包含的，位于关键字 EXCEPT 之后的分组中的元件是需要被排除的。

（3）Defining Flip-Flop Subgroups by Clock Sense。

使用这种方法，可以对寄存器分组。使用关键字 RISING 和 FALLING 可以将寄存器根据上升沿触发或者下降沿出发来分组，语法如下：

```
TIMEGRP "group1"=RISING FFS;
TIMEGRP "group2"=RISING "ffs_group";
TIMEGRP "group3"=FALLING FFS;
TIMEGRP "group4"=FALLING "ffs_group";
```

这里 group1、group2、group3 和 group4 是新定义的分组，ffs_group 必须是只包含寄存器的分组。

2.7 Chipscope 调试技巧

Chipscope 是 Xilinx 开发的针对 FPGA 的在线片内信号分析工具，使用 Chipscope 可以通过 JTAG 实时在线观察 FPGA 的内部信号。

Chipscope 使用集成逻辑分析仪 ILA（Integrated Logic Analyzer）采样 FPGA 的内部信号，将采样值存储在 FPGA 内多余的 Block RAM 中，然后通过 JTAG 观察采样信号。

因为是通过 JTAG 在线观察内部信号，使用 Chipscope 调试 FPGA 非常方便，具有以下几个优点。

- 价格便宜，只要有 JTAG 电缆就可以连接到电路板上观察信号，省去了昂贵的逻辑分析仪。
- 方便调试，可以设置多种触发条件观察信号，并且观察的信号不受电路板面积的限制。
- 可以快速定位问题，因为是在硬件上运行，所以比仿真快很多。
- 使用集成的 ILA/ATC CORE（Integrated Logic Analyzer with Agilent Trace Core），还可以将 FPGA 与 Agilent 的逻辑分析仪连接起来，提供了很大的捕获深度和更加复杂的出发设置，功能非常强大。
- 可以通过远程连接的方式控制 Chipscope 调试，对多点开发非常有利。

2.7.1 使用 Chipscope Inserter 的技巧

❖ 技巧内容

使用 Chipscope Inserter 可以在已经生成的网表中插入 Chipscope CORE，对需要观察的信号采样，以便观察。使用 Chipscope Inserter，用户不用关心具体的实现细节，只关心需要观察的信号。

❖ 技巧详解

设计者首先要将 ChipScope Inserter 插入 ChipScope Core，然后才能用 ChipScope Analyzer 进行观察和调试。使用 Chipscope Inserter 在网表中插入 Chipscope CORE 的方法如下，这里以 ChipScope 9.1 i 为例。

（1）单击“开始”菜单，选择【程序】/【ChipScope Pro 9.1i】/【Xilinx ChipScope Pro Core Inserter】，启动 Inserter 工具，如图 2.93 所示。

左边的“Core Utilization”窗口是 Chipscope Core 占用的 FPGA 资源，现在还没有连接 FPGA 内部信号，所以可以不必关心这个值，因为连接内部信号进行采样还会占用更多的 FPGA 资源。

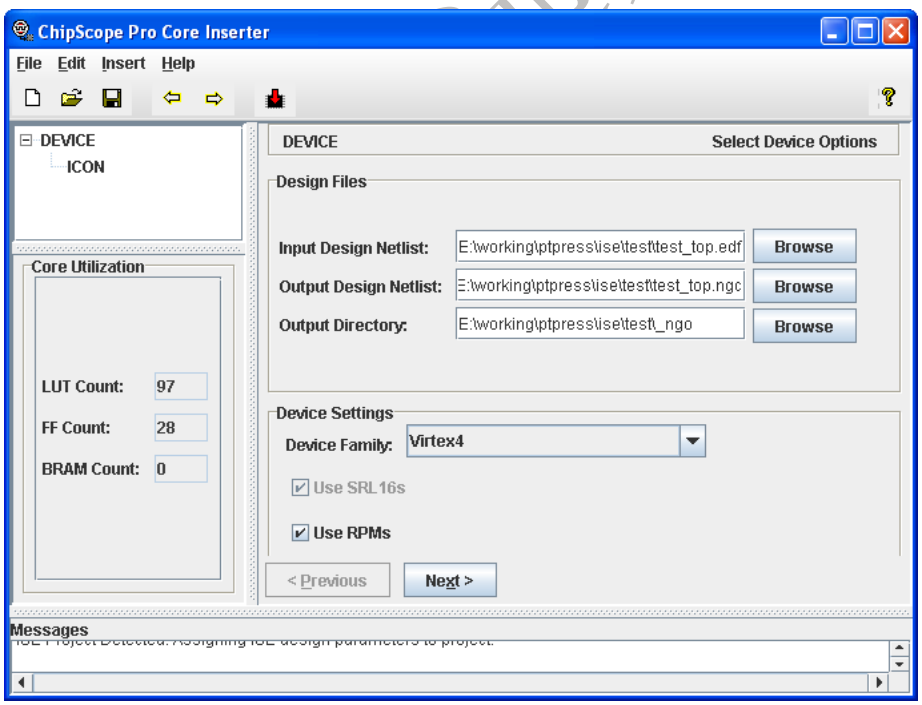

图 2.93 ChipScope Inserter 指定输入网表文件

右边的“Design Files”窗口中，指定输入的网表文件、输出网表文件和输出的目录。输入网表就是需要在其中插入 ChipScope Core 的网表，输出网表就是插入以后的网表，输出目录包含了 ChipScope Core 的网表文件，这些文件在布局布线的时候需要使用。

