

附录 A 开发工具简介与相关操作

A.1 Xilinx ISE14.x FPGA 开发平台

本教程使用 Xilinx ISE 作为 FPGA 的软件开发平台。ISE(Integrated Software Environment)功能包括 HDL 描述输入编辑、逻辑图描述输入编辑、HDL 或逻辑电路综合、仿真以及实现等。

FPGA 的开发工具有很多，每个芯片生产厂商都有自己的开发平台，比较流行的有 Xilinx 公司的 Foundation Series ISE 和 Vivado、Altera 公司的 Max+ Plus II 和 Quartus II 以及 Lattice 公司为 ispLSI 器件提供的 ispDesignExpert 等。除了芯片商开发的工具软件外还有第三方的开发平台，比较著名的有 Cadence、Mentor Graphics、Synopsys、Synplify 等。软件开发商提供的开发工具其电路的仿真功能更强大，如 Model Technology 公司的 ModelSim 仿真软件。要进行精确和复杂的仿真最好结合第三方的仿真工具。

A.1.1 Xilinx ISE 简介

本教程采用 ISE 而不选用 Vivado 是因为 ISE 更有利于逻辑电路的学习和理解，而 Vivado 且偏重事件行为描述在工程开发时更为适合。

ISE 集成 FPGA 开发平台，主要功能有设计输入 (Design Entry)、综合 (Synthesis)、仿真 (Simulation)、实现 (Implementation)、生成编程文件 (Generate Programming File) 和下载管理 (Configure Target Device)。

设计输入 (Design Entry): ISE 平台提供了三种设计输入模式。

文本编辑器用于输入 HDL 描述设计或 REPORT 查看；

工程捕获系统 (ECS, The Engineering Capture System) 用于输入逻辑原理图描述设计；

状态机编辑器 (StateCAD) 用于状态机直接描述设计。

其它还有 IP 核生成器 (Core Generator) 用于生成逻辑电路的 IP CORE；约束文件编辑器 (Constraint Editor) 用于编辑约束文件。

综合 (Synthesis): ISE 提供了 Xilinx 的综合工具 XST，用于对 HDL 描述设计的代码进行逻辑电路综合。同时还可以集成第三方综合工具，如 Mentor Graphics 公司的 LeonardoSpectrum 和 Synplify 公司的 Synplify。

仿真 (Simulation): ISE 提供了具有图形化波形编辑功能的仿真工具 HDL Bench 用于对

设计的逻辑功能进行仿真。同时还提供了第三方仿真工具接口，如 Model Technology 公司的 ModelSim 仿真工具。

实现 (Implementation): 将 HDL 综合后代码进行翻译转换 (translate)、映射(map)和布局布线(Place & Route)。

FPGA 编程文件生成(Generate Programming File): ISE 提供了位流生成器 (BitGen) 用于将布局布线后的设计文件转换为位流 (Bitstream) 文件，即“.bit”流代码文件，用于对 FPGA 编程。

下载管理 (Configure Target Device): 提供开发平台调试环境配置程序(iMPACT)，配置目标平台器件的物理参数，调试链路，并通过 JTAG 链将流文件(.bit)代码下载到 FPGA，实现物理上的布线连接，即 FPGA 编程(烧写.bit 数据到 FPGA 芯片上)。同时也提供.bit 文件格式转换，下载到配置存储器 NAND SPI Flash，供开发板复位或加电时自动对 FPGA 编程。

ISE 运行环境推荐采用较高的配置，以节省综合、仿真和实现的时间。一般配置最低要求是 Intel i5 处理器的台式计算机或兼容机，主频 1.5G 以上；内存 2GB 以上；500GB 以上硬盘空间。ISE 运行的操作系统有 Window、UNIX、Linux。建议本教程的实验在 Window 7 上运行，但也可 Windows 10 系统上运行。

A.1.2 ISE 14.x 简介

ISE 开发环境的主界面分为标题栏、菜单栏、工具栏、源文件处理窗口、源文件管理窗口、编辑窗口区、信息提示窗口以及状态栏等，如图 A. 1-1 所示。其中标题栏用于显示目前编辑工程的路径和当前编辑的文件名；编辑窗口区用于原代码的编辑功能，包括 HDL 编辑、波形文件编辑、原理图编辑以及状态机编辑；状态栏用于显示相关命令或者操作的提示信息。

A.1.2 ISE 14.x 简介

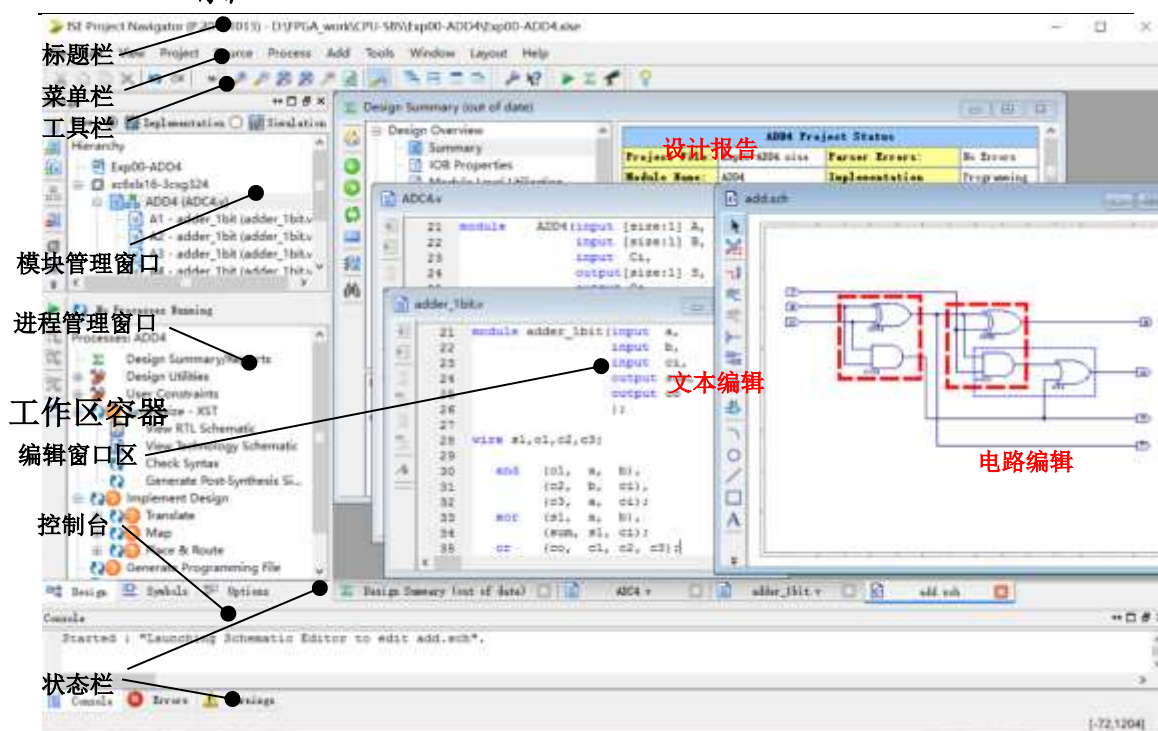


图 A.1-1 ISE 14.x 界面

1. 菜单功能

菜单栏由文件（File）、编辑（Edit）、视图（View）、工程（Project）、源文件（Source）、处理（Process）、添加(Add)、工具（Tools）、窗口（Window）、布局 Layout 和帮助（Help）11 个下拉菜单组成，其中 Add 在电路图编辑时才出现。每个菜单的主要功能如下：

File: 包含新建、打开、保存和关闭工程文件或源文件，如图 A.1-2（a）所示。

Edit: 包括了常用文本编辑功能，如复制(Copy)、剪切(Cut)、粘贴(Paste)、查找(Find)等。另外还包含语言模版（Language Templates）功能和参数设置（Preferences）功能。语言模版中可查询常用的语法、格式等信息。见图 A.1-2（b）所示。

View: 可用于控制显示或隐藏某个界面功能，如工具栏、信息指示区等，如图 A.1-2（c）所示。

Project: 针对工程的管理操作，如新建、添加、删除源文件，备份工程文件等，如图 A.1-2（d）所示。在工程调试时作了许多修改，有时需要清除工程文档重新综合实现，也是在这里操作：Cleanup Project Files...。

Source: 针对工程中的源文件的管理操作,如打开、删除、添加到库中等,如图 A.1-3(c)所示。

Process: 针对当前在源文件管理窗口中选中的源文件进行处理,如运行(Run)、重运行(Rerun)、完全重运行(Rerun All)以及停止(Stop)等,相当于文件处理窗口中双击鼠标左键的某处理,如图 A.1-3(f)所示。