在“Device Settings”窗口中，指定选用的器件型号，单击【Next】按钮。

（2）接下来出现 ICON 设置窗口，设计者可以指定是否插入 JTAG 时钟 BUFG，设置好以后单击【Next】按钮。

（3）接下来出现设置触发参数的窗口，如图 2.94 所示。

在“Number of Input Trigger Ports”下拉菜单中，可以选择触发端口的数目，每个触发端口的参数在下方的窗口中列出。

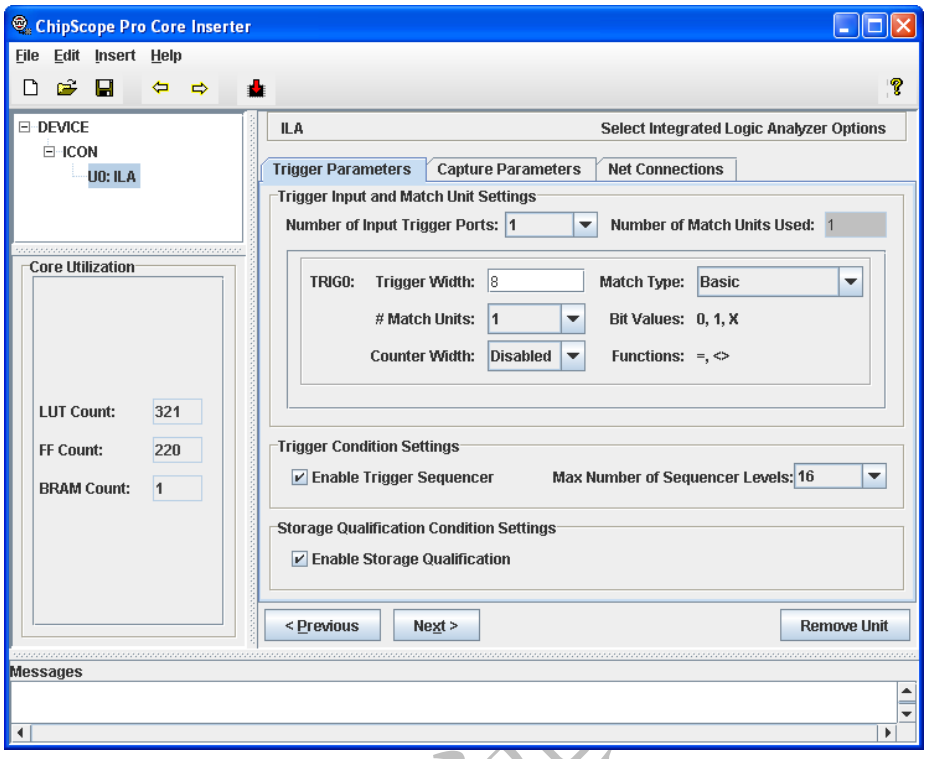

图 2.94　Chipscope Inserter 的触发参数设置

触发端口参数包括触发宽度、匹配类型、匹配单元和计数器宽度。

每个触发端口是由单个的信号组成的总线，触发宽度（Trigger Width）表示触发 ILA 逻辑分析仪进行采样操作的信号宽度，设计者根据调试的需求来设置。触发宽度可以设置的范围是 1～256。

匹配单元（Match Units）是一个比较器，连接在触发端口上，用来检测触发事件。将一个或者多个触发单元的触发事件组合在一起组成触发条件，用来控制捕获需要观察的信号。每个触发端口可以有 1 到 16 个匹配单元。使用多个匹配单元可以增加观察信号的灵活性，但是也会消耗更多的资源。

不同的比较和匹配函数可以用在触发端口上，这就是匹配类型（Match Type），有 6 种匹配类型可以用在 ILA Core 中，如表 2.7 所示。

其中取值 0 表示逻辑 0，1 表示逻辑 1，X 表示不关心，R 表示 0 到 1 的跳变，F 表示 1 到 0 的跳变，B 表示任何跳变。

表 2.7　　ILA 触发端口匹配类型

匹 配 类 型	取　　值	匹 配 函 数	详 细 描 述
Basic	0,1,X	‘=’, ‘<>’	用来比较信号的值，可以是等于（=）或者不等于（<>），但是不能比较跳变，是最节约资源的匹配类型
Basic w/edges	0,1,X,R,F,B	‘=’, ‘<>’	可以检测信号值（0、1），可以是等于（=）或者不等于（<>），还可以检测信号跳变（0 到 1，1 到 0）
Extended	0,1,X	‘=’, ‘<>’, ‘>’, ‘>=’, ‘<’, ‘<=’	可以用来比较地址或者数据，可以进行数值大小的比较，可以等于（=），不等于（<>），大于（>），大于等于（>=），小于（<），小于等于（<=）

Extended w/edges	0,1,X,R,F,B	‘=’, ‘<>’, ‘>’, ‘>=’, ‘<’, ‘<=’	可以用来比较地址或者数据，可以进行数值大小的比较，可以等于（=），不等于（<>），大于（>），大于等于（>=），小于（<），小于等于（<=）。 还可以用于检测信号跳变。
Range	0,1,X	‘=’, ‘<>’, ‘>’, ‘>=’, ‘<’, ‘<=’, ’in range’, ‘not in range’	用来比较信号的值，可以是等于（=）或者不等于（<>）。 还可以用于检测信号的范围
Range w/edges	0,1,X,R,F,B	‘=’, ‘<>’, ‘>’, ‘>=’, ‘<’, ‘<=’, ‘in range’, ‘not in range’	可以用来比较地址或者数据，可以进行数值大小的比较，可以等于（=），不等于（<>），大于（>），大于等于（>=），小于（<），小于等于（<=）。 还可以用于检测信号的范围

计数器宽度“Counter Width”用来统计匹配单元的触发事件，可以选择的范围是 1 到 32bit。

在“Trigger Condition Settings”窗口中可以设置使用触发条件序列，表示在一系列触发条件同时满足的情况下才触发，最多支持 16 个触发条件。如图 2.95 所示，只有满足一个检测条件的时候才会继续检测下一个条件，直到所有的条件都满足了，才会触发逻辑分析仪去采样数据。

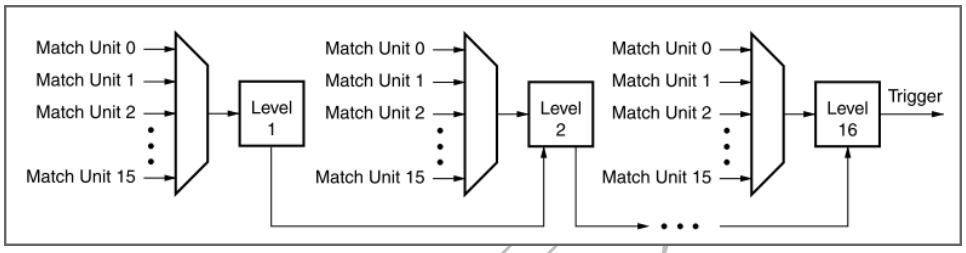

图 2.95　触发序列示意图

设置好以后单击【Next】按钮，进入“Capture Parameters”设置窗口。

（4）Capture Parameters 窗口用来设置捕获信号时候的一些参数，如图 2.96 所示。

如果触发的信号就是需要观察的信号，可以勾选“Data Same As Trigger”。可以指定数据的宽度和采样的深度，以及采样是在时钟上升沿还是下降沿。设置好以后单击【Next】按钮进入“Net Connections”设置窗口。

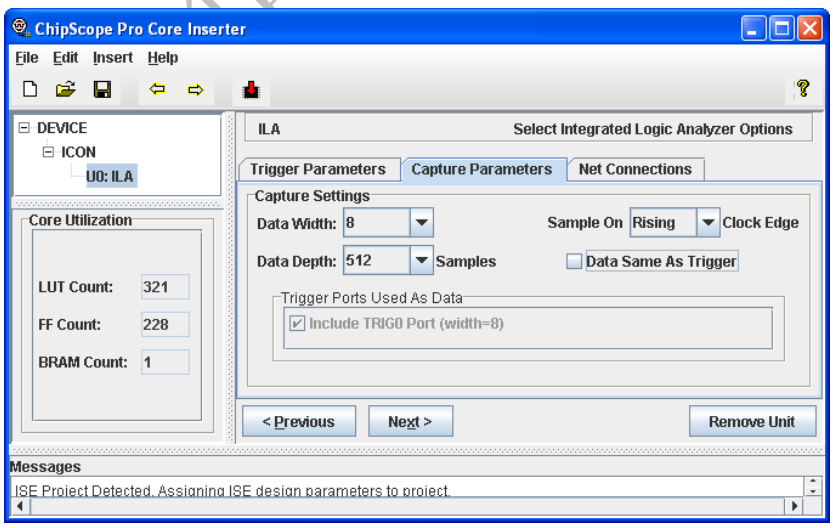

图 2.96　Capture Parameters 设置窗口

（5）Net Connections 设置窗口用来连接采样时钟、触发信号以及需要观察的数据信号，如图 2.97 所示，图中红色的部分表示没有全部连接。

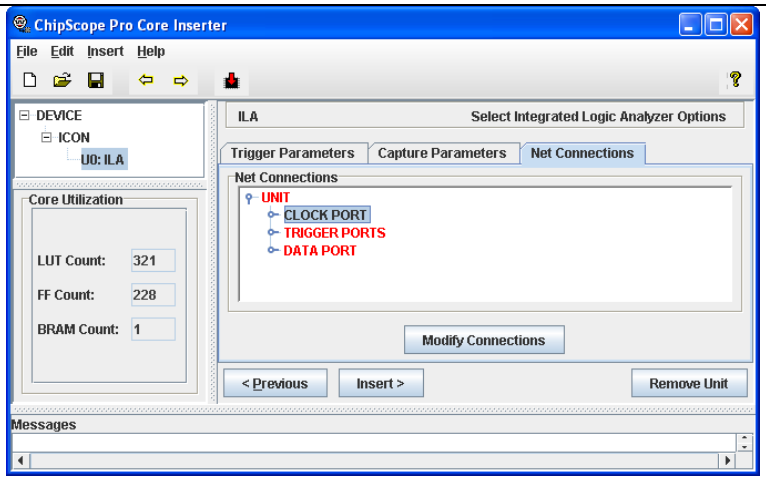

图 2.97　Net Connections 设置窗口

单击【Modify Connections】按钮，弹出“Select Net”对话框，设计者可以在此选择采样时钟，触发信号和需要观察的数据信号。如图 2.98 所示，在右边的“Net Selections”窗口中，3 个选项卡“Clock Signals”、“Trigger Signals”和“Data Signals”分别用来连接采样时钟、触发信号和采样观察信号。

左边的【Filtter】按钮可以用来筛选需要的信号，筛选信号使用的关键字支持通配符。分别点选左边的信号名和右边的网络名，单击【Make Connections】按钮即可将对应信号相连。

连接完以后，单击【OK】按钮回到图 2.97 所示的窗口，此时“Net Connections”窗口中的各个部分变成黑色，表示完成连接。

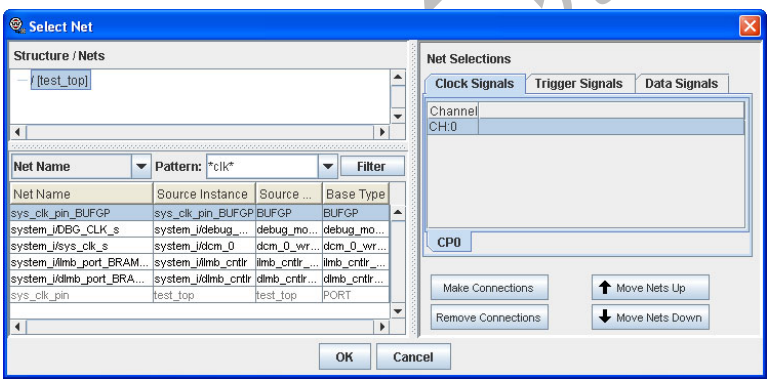

图 2.98　ChipScope Inserter 中连接信号

（6）在图 2.97 所示的窗口中单击【Insert】按钮，在弹出的确认对话框中单击【是（Y）】按钮，在“Messages”对话框中会出现以下信息，表示完成插入 ChipScope CORE。

```
ProjNav Notification Step Complete
Core Insertion Complete.
You must now re-implement the design from the Translate (NGDBUILD) step.
```