Add: 针对逻辑图描述时自动出现,在电路编辑时增加导线、总线、器件逻辑符号等,对应逻辑图编辑工具栏,如图 A.1-3(g)所示。

Tools: ISE 的一些内嵌软件功能菜单,如约束编辑、核生成器、下载管理等。在逻辑图编辑时自动增加原理图检查、逻辑符号编辑、符号库管理等功能

Window: 对编辑窗口区内的窗体管理,如窗口的排列、切换、查看等。

Help: ISE 的帮助信息,相当 F1 键功能。

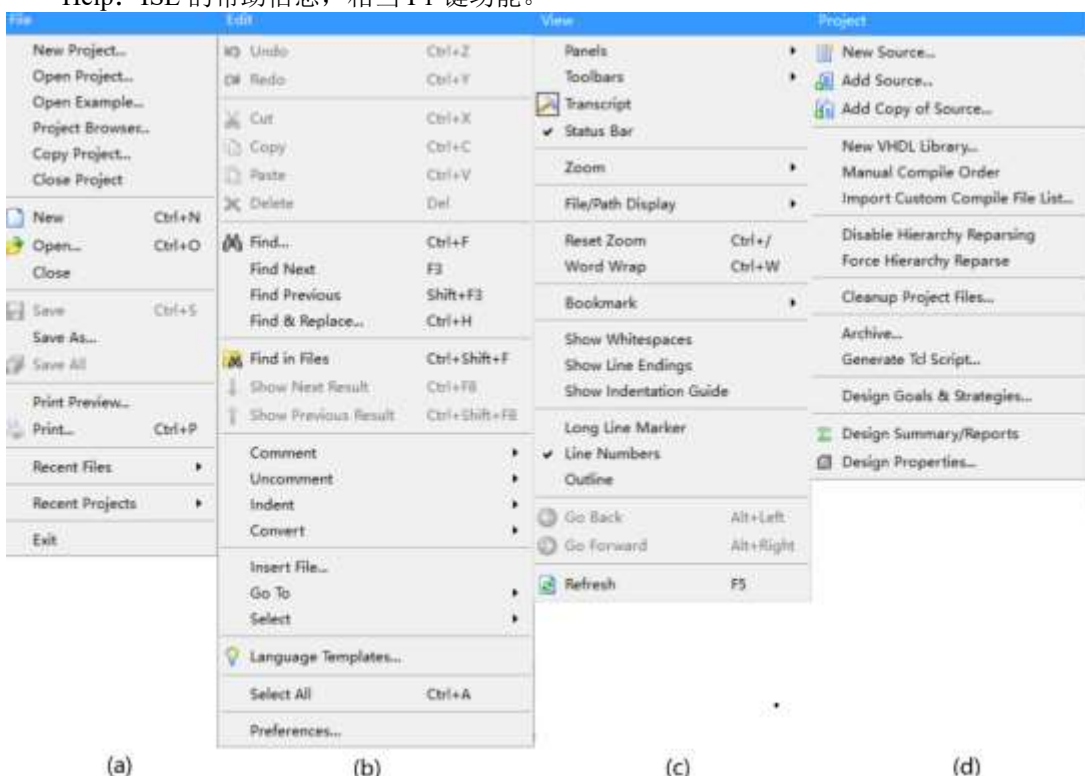


图 A.1-2 ISE 主菜单结构(一)

A.1.2 ISE 14.x 简介

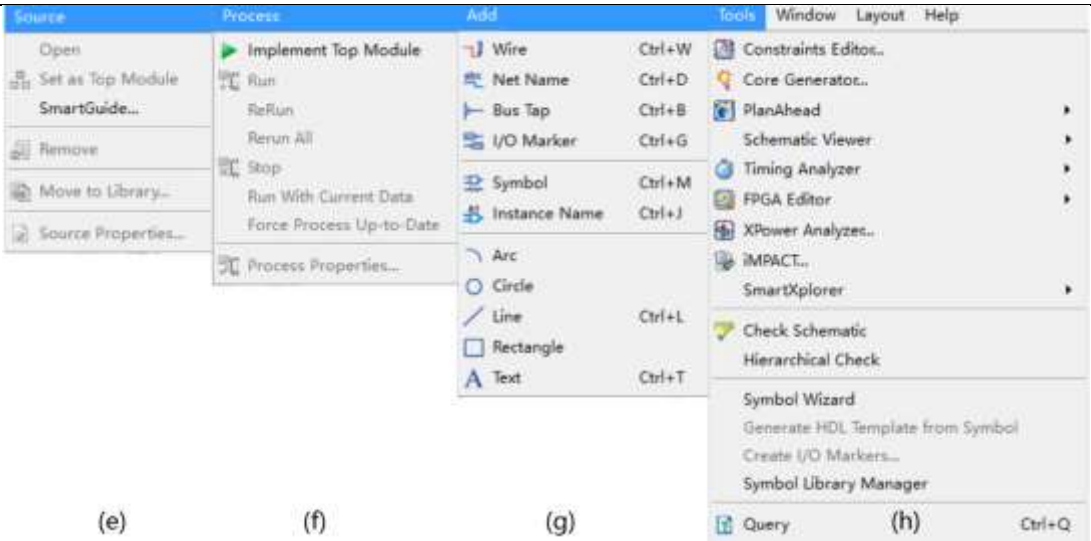


图 A.1-3 ISE 主菜单结构(二)

2. 工具栏

工具栏包括了常用功能的快捷图标。ISE 14.x 提供了多个工具栏。主要有：标准（Standard）工具栏、编辑（Edit）工具栏、视图和逻辑图编辑工具栏。

标准工具栏提供工程文件管理和 Process 功能图标；

编辑工具栏提供了文本编辑的常用图标；工具栏提供了窗口操作及仿真功能管理图标，如图 A.1-4 所示。如果将鼠标停留在某个按钮上一段时间，就会出现按钮功能的提示。表 B-1 是标准工具图标功能的简要说明。










图 A.1-4 ISE 的工具栏

表 B-1 标准工具栏图标

图标	悬停提示	功能简介
	New File	新建一个文本文件，该文件没有限定类型，在保存此文件的时候通过设定其扩展名来设定类型
	Open File	打开一个文件并且在编辑区中显示
	Save File	保存当前编辑的文件
	Save All	保存目前打开的所有文件
	Print	打印在源文件管理窗口中选定的文件
	Print Preview	预览在源文件管理窗口中选定的文件
	New Source	新建源文件，此功能会打开新建源文件的对话框，
	Open Source	打开在源文件管理窗口中选中的源文件
	Edit Source Property	编辑在源文件管理窗口中选定的文件属性。在源文件管理窗口中文件，再单击此按钮就会弹出工程属性编辑对话框
	Implementation Top Module	从顶层模块开始运行设计实现，HDL 编程代码转换为 FPGA 的程序代码
	Run Process	运行在源文件处理窗口所选中的处理
	Rerun All Process Steps	按序运行源文件处理窗口选中操作的所有相关处理
	Stop Process	停止目前正在运行的处理
	Edit Process Properties	编辑在源处理窗口选中某项处理属性。单击此按钮就会弹出属性编辑的对话框；不同的操作有不同的属性对话框
	Toggle Transcript Window	显示或者隐藏信息指示窗口
	Redraw All Windows	刷新所有窗口
	Support and Services	在线技术支持和服务
逻辑图编辑工具图标		
	Select	选择导线、器件等目标
	Zoom Select	放大选中区域

A.1.2 ISE 14.x 简介

	Add wire	放置导线
	Add Net name	放置节点名称
	Rename Selected Bus	重命名选中总线
	Add Bus Tap	放置总线分支符
	Add I/O Marker	放置 I/O 标志
	Add Symbol	放置模块逻辑符号
	Add instance 名称	放置实例(模块)名称

3. 模块管理窗口

此窗口用于模块源(文件)管理，其显示内容取决于对应工程的处理阶段（View 位置）。工程处理分二部分：Synthesis/Implementation(综合/实现)、Simulation(仿真)分别对应图 A. 1-6(a)、(b)。双击此处的文件可以在模块源文件编辑窗口区打开对应文件的编辑窗口。在此窗口可以管理、添加、创建工程所需要的文件。选中此窗口的文件，对应此文件的的处理会出现在进程管理窗口中。