（7）此时，需要使用 ChipScope Inserter 的输出网表来重新进行布局布线，最后生成下载文件，下载到 FPGA 后就可以使用 ChipScope Analyzer 观察内部信号。

（8）最后关闭 ChipScope Inserter 的时候保存项目，该项目保存了信号连接的信息，可以导入 ChipScope Analyzer 进行观察。

2.7.2　生成 ChipScope 下载文件的技巧

❖ 技巧内容

使用 ChipScope 的目的是为了调试 FPGA，所以插入 ChipScope CORE 以后一定要生成下载文件，然后用包含 ChipScope CORE 的下载文件配置 FPGA，就可以在线调试了。

❖ 技巧详解

生成包含 ChipScope CORE 的下载文件的方法如下。

（1）如果 ChipScope Inserter 的输入网表就位于 ISE Navigator 项目目录下，生成的输出网表和_ngo 目录也位于 ISE Navigator 项目目录下，插入 ChipScope CORE 之后直接在 ISE 中运行布局布线就可以了，不用重新指定网表文件。

如图 2.99 所示，插入 ChipScope CORE 之后，只需要双击“Implement Design”就可以了。但是不能重新综合，因为会生成新的网表文件覆盖已经插入 ChipScope CORE 的网表。

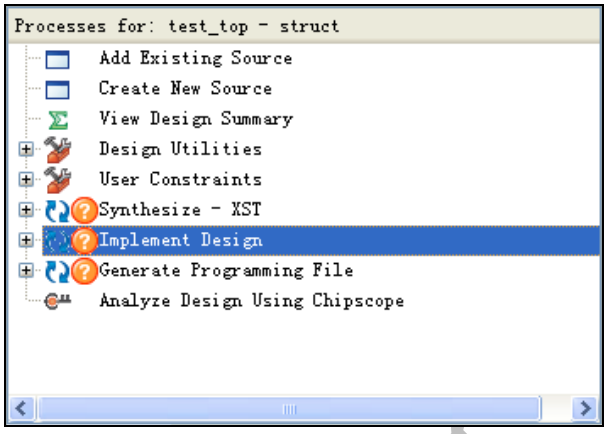

图 2.99　插入 ChipScope CORE 以后重新布局布线

（2）如果 ChipScope Inserter 的输入网表不在 ISE 项目目录下，需要在布局布线的时候指定网表文件，还需要指定_ngo 目录以便调用 ChipScope CORE（ILA、ICON、VIO 等）的网表文件。

这种情况需要先设置 ISE 项目的顶层为 NGO 网表文件，如图 2.100 所示，右键单击“Source”窗口中的器件名称，在弹出的菜单中选择“Properties”。

如图 2.101 所示，设置项目顶层为“NGC/NGO”文件，这样就可以在项目中加入 ChipScope Inserter 的输出 NGO 文件了。

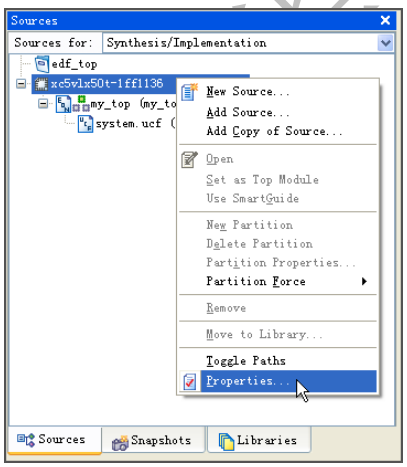

图 2.100　设置项目属性

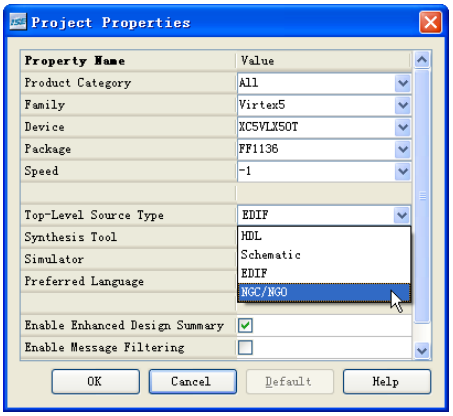

图 2.101　设置项目顶层为 NGO 文件

设置好顶层文件格式后，把以前的顶层文件移除，加入 ChipScope Inserter 输出的 NGO 文件，重新执行“Implement Design”就可以生成包含 ChipScope 的下载文件。

（3）还可以使用脚本的方式，在执行 NGDBuild 的时候指定输入的网表文件为 NGO 文件就可以了，使用脚本的例子如下：

```
    # NGBDuil 中指定输入的网表文件为 chipscope_macro 目录下的 test_top.ngo
    ngdbuild -quiet -p xc5vlx50t-ff1136-1 -insert_keep_hierarchy -dd .\par_output -sd .\chipscope_macro
-uc .\constraints\test.ucf .\chipscope_macro\test_top.ngo . \par_output\test_top_ngdbuild.ngd
```

```
    # MAP
    map -intstyle ise -ol high -cm speed -ignore_keep_hierarchy -c 100 -tx on -w
-o .\par_output\test_top_map.ncd .\par_output\test_top_ngdbuild.ngd .\par_output\test_top_map.pcf
    # PAR
    par -w -intstyle ise -ol high -pl high -rl high .\par_output\test_top_
map.ncd .\par_output\test_top_par.ncd .\par_output\test_top_map.pcf
    # 生成 BIT 文件
    bitgen -w -g unusedpin:pullnone .\par_output\test_top_par.ncd .\par_output\
test_top.bit .\par_output\test_top_map.pcf
```

2.7.3 使用 ChipScope 下载 FPGA 的技巧

❖ 技巧内容

为了调试 FPGA，首先需要配置 FPGA。ChipScope Analyzer 提供了配置 FPGA 的功能，设计者不用退出 ChipScope 可以改变 FPGA 功能，可以提高调试的速度。

❖ 技巧详解

使用 ChipScope Analyzer 下载 FPGA 的方法如下。

（1）单击“开始”菜单，选择【程序】/【ChipScope Pro 9.1i】/【Xilinx ChipScope Pro Core Analyzer】，启动 Analyzer 工具。

（2）在“Project”窗口中单击扫描 JTAG 链按钮，扫描到连接在 JTAG 链上的器件后，右键单击需要配置的器件，选择“Configure”，如图 2.102 所示。

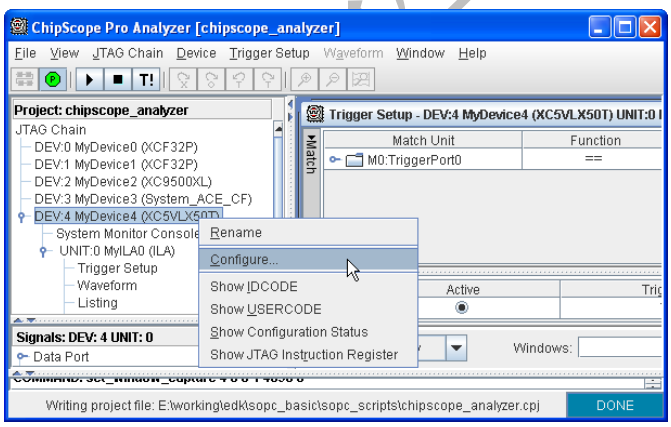

图 2.102 使用 ChipScope Analyzer 配置 FPGA

（3）在弹出的配置窗口中单击【Select New File】按钮，指定需要下载的文件，单击【OK】按钮开始下载，如图 2.103 所示。正确下载后，会提示完成操作。

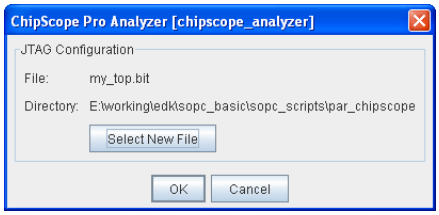

图 2.103 在 ChipScope Analyzer 中指定下载文件

2.7.4 使用 ChipScope Analyzer 的技巧

❖ 技巧内容

ChipScope Pro Analyzer 直接和 ICON、ILA 等 Core 相连，用于观察 FPGA 内部信号，可以设置触发条件，采样内部信号并观察结果。

❖ 技巧详解

使用 ChipScope Analyzer 观察 FPGA 内部信号的方法如下：

（1）单击【开始】菜单，选择【程序】/【ChipScope Pro 9.1i】/【Xilinx ChipScope Pro Core Analyzer】，启动 Analyzer 工具。

（2）在“Project”窗口中单击扫描 JTAG 链按钮，搜索 JTAG 链，扫描到连接在 JTAG 链上的器件后，如果 FPGA 中已经包含 ChipScope Core，出现如图 2.104 所示的窗口。

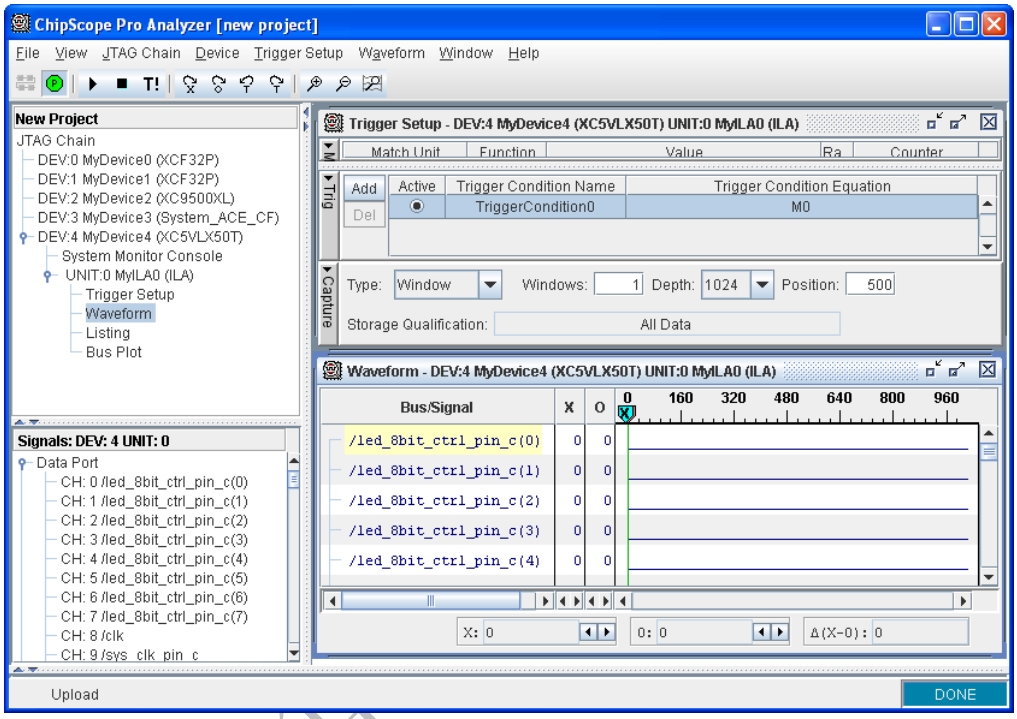

图 2.104　ChipScope Analyzer 窗口

（3）双击 ILA Core 下的 Trigger Setup 出现设置触发条件窗口，其中包括触发条件函数 Match Functions、触发条件 Trgger Conditions 和捕获设置 Capture Settings3 个部分。

Match Functions 设置触发条件判断的条件函数，多个函数的逻辑组合构成完整的触发条件。

- Match Unit 指定条件函数用于触发条件的判断。
- Function 表明判断的准则，比如等于或者不等于。
- Value 设置比较的数值。
- Radix 指定数值的表示进制。
- Counter 规定触发条件定义的事件发生多少次之后才满足条件函数。