4. 进程管理窗口

此窗口提供对应工程处理阶段，在模块管理窗口选中模块(文件)的所有操作管理。用鼠标

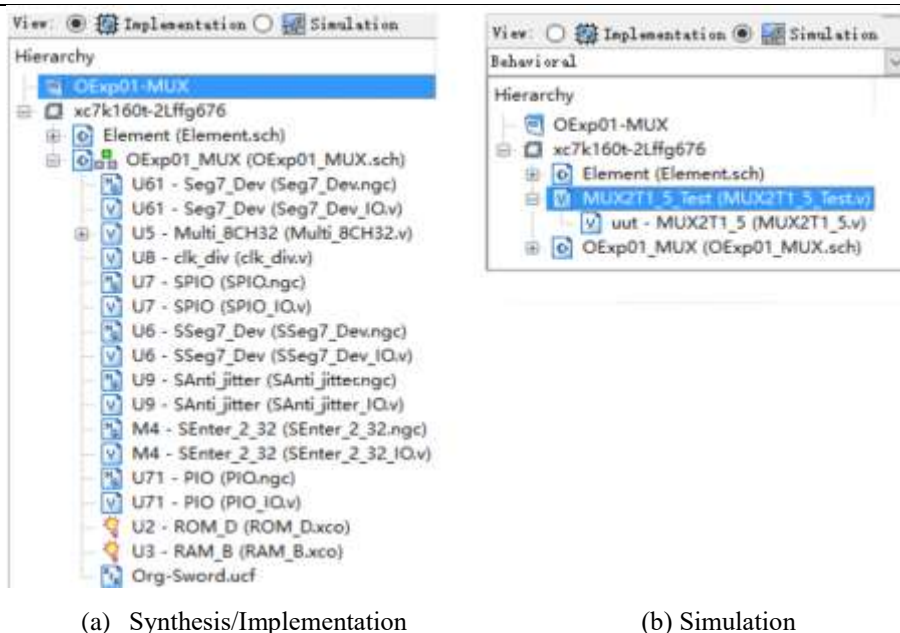


图 A.1-5 ISE 14.x 模块源管理窗口

双击窗口中的条目即可执行相应的处理。在 Synthesis/Implementation 阶段，主要提供用户约束（User Constraints）处理、逻辑电路综合（Synthesize-XST）、设计实现（Implement Design）、编程文件生成（Generate Programming File），如图 A.1-6(a)所示；

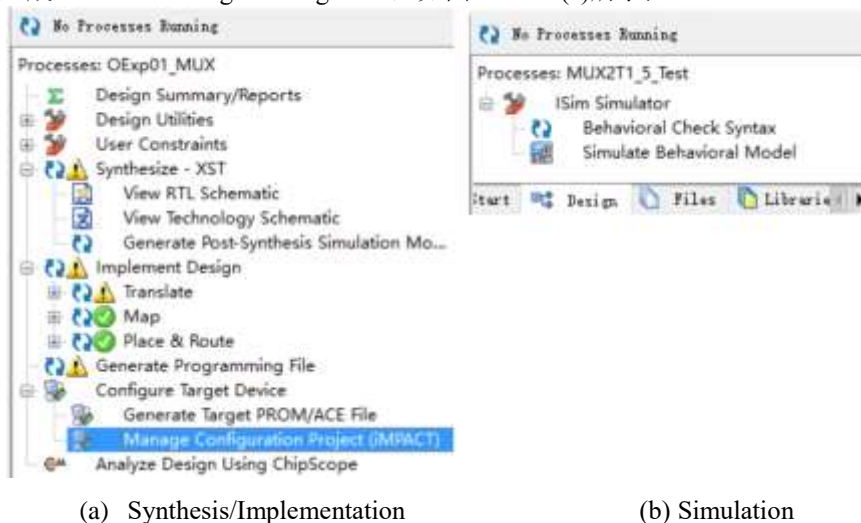


图 A.1-6 ISE 14.x 模块进程管理窗口

A.1.3 使用 ISE 的开发流程

在 Simulation 阶段，主要提供测试代码或激励代码编辑和行为模拟/仿真（Xilinx ISE Simulator）管理，如图 A. 1-6(b)所示。

ISE 14.x 版本，Post-Synthesis Simulation 和 Post-Route Simulation 分别放在 Synthesis 和 Implementation 菜单中，主要提供综合后及布局/布线后仿真，如图 A. 1-6(a)所示。针对在模块管理窗口选中文件的不同，某些窗口的条目也会有所不同。

5. 信息区

信息指示窗口有 4 个选项卡，分别是控制台(Console)、错误 (Errors)、警告(Warnings)、文件内查找 (Find in Files)。其中 Console 选项卡中显示了各个处理操作的综合信息，包括处理过程、错误、警告等；Find in Files 选项卡中显示了在源文件中查找字符串的结果；Warnings 选项卡中显示了处理的警告信息；Errors 选项卡中显示了处理的错误信息。

A.1.3 使用 ISE 的开发流程

完整的 FPGA 设计开发流程主要分为：需求分析、设计方案、设计输入、设计综合、功能仿真、设计实现、布局布线、时序仿真和下载设计等过程。根据调试、仿真、分析发现的问题重复这几个过程，直到设计正确，流程框图如图 A. 1-7 所示。

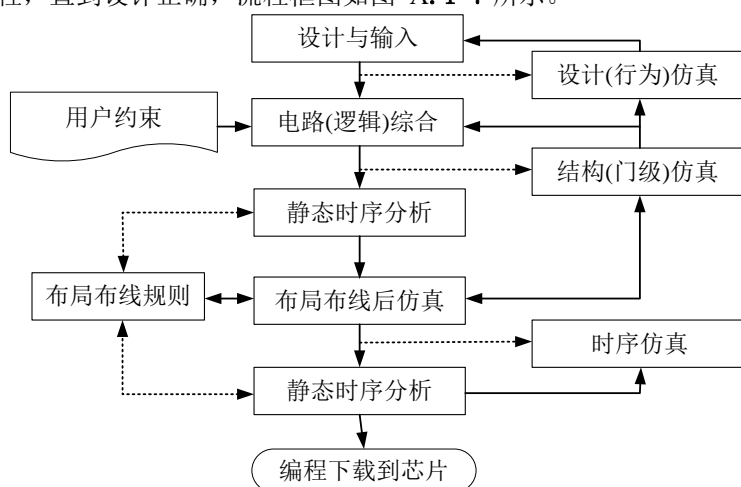


图 A. 1-7 Xilinx 开发设计流程

1. 设计输入与仿真

设计输入 (Design Entry) 是指用 HDL 代码、原理图、波形图以及状态机的形式输入设计源文件；功能仿真 (Simulation) 是指用仿真工具对设计的整体模块或者局部模块进行行为仿真

来检验设计的功能和性能。

2. 用户约束 (User Constraints)

用户约束是指输入对设计的各种约束条件，如时间、布局布线、I/O 映射等。在用户约束的限制下进行综合、实现过程进行控制，以满足速度、面积、引脚位置等需求。这些约束条件存放在文本文件内，可以随时编辑修改。

3. 综合 (Synthesize)

综合是用 HDL 设计流程中的重要环节，是根据 FPGA 的资源，用户约束条件将 HDL 描述转换成 RTL 结构（寄存器传送层描述）。综合结果的优劣直接影响到设计的最终性能，如果 HDL 描述不恰当、约束条件过于苛刻或设计超出 FPGA 资源等，综合均不会成功或给出错误综合。

这个过程与综合工具的智能程度密切相关。ISE 自带的综合工具是 XST，对于简单的设计综合完全没有问题。对应图 A. 1-6(a)中的综合条目（Synthesize-XST），其包含了 3 个子项，分别为查看综合报告、查看综合器件的 RTL 级原理图和检查语法。

4. 实现 (Implementation)

实现过程是根据用户约束和 FPGA 资源作编译、映射和布局布线。对应图 A. 1-6(a)中的设计实现条目（Implement Design）也包含了 3 个子项，分别是编译、映射和布局布线。话别注意的是在进行设计实现处理之前必须进行约束条件的编辑（引脚约束是必须定义的），否则设计实现可能会出错或实现得到的结果没有意义。


5. 硬件编程下载 (Programming)

可编程逻辑器件的编程是指芯片内部的开关阵列、查表器、连线资源等进行配置的过程，它是硬件资源连接的一种编程，因此硬件编程就是要生成配置文件。这一步要生成编程位流文件.bit，并且将其下载到 FPGA 芯片内部，对应图 A. 1-6(a)的生成编程文件条目（Generate Programming File）项。

详细的 ISE 设计开发过程请参考 Xilinx ISE4. x 使用指南。

A.2 语言描述开发操作与实现

A.2.1 建立 ISE 工程

首先，启动 Xilinx ISE 系统软件，双击桌面的图标进入 ISE 集成开发系统，缺省情况下系统自动打开上次退出时使用的工程，，下面来建立第一个 ISE 工程：Gate_Describing。