Trgger Conditions 窗口中可以把一个或者多个触发条件组合起来构成触发条件。

- 使用【Add】按钮可以加入触发条件，也可以用【Del】按钮删除不需要的触发条件。
- Active 是当前选定的触发条件。
- Trigger Condition Name 是触发条件的名字，可以修改成便于记忆的名字。
- Trigger Condition Equation 是触发条件等式，默认是不同触发条件判断单元的触发条件函数的逻辑与。双击该位置，会弹出“Trigger Condition”对话框，用户可以在对话框中进行设置。

Capture Settings 窗口中可以设置窗口的数目和触发事件的位置。

- Type 定义了窗口的类型，一般选择 Windows。
- Windows 定义了窗口的数目。

- Depth 是捕获窗口的深度。
- Position 是触发位置，默认触发位置是在窗口的最左端，如果需要在窗口的中间显示，可以设置一个数值。

（4）双击 ILA Core 下的 Waveform 出现波形窗口，该窗口显示用于观察的数据，默认的数据名称是 DataPort。为了显示它们对应的名字，可以导入 ChipScope Inserter 项目，方法是单击【File】/【Import】，选择 ChipScope Inserter 项目，最后结果如图 2.104 所示。

设置好触发条件后，单击 按钮开始检测触发条件，如果条件满足，就采样数据进行观察。单击 T! 按钮立即采样当前的数据进行观察。结果会显示在波形窗口中，如图 2.104 所示。

为了便于观察总线信号，用户可以将单独的数据线组合成总线信号，方法是选中所有需要归类总线的信号，单击鼠标右键，选择【Add to Bus】/【New Bus】就可以了。

2.7.5 直接从 ISE 调用 ChipScope 的技巧

❖ 技巧内容

ISE 集成开发环境还提供了在 Project Navigator 中直接调用 ChipScope 的方法，这对于初学者非常方便，因为不需要关心相应的文件放在什么地方。

❖ 技巧详解

直接从 ISE 中调用 ChipScope 的方法如下。

（1）首先启动 ISE Project Navigator，在主窗口中单击【Project】/【New Source】，弹出“New Source Wizard”对话框。

（2）如图 2.105 所示，在“New Source Wizard”对话框中选择“ChipScope Definition and Connection File”，在“File name”区域中输入文件名，其他的保持不变，单击【Next】按钮。

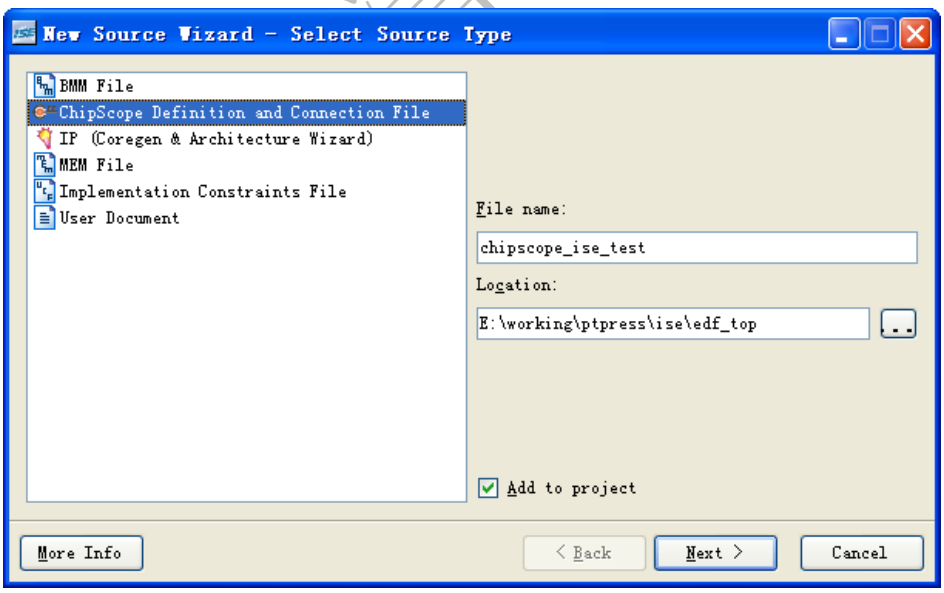

图 2.105 New Source Wizard 对话框

（3）接下来弹出的“Associate Source”中会要求选择 ChipScope 和哪个文件关联。如果选择和顶层文件关联，则整个设计中的信号都可以观察。单击需要观察内部信号的文件，单击【Next】按钮。

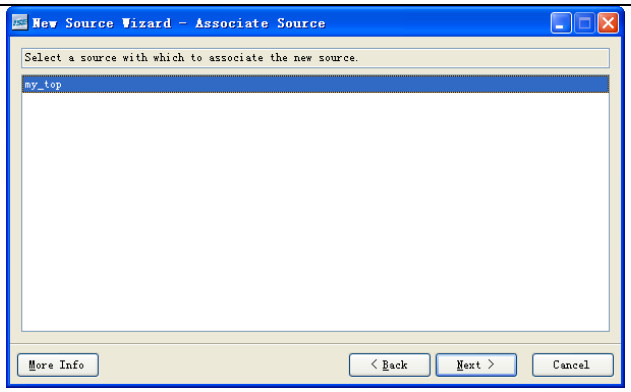

图 2.106　Associate Source 关联文件窗口

（4）在“Summary”窗口中单击【Finish】按钮完成加入 ChipScope 的操作。这时在 ISE 的“Source”窗口中可以看到加入的 ChipScope 项目文件，如图 2.107 所示。该文件已经自动将输入和输出文件与目录设置好，设计者不用关心。

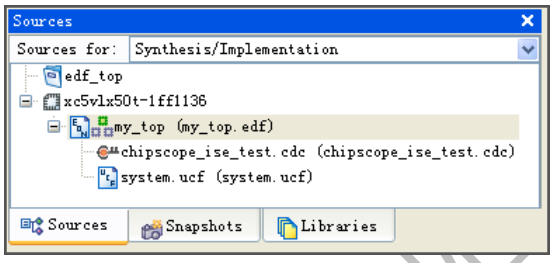

图 2.107　在 ISE 中加入 ChipScope 工程

（5）双击 ISE 窗口中的 ChipScope 项目，会打开 ChipScope Inserter 的窗口，连接信号等操作和 ChipScope Inserter 中一样。

（6）完成以后，在 ISE 中布局布线，生成下载文件即可对 FPGA 进行调试。

2.7.6　使用 ICON Core 的技巧

❖ 技巧内容

ICON（Integrated Controller core）集成控制器，用于控制集成逻辑分析仪与 JTAG 相连，通过 ICON 可以从 JTAG 调试和观察 FPGA 内部信号。

❖ 技巧详解

Core Generator 工具可以新建 ICON Core，完成之后会生成 EDIF 网表文件、网表约束文件和 HDL 示例文件。使用 ICON 的方法和步骤如下。

（1）单击【开始】菜单，选择【程序】/【ChipScope Pro 9.1i】/【Xilinx ChipScope Pro Core Generator】，启动 Generator 工具，如图 2.108 所示，选择 ICON，单击【Next】按钮。

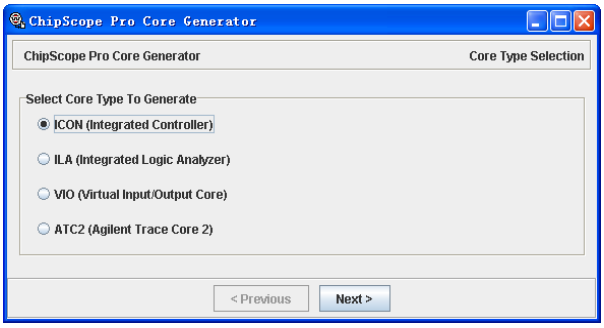

图 2.108　启动 Core Generator 新建 ICON Core

（2）接下来需要指定输出文件的位置和名字，选择器件型号。在“Number of Control Ports”位置表明了 ICON 可以接多少个 ILA 或者 VIO。需要注意的是不要多选，因为每个“Control Port”都要连接。这里选择一个“Control Port”，单击【Next】按钮。

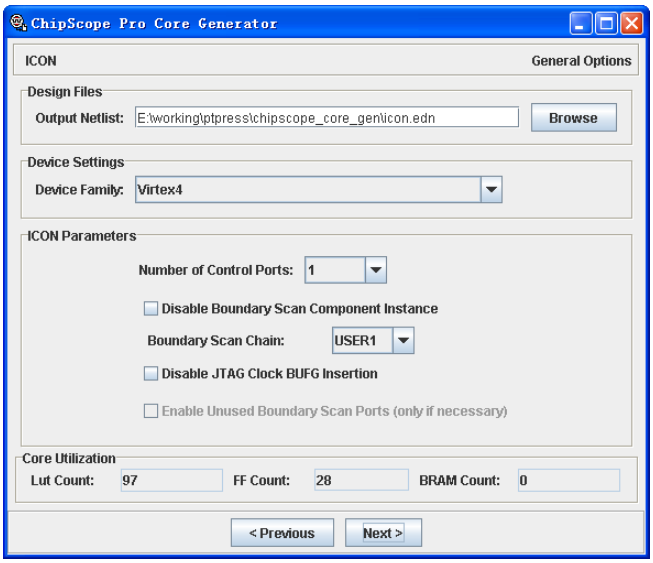

图 2.109　指定 ICON 的输出位置

（3)接下来选择生成的 CORE 的 HDL 语言，可以指定为 VHDL 或者 Verilog HDL，设置好以后单击【Generate Core】按钮。

（4）生成 CORE 之后会弹出“Messages”窗口，关闭窗口。

（5）打开输出文件所在的目录，可以看到以下几个文件：“icon.arg”、“icon.edn”、 “icon.ncf”、“icon_xst_example.vhd”、“icon_xst_VHDL_example.arg”。

- “icon.edn”是集成控制器的的网表文件，布局布线的时候需要使用这个网表文件。
- “icon.ncf”是集成控制器的网表约束文件。
- “icon_xst_example.vhd”是集成控制器的例化文件，设计者可以参考该文件的内容，连接相应的 ILA，其内容和说明如下：

```
library IEEE;
use IEEE.std_logic_1164.all;

entity icon_xst_example is
end icon_xst_example;

architecture structure of icon_xst_example is
  --  ICON core 元件声明，设计者复制该程序到用户代码的 architecture 部分
  component icon
    port
    (
      control0    :   out std_logic_vector(35 DOwnto 0)
    );
  end component;

  --  ICON core 控制信号声明，该信号用来连接 ILA 的控制信号
  signal control0      : std_logic_vector(35 DOwnto 0);

begin
--  ICON core 例化，control0 需要和 ILA 的 control0 相连接
  i_icon : icon
```

```
    port map
    (
      control0    => control0
    );

end structure;
```

2.7.7 使用 ILA Core 的技巧

❖ 技巧内容

ILA（Integrated Logic Analyzer）集成集成逻辑分析仪，用于采样 FPGA 内部信号，以便观察。

❖ 技巧详解

生成 ILA Core 的方法如下：

（1）单击【开始】菜单，选择【程序】/【ChipScope Pro 9.1i】/【Xilinx ChipScope Pro Core Generator】，启动 Generator 工具，如图 2.108 所示，选择 ILA，单击【Next】按钮，出现如图 2.110 所示的窗口。