【步骤1】 点击左上方 file 菜单，选择 new project 新建一个 project，弹出如图 A.2-1 所示
~ 485 ~

A.2.1 建立 ISE 工程

新工程建立向导 New Project Wizard。

命名和参数设置：

Name: Gate_Describing, 不能作用中文字符；

Location: “工程位置”。可建立或选择工程所在目录，缺省是上次填写的位置，建议同一个项目所有工程在一个目录下；

Working Directory: “工作目录”，改变 Location 时会自动改变；

Description: 描写工程内容，可省略；

Top-level source type: 此工程顶层使用 Verilog HDL 描述，下拉选择 HDL。

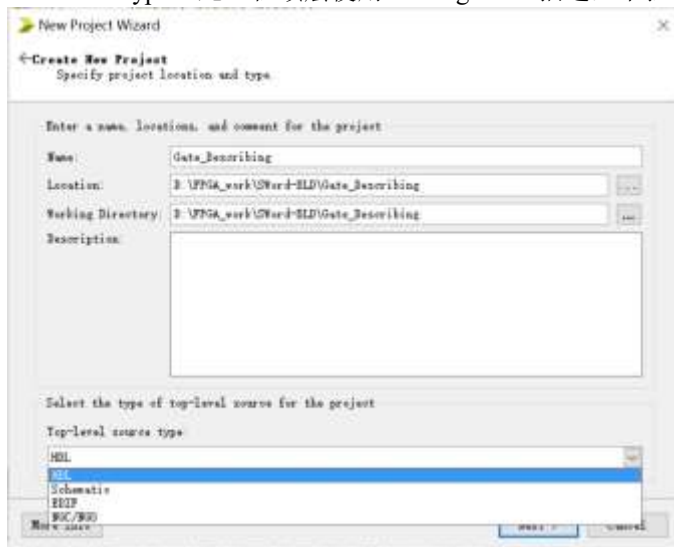


图 A.2-1 HDL 描述工程命名和位置参数设置

完成设置后点击“Next”进入到设备属性窗口，如所示图 A.2-2 所示。

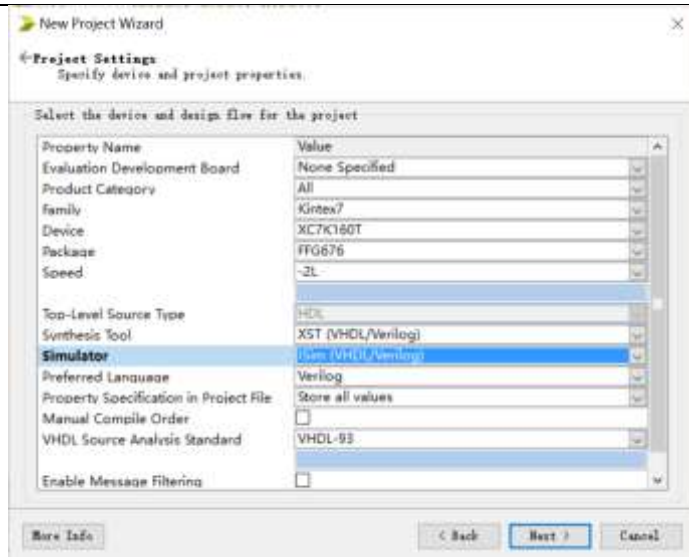


图 A.2-2 设置后的工程设备属性

在设备属性页中主要设置以下选项，其它选项默认：

Family: 根据实验板的型号来选择，(选择 Kintex7/SPARTAN-3)。

Device: 根据实验板上的 FPGA 芯片型号来选择，(选择 XC7K160T/XC3S200)。

Package: 根据实验板上的 FPGA 芯片型号来选择，(选择 FFG676/FT256)

Speed: 根据实验板上的 FPGA 芯片型号来选择，(选择-2L/-4)

Simulator: 选择 ISim(缺省)，ISE 自己的仿真模拟软件。也可选择第三方，但要另行安装。

完成详细的属性设置后，点击 NEXT，弹出工程属性核对窗口如图 A.2-3 所示。

A.2.2 HDL 创建编辑模板及设计输入

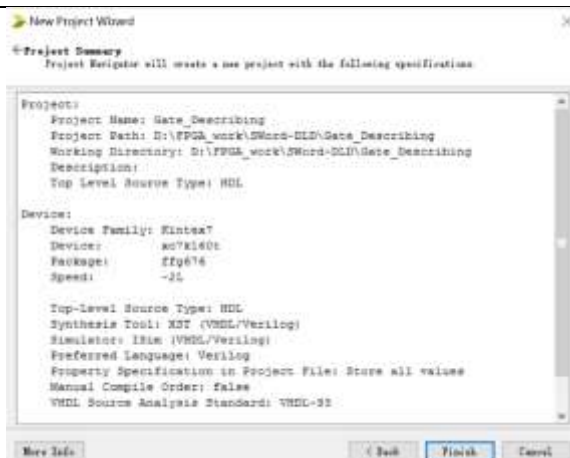


图 A.2-3 Gate_Describing 工程属性核对窗口

核对无误后点击 Next，系统弹出工程模板窗口，如图 A.2-4 所示。工程模板菜单等相关内容在第一章已经详细介绍过了，这已经不再赘述，请参考第一章相关内容。

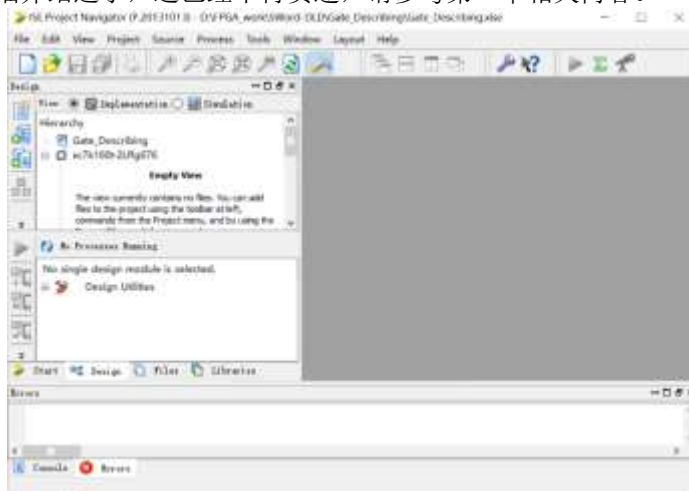


图 A.2-4 工程模板窗口

A.2.2 HDL 创建编辑模板及设计输入

【步骤2】创建 HDL 描述编辑调试模板，并输入电路设计描述，模块命名：xor.v。

具体操作如下：

1) 点击左上方 Project 菜单选择 **New Source** 或在 **Source** 窗口下空白处用鼠标右键点击 **New Source** 如图 A.2-5 所示。

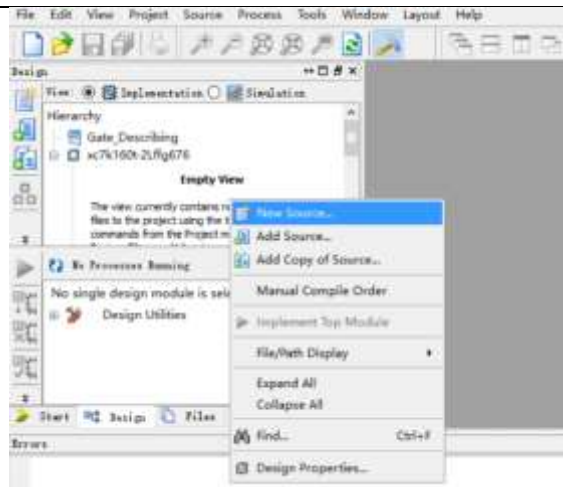


图 A.2-5 New Source 窗口

2) 点击后, 出现如图 A.2-5 所示新 Source 模板向导窗口, 选择 Verilog Module 作为源类型、输入文件名 **xor**、确认 **Add to project** 被选中。

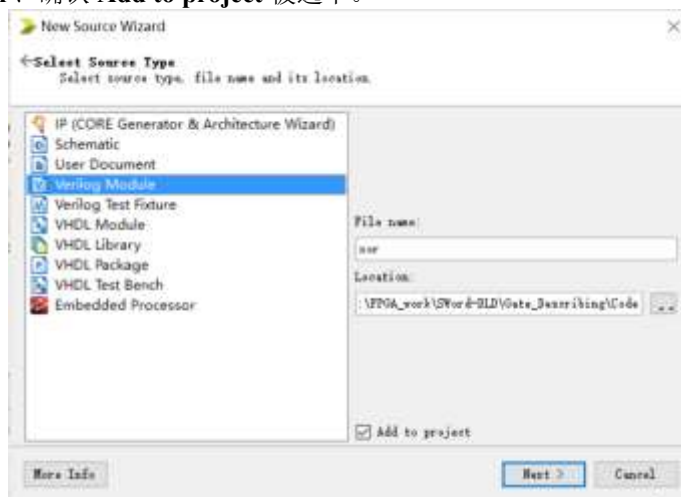


图 A.2-6 选择 Verilog 作为输入源类型

注意: 建议所有设计文件存放在一个独立目录, 此例在工程根目录建立了“..\Code”, 以便重建工程时方便分离, 由于硬件设计调试固有的调试特征, EDA 软件经常会发生古怪的现象, 需要重建工程。

连续二次点击 **Next**, 接着点 **Finish** 弹出如图 A.2-7 所示编辑模块窗口。此时可以看见一个错误, 这是正常的, 因为建立模块时跳过了模块端口信号的输入。可以在模板建立后输入模
~ 489 ~

A.2.2 HDL 创建编辑模板及设计输入

块端口信号，这样更方便。

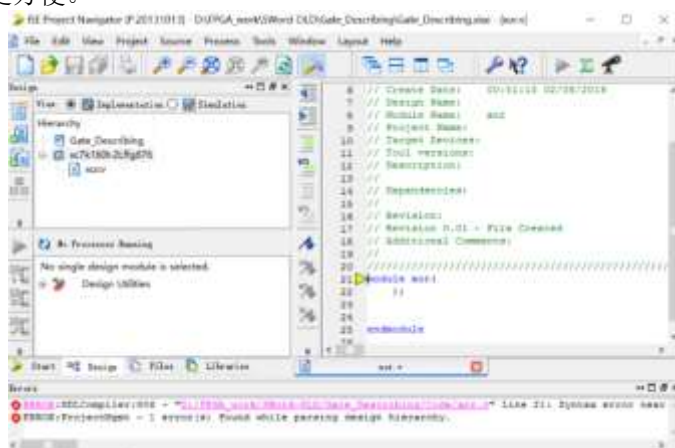


图 A.2-7 HDL 编辑模板窗口

【步骤3】 在 HDL 模板窗口设计并编辑输入 Verilog HDL 内置门描述的“异或门”代码，模块代码输入编辑完后保存 xor.v 文件。

在设计(Design)模块管理窗口 View 选项处选择 Implementation，然后选中 xor.v 文件，在 Processes Running 进程运行管理窗口点击“+Synthesize-XST”的“+”号，展开菜单，并双击“Check Syntax”，检查输入代码的语法。若存在错误调试并排除语法错误，语法调试结束后的结果如图 A.2-8 所示。

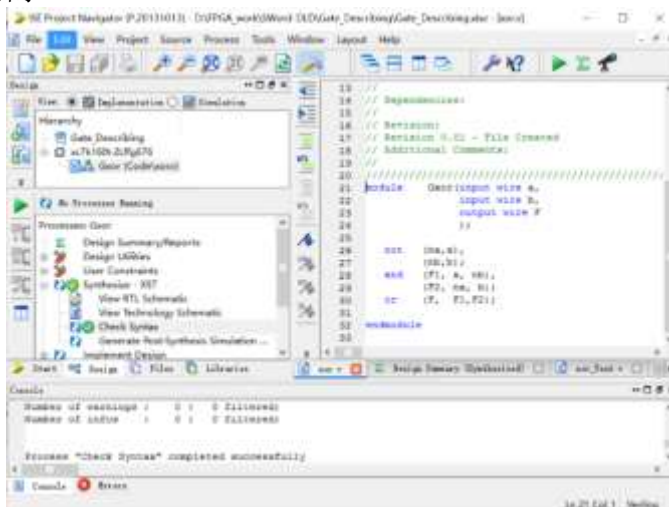


图 A.2-8 语法调试后的 HDL 描述

A.2.3 HDL 设计综合

【步骤4】 综合“异或门”电路并查看 ISE 综合后生成的逻辑电路。

在模块管理窗口 View 选择 Implementation 并选中 xor.v 文件，在 Processes Running 进程运行管理窗口双击“Synthesize-XST”，ISE 自动进行设计综合。

综合结束后双击“View RTL Schematic”后弹出如图 A.2-9 所示窗口，点击 OK 查看分析 xor.v 代码编译综合后的 RTL 层电路结构。

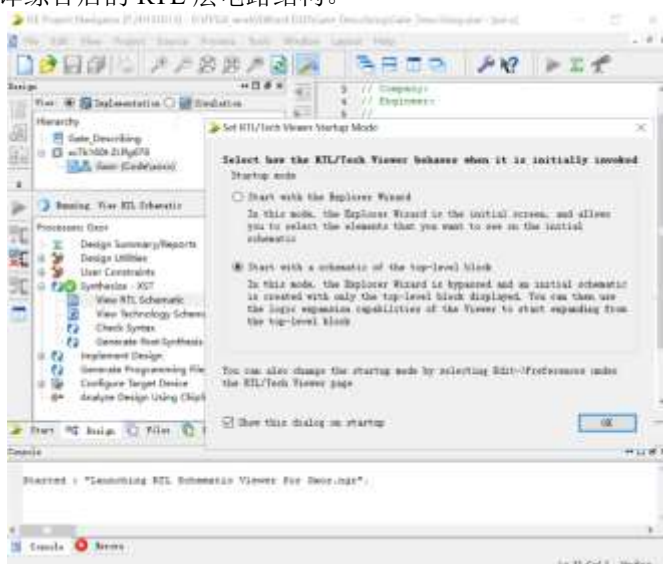


图 A.2-9 双击 View RTL Schematic 弹出窗口

在 RTL 窗口双击 GXor 模块，展开 GXor 模块内部电路，查看 ISE 综合后的 RTL 电路如图 A.2-10 所示，分析综合后的电路结构与设计的目标是否一致。PlanAhead 工具软件有更强大的 RTL 分析功能。

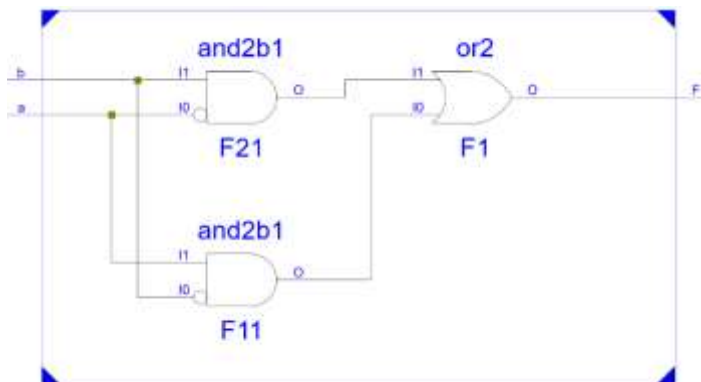


图 A.2-10 xor.v 综合后 ISE 的生成电路

A.2.4 时序仿真测试

【步骤5】设计异或门 xor.v 模块的仿真激励代码 xor_Test.v。

1) 设计测试方案

仿真验证的基本思想是在被测模块输入端加入激励信号，在输出测量并将测试结果绘制成真值表与设计时比较，如果二个真值表完全重叠即为设计达到预先目标。

2) 编写激励代码

Xor 模块只有二个输入端，总共只有四种输入组合，故可以采用遍历方式写测试激励代码，参考代码如下：

```
a = 1'b0;           //ab = 00
b = 1'b0;
#100;               //延时 100 单位
a = 1'b0;           //ab = 01
b = 1'b1;
#100;
a = 1'b1;           //ab = 10
b = 1'b0;
#100;
a = 1'b1;           //ab = 11
b = 1'b1;
#100;
a = 1'b0;
b = 1'b0;
```

用上述激励值作为输入调用 xor 模块，并加入测试输出的系统任务 \$display 或 \$monitor 描述仿真结果显示就是完整的测试代码模块：

```
a = 1'b0;           //ab = 00
b = 1'b0;
$display("a,b = %b %b",a,b,":::F = %b",F); //即时显示后失效
#100;               //延时 100 单位
.....
$monitor($time,,,"a = %b b = %b F = %b",a,b,F); //持续实时显示只要变量变化
```

\$display 或 \$monitor 的区别在于前者是即时显示一次，后者是持续实时显示到仿真结束，但都只能在控制台上显示输出结果。

3) 建立测试代码模板

每次用系统任务或函数编写仿真显示输出不仅麻烦而且没有波形图那样直观，实际仿

真时只要写激励代码即可，仿真显示结果可以利用 ISE 自动生成的模板(fixture)，非常方便。

生成测试模板与新建 HDL 编辑模板相同，只是选项不一样。首先在设计(Design)表单 View 选项选中 Simulation，再点击左上方 Project 菜单选择 **New Source** 或在 **Source** 窗口下空白处用鼠标右键点击 **New Source** 弹出如图 A.2-11 所示新 Source 模板向导窗口。

选择 Verilog Test Fixture 作为源类型，点击 Next，如果有多个 HDL 模块会弹出关联窗口选择关联那个模块(选择被测试模块)，其余与 HDL 编辑模板生成过程完全一样，完成后的测试代码编辑模板如图 A.2-11 所示。

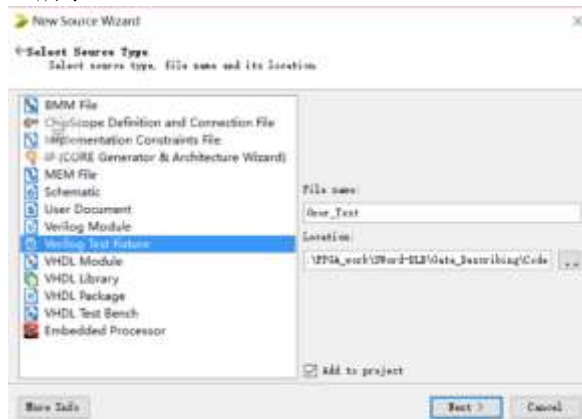


图 A.2-11 仿真测试代码编辑模板

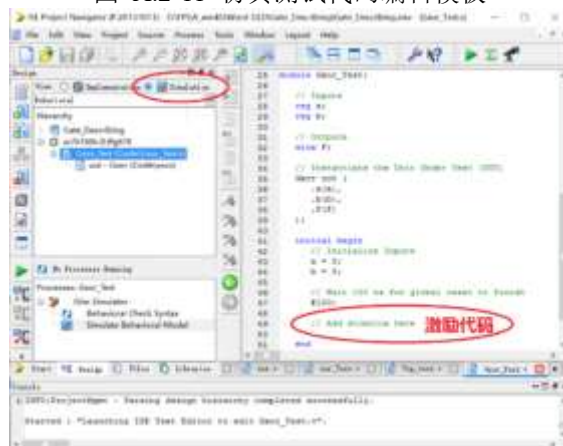


图 A.2-12 仿真测试代码编辑模板

激励代码描述只要写在“// Add stimulus here”后面那可。一般激励代码实际上就是模块输

A.2.4 时序仿真测试

入信号的初始化值，可以用 `initial begin.....end` 初始化过程块或 `always@beginend` 过程块来描述，只不过是实际电路只初始化一次，而仿真激励是在不同时间多次初始化而已。