（2）指定 ILA 的输出文件位置，选定器件，其他的保持不变，单击【Next】按钮。

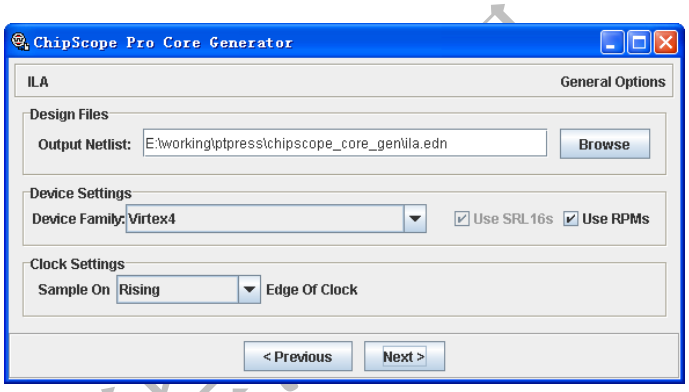

图 2.110　指定 ILA 输出文件位置

（3）接下来需要指定触发参数，和 ChipScope Inserter 中的方法一样，如图 2.111 所示。

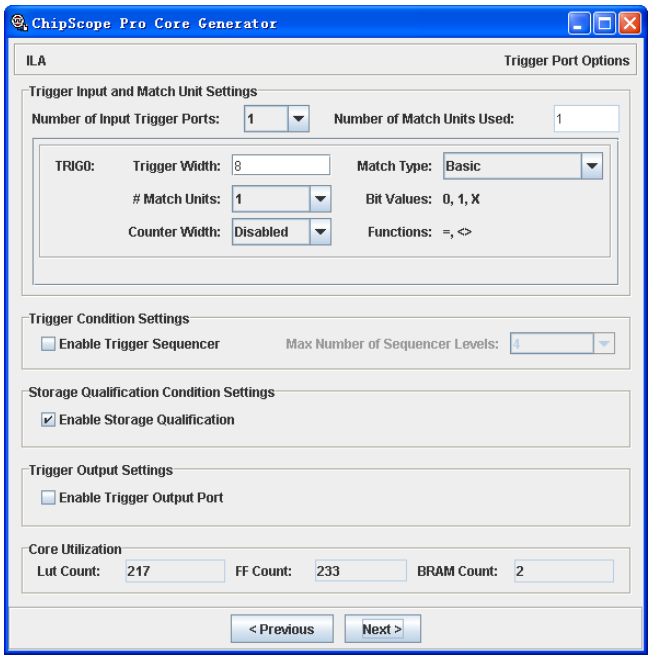

图 2.111　设置 ILA 触发参数

（4）其他设置都和 ChipScoe Inserter 的设置方法一样，最后单击【Generate Core】按钮，生成 ILA Core。

（5）最后会生成“ila.arg”、“ila.edn”、“ila.ncf”、“ila_xst_example.vhd”、“ila_xst_ VHDL_example.arg”。“ila_xst_example.vhd”文件的内容和说明如下：

```
architecture structure of ila_xst_example is
  --  ILA core 元件声明
  component ila
    port
    (
     control    : in    std_logic_vector(35 DOwnto 0);
     clk        : in    std_logic;
     data       : in    std_logic_vector(31 DOwnto 0);
     trig0      : in    std_logic_vector(7 DOwnto 0)
    );
  end component;

  --  ILA core 信号声明
  signal control    : std_logic_vector(35 DOwnto 0);
  signal clk        : std_logic;
  signal data       : std_logic_vector(31 DOwnto 0);
  signal trig0      : std_logic_vector(7 DOwnto 0);

begin
  --  ILA core 例化
  i_ila : ila
    port map
    (
     control   => control,
     clk       => clk,
     data      => data,
     trig0     => trig0
    );

end structure;
```

（6）为了使用 ILA，设计中必需包含 ICON。例如设计中有一个 ILA 用于观察信号，触发端口宽度是 8bit，数据宽度是 32bit，则需要使用如下的代码。

- clk 是采样时钟。
- data 是 ILA 需要观察的信号，和设计中需要观察的信号相连。
- trig0 是触发信号，和作为触发条件的信号相连。
- control0 是 ICON 的控制信号，和 ILA 的 control0 信号相连。

```
library IEEE;
use IEEE.std_logic_1164.all;

entity icon_xst_example is
  port(
    clk        : in    std_logic;
    data       : in    std_logic_vector(31 DOwnto 0);
    trig0      : in    std_logic_vector(7 DOwnto 0)
  );
end icon_xst_example;
```

```
architecture structure of icon_xst_example is
  --  ICON core 元件声明
  component icon
    port
    (
      control0    :   out std_logic_vector(35 DOwnto 0)
    );
  end component;

  --  ILA core 元件声明
  component ila
    port
    (
      control     : in    std_logic_vector(35 DOwnto 0);
      clk         : in    std_logic;
      data        : in    std_logic_vector(31 DOwnto 0);
      trig0       : in    std_logic_vector(7 DOwnto 0)
    );
  end component;

  --  控制信号声明，该信号用来连接 ILA 的控制信号
  signal control0       : std_logic_vector(35 DOwnto 0);

begin
--  ICON core 例化，control0 需要和 ILA 的 control0 相连接
  i_icon : icon
    port map
    (
      control0    => control0
    );

  --  ILA core 例化
  i_ila : ila
    port map
    (
      control   => control0,
      clk       => clk,
      data      => data,
      trig0     => trig0
    );

end structure;
```

联系方式

集团官网：www.hqyj.com　　嵌入式学院：www.embedu.org　　移动互联网学院：www.3g-edu.org

企业学院：www.farsight.com.cn　　物联网学院：www.topsight.cn　　研发中心：dev.hqyj.com

集团总部地址：北京市海淀区西三旗悦秀路北京明园大学校内 华清远见教育集团

北京地址：北京市海淀区西三旗悦秀路北京明园大学校区，电话：010-82600386/5

上海地址：上海市徐汇区漕溪路 250 号银海大厦 11 层 B 区，电话：021-54485127

深圳地址：深圳市龙华新区人民北路美丽 AAA 大厦 15 层，电话：0755-22193762

成都地址：成都市武侯区科华北路 99 号科华大厦 6 层，电话：028-85405115

南京地址：南京市白下区汉中路 185 号鸿运大厦 10 层，电话：025-86551900

武汉地址：武汉市工程大学卓刀泉校区科技孵化器大楼 8 层，电话：027-87804688

西安地址：西安市高新区高新一路 12 号创业大厦 D3 楼 5 层，电话：029-68785218