在“// Add stimulus here”后面插入设计好的激励代码完整测试程序设计，完整的参考代码如下，其中斜体加粗部分为激励代码，其余是 ISE 模板生成：

```
module Gxor_Test;           // 测试代码没有端口信号列表
    // Inputs
    reg a;
    reg b;

    // Outputs
    wire F;

    // Instantiate the Unit Under Test (UUT)
    Gxor uut (
        .a(a),
        .b(b),
        .F(F)
    );
    initial begin
        // Initialize Inputs
        a = 0;
        b = 0;
        // Wait 100 ns for global reset to finish
        #100;
        // Add stimulus here
//          $display("a,b = %b%b",a,b,":::F = %b",F);           // $display 与 $monitor 可选择
//          #50;
//          a = 1'b0;
//          b = 1'b1;
//          $display("a,b = %b%b",a,b,":::F = %b",F);
//          #50;
//          a = 1'b1;
//          b = 1'b0;
//          $display("a,b = %b%b",a,b,":::F = %b",F);
//          #50;
//          a = 1'b1;
//          b = 1'b1;
//          $display("a,b = %b%b",a,b,":::F = %b",F);
    end
    //          initial      $monitor($time,,"a = %b b = %b F = %b",a,b,F);
endmodule
```

其中系统任务命令 `$display` 和 `$monitor` 只要选择一种即可。

4) 仿真测试代码运行

ISE 系统自带 ISim 仿真软件,也可以使用第三方的仿真软件,如 Modelsim 仿真软件,但 ISim 是免费的,一般的数字电路和计算机系统设计仿真应该足够了。目前 ISim 仿真软件功能也非常强大的,可任意添加中间变量到波形图中,数据导出等。在工程建立时已经选择了 ISim,参见图 A.2-2 工程属性选项。

首先选择设计(Design)表单, View 选项选中 Simulation, 下拉菜单选择 Behavioral 做行为仿真,在层次化模块管理窗口点中测试代码模块(Gxor_Test.v)。

然后在进程运行指示窗口双击“Simulate Behavioral Model”,如图 A.2-13 所示,ISE 将自动激活 ISim 仿真软件并在弹出窗口运行 Gxor_Test.v 仿真测试代码,如图 A.2-14 所示。

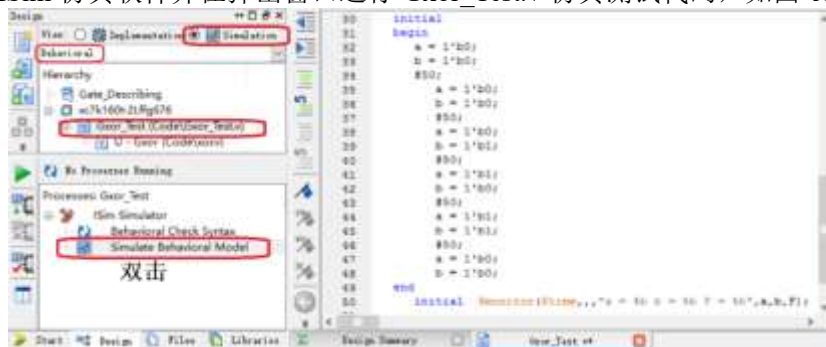


图 A.2-13 ISim 仿真软件关联与激活

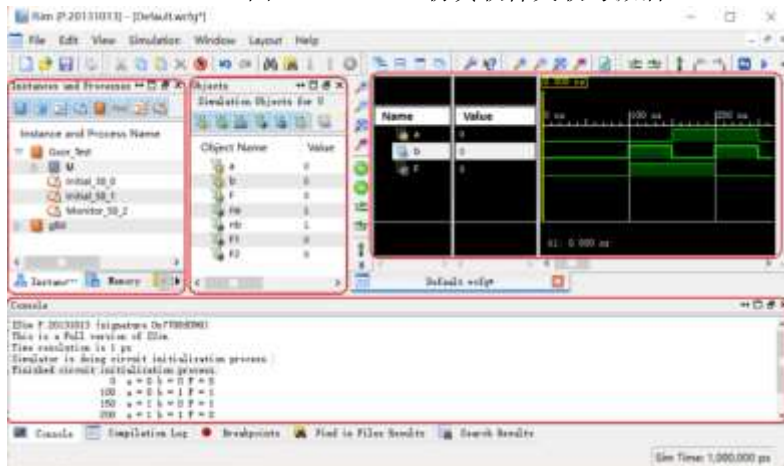


图 A.2-14 ISim 集成管理窗口

ISim 仿真软件运行界面也是一个简单的集成环境,主要有实例和进程(Instance and

A.2.4 时序仿真测试

Processes)、模拟对象(Simulation Objects)、仿真波形输出窗口和控制台(Console)。现在在控制台窗口已经可以看见\$monitor 运行结果, ISim 仿真结果以时序波形图形式也已经在输出窗口显示, 如图 A.2-15 所示。以后的仿真例子只给如图 A.2-15 的最后结果。

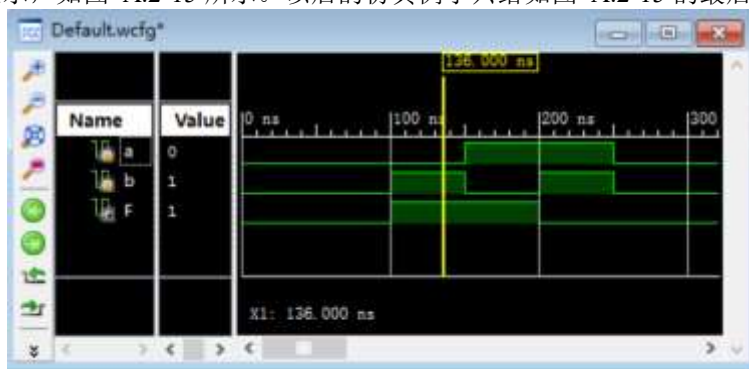


图 A.2-15 Gxor.v 仿真结果

分析比较图 A.2-15 的时序波形, 可以看出当 a、b 输入不同时, 输出 F 为 1 符合异或门逻辑定义, 验证了 Gxor.v 的设计在逻辑功能上是正确的。

5) 添加中间变量

在实际电路仿真调试时往往需要查看中间逻辑变量来排查电路错误或故障, ISim 可以很方便地增加中间变量进入仿真时序。下面通过异或门包含的 HDL 与、或、非内置基本门的仿真来学习中间变量的仿真, 比直接写仿真激励要方便得多。

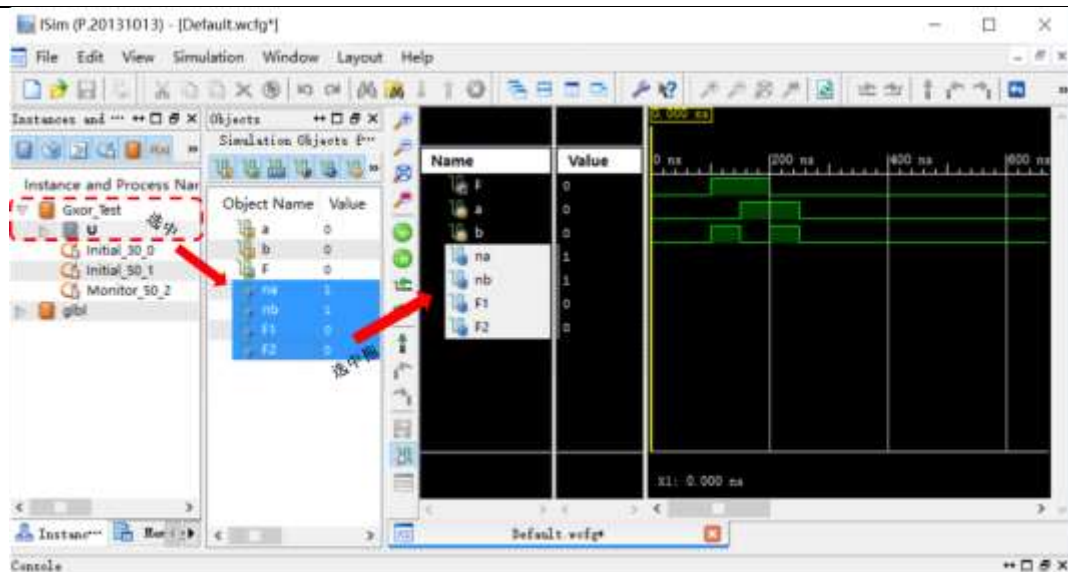


图 A.2-16 添加仿真中间变量

在实例和进程(Instance and Processes)窗口点击 Gxor_Test 左边的 ▼ 展开测试实例并选中被测模块 “U”，在模拟对象(Simulation Objects)窗口选中需要加入的仿真变量 na、nb、F1 和 F2 拖入仿真波形输出窗口，如图 A.2-16 所示。

整理仿真波形输出窗口变量顺次后，先选择菜单栏 Simulation→ 点击 Restart 项重新开始仿真，然后再点击 Run 仿真，新增加变量的仿真结果就显示在输出窗口，如图 A.2-17 所示。

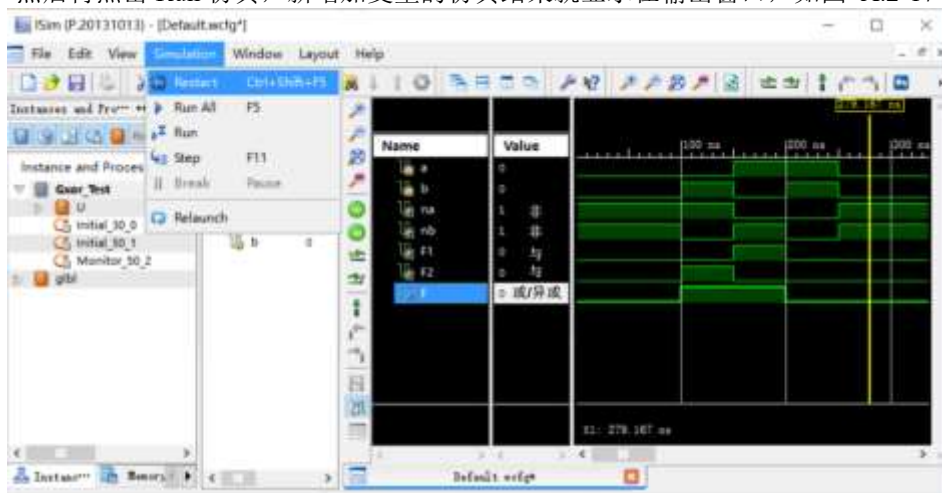


图 A.2-17 添加新变量后重新运行后的仿真结果

A.2.5 建立用户约束

在仿真波形输出窗口用鼠标右键点击变量将弹出浮动菜单，可以改变变量的显示属性，如图 A.2-18 所示的命名(Name)、显示进制基数(Radix)、变量组合(Group)、虚拟总线(Virtual Bus)等。

A.2.5 建立用户约束

HDL 设计描述电路通过综合和仿真后就可以对其做针对 FPGA 的编程，这一步称这 FPGA 设计实现。要正直进行“实现”还需要知道设计电路与 FPGA 外部引脚的关系，也就是电路通过 FPGA 那此引脚输入输出，引出脚有什么限制，FPGA 内部布线有什么要求，如时序、区域或功耗等，这个工作称为用户约束(User Constraints)。

FPGA 设计中有三类约束文件：

- 用户设计文件 (.UCF 文件)。用户在设计阶段编写文件，UCF 是个文本码文件可以用文本编辑器和 Xilinx 约束文件编辑器进行编辑；
- 网表约束文件(NCF 文件)。ISE 综合后自动生成网表约束文件，NCF 约束文件的语法和 UCF 文件相同，别在于 UCF 文件由用户输入，NCF 文件由综合工具自动生成，发生冲突时，以 UCF 文件为准，UCF 的优先级最高；
- 物理约束文件 (.PCF 文件)。ISE 运行实现后生成的物理约束文件，由映射产生的物理约束和用户输入的约束二部分构成，同样 UCF 用户约束优先级最高。

ISE 提供了 PlanAhead 软件工具来进行可视化的设计分析，帮助用户方便实现 FPGA 实现约束。目前的 PlanAhead 功能已经非常强大了，不再仅仅是以前的约束软件，而是加入了 RTL Design Synthesize、Netlist Design(Implement)，等 ISE 的 Project Navigator 功能，完全可以独立实现 FPGA 的设计工作。在用户约束比较简单时可以用 ISE 直接添加用户约束文件(.ucf)，也可以用 PlanAhead 工具软件来实现。

ISE 直接建立用户约束

首先在 ISE 设计(Design)窗口的 View 选项选中 Implementation，从 Simulation 窗口切换回 Implementation 窗口，并选中 xor.v 文件。与建立 HDL 编辑模板一样，点击左上方 Project 菜



图 A.2-18 仿真弹出菜单

单选择 **New Source** 或在 **Source** 窗口下空白处用鼠标右键点击 **New Source** 如图 A.2-5 所示。

在新 Source 模板向导窗口，选择 **Implementation Constraints File** 作为源类型、输入文件名 **Gxor** 并确认 **Add to project** 被选中，如图 A.2-19 所示，点击 **Next** 弹出核对窗口。

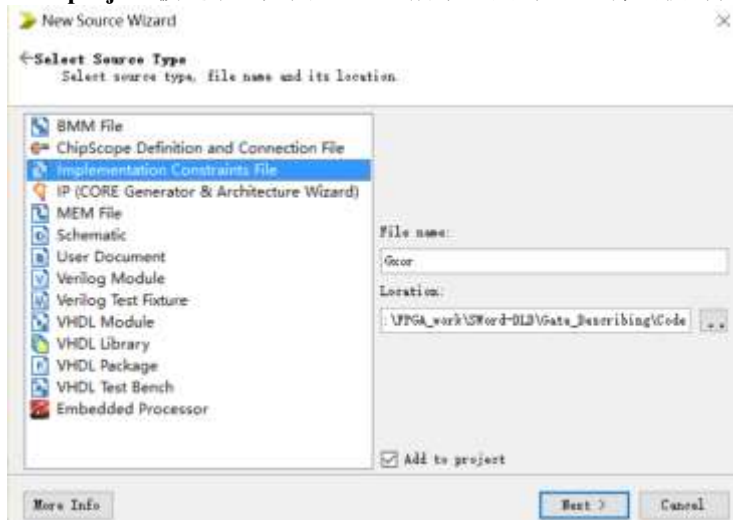


图 A.2-19 建立用户约束文件

检查无误后点击 **Finish**，此时在设计模块管理窗口的 **Gxor.v** 下面出现关联文件 **Gxor.ucf**，同时打开文档编辑，如图 A.2-20 所示。

参考第一章引脚介绍或查阅 **SWORD** 开发平台手册可用的 **GPIO** 资源有 16 位拨动开关和二个三色 LED 指示灯（后继实验将对引脚约束直接分配，不再一一列出）：

#三色信号灯：Tri_LED

NET "LEDR0" LOC = U21 | IOSTANDARD = LVCMOS33 ;

NET "LEDG0" LOC = U22 | IOSTANDARD = LVCMOS33 ;

NET "LEDR1" LOC = V22 | IOSTANDARD = LVCMOS33 ;

NET "LEDG1" LOC = U24 | IOSTANDARD = LVCMOS18 ;

NET "LEDR1" LOC = U25 | IOSTANDARD = LVCMOS18 ;

NET "LEDG1" LOC = V23 | IOSTANDARD = LVCMOS18 ;

#switch

NET "SW[0]" LOC = AA10 | IOSTANDARD = LVCMOS15 ;

NET "SW[1]" LOC = AB10 | IOSTANDARD = LVCMOS15 ;

NET "SW[2]" LOC = AA13 | IOSTANDARD = LVCMOS15 ;

NET "SW[3]" LOC = AA12 | IOSTANDARD = LVCMOS15 ;

NET "SW[4]" LOC = Y13 | IOSTANDARD = LVCMOS15 ;

NET "SW[5]" LOC = Y12 | IOSTANDARD = LVCMOS15 ;

NET "SW[6]" LOC = AD11 | IOSTANDARD = LVCMOS15 ;

NET "SW[7]" LOC = AD10 | IOSTANDARD = LVCMOS15 ;

NET "SW[8]" LOC = AE10 | IOSTANDARD = LVCMOS15 ;

NET "SW[9]" LOC = AE12 | IOSTANDARD = LVCMOS15 ;

NET "SW[10]" LOC = AF12 | IOSTANDARD = LVCMOS15 ;

NET "SW[11]" LOC = AE8 | IOSTANDARD = LVCMOS15 ;

NET "SW[12]" LOC = AF8 | IOSTANDARD = LVCMOS15 ;

NET "SW[13]" LOC = AE13 | IOSTANDARD = LVCMOS15 ;

NET "SW[14]" LOC = AF13 | IOSTANDARD = LVCMOS15 ;

NET "SW[15]" LOC = AE10 | IOSTANDARD = LVCMOS15 ;

分配“异或门”二个输入信号 **a**、**b** 可映射到开关(Switch)最低二位 **SW[1:0]**，输出信号 **F** 映

A.2.5 建立用户约束

射到三色 LED 红色端，其余暂时不用引脚用“#”注释掉，编辑完成后的引脚约束如图 A.2-20 所示，保存编辑结果即可。

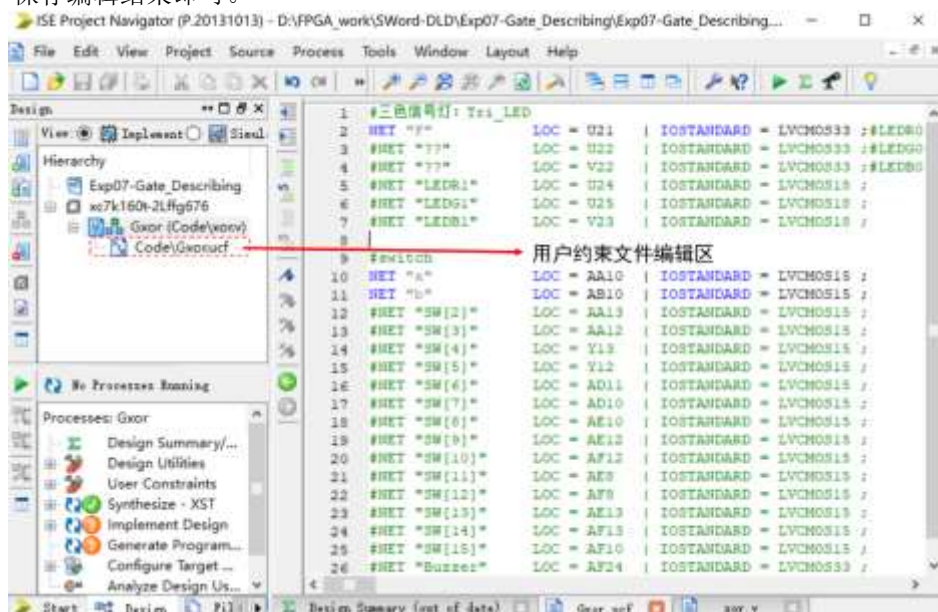


图 A.2-20 关联与编辑用户约束文档 ucf

PlanAhead 辅助建立用户约束

也可以用 PlanAhead 辅助建立用户约束。首先在 ISE 设计(Design)窗口的 View 选项选中 Implementation，从 Simulation 窗口切换回 Implementation 窗口，并选中 xor.v 文件。

在进程运行管理(Processes Running)窗口展开用户约束 (User Constraints) 子菜单，双击 “I/O Pin Planning (PlanAhead) – Pre Synthesize”或“I/O Pin Planning (PlanAhead) – Post Synthesize”二者之一都可以，系统将激活或综合后激活 PlanAhead 软件，如图 A.2-21 所示，点击 “Yes”即启动 PlanAhead 软件并自动建立.ucf 文件。

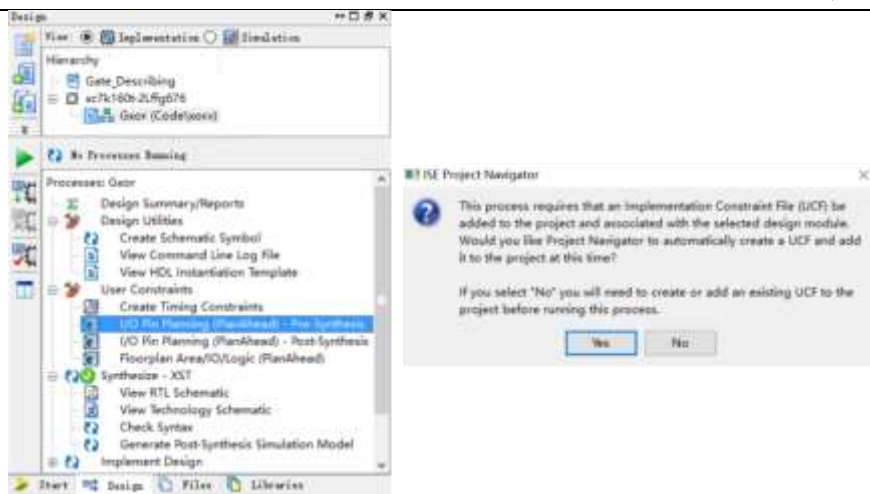


图 A.2-21 启动 PlanAhead 软件工具

PlanAhead 工具启动欢迎界面如图 A.2-22 所示,系统需要分析 ISE 工程并导入网表文件,这个启动过程有点长,结束后点击 Close 关闭欢迎界面即可看见如图 A.2-23 所示的 PlanAhead 用户约束配置界面。



图 A.2-22 PlanAhead 工具 Welcome 启动界面

PlanAhead 工具界面内容非常丰富,主要分四个区域,I/O 端口约束区的“Site”处可以配
~ 501 ~

A.2.5 建立用户约束

置用户引脚约束。

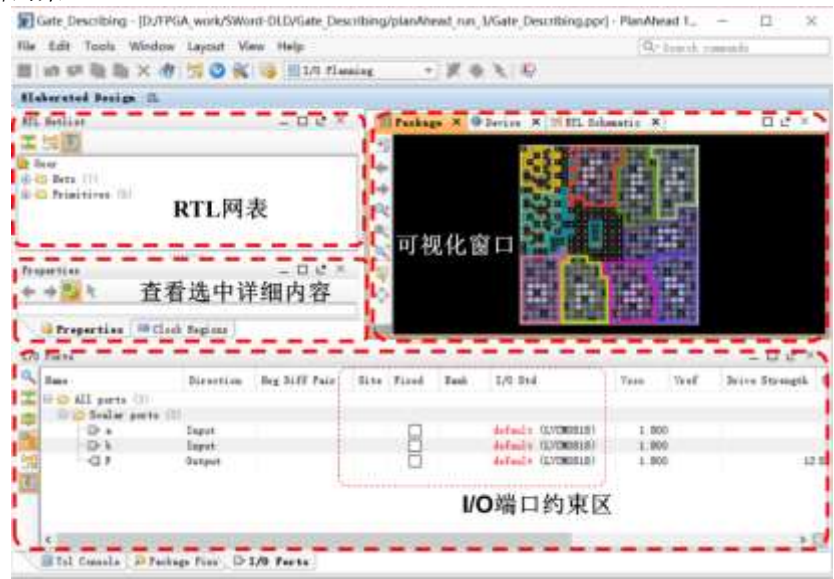


图 A.2-23 PlanAhead 约束配置界面

在 I/O 端口约束区的 Site、I/O Std 处可以配置用户引脚约束，点击 Site 即可弹出下拉式 FPGA 输出端口的选项，选择或输入引脚位置、电压标准等信息配置非常方便。在 Site 处输入引脚位置信息同时，在可视化窗口的封闭图(Package)立即显示引脚物理位置。

需要配置的约束信息如下：

NET "LED0"	LOC = U21	IOSTANDARD = LVCMOS33 ;
NET "SW[0]"	LOC = AA10	IOSTANDARD = LVCMOS15 ;
NET "SW[1]"	LOC = AB10	IOSTANDARD = LVCMOS15 ;

配置完成后的约束结果如图 A.2-24 所示，点击保存即完成了用户引脚约束文件。



图 A.2-24 配置完成的用户引脚约束

异或门仅是一个简单的组合电路没有时钟等其他约束，此处直接做模块的引脚约束，对应引脚约束的封装视图如图 A.2-25 所示。

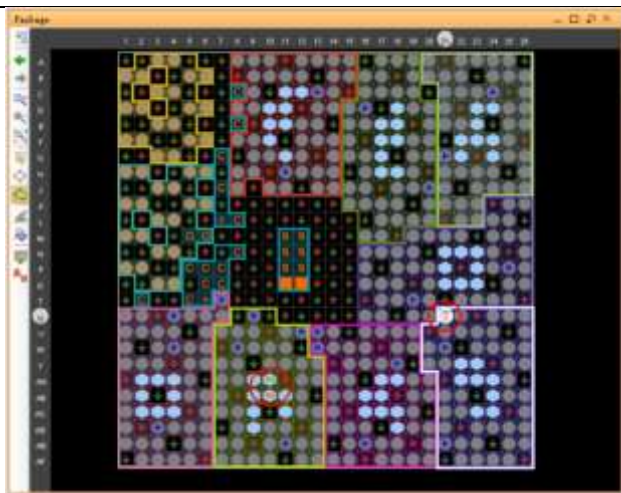


图 A.2-25 XC7K160T 封装图-绛红色圈住的为已约束端口

返回 ISE 点击 Gxor.ucf 可以打开看看 PlanAhead 自动生成的约束文件，有点凌乱，后续编写用户约束直接在 ISE 用文本编写，PlanAhead 软件工具还有其他功能请读者参考资料或 Help 深入学习，这里不再展开。

A.2.6 设计实现

FPGA 设计实现(Implement Design)是指通过 HDL 设计描述的电路综合、仿真和用户约束后生成的逻辑层网表文件做进一步的翻译(Translate)、映射(Map)和布局布线(Place & Route)，生成 FPGA 的底层模块编程信息，主要分三步，如图 A.2-26 所示。



图 A.2-26 FPGA 实现菜单

A.2.7 生成 FPGA 编程的流文件

Translate: 将综合后输出的逻辑层网表翻译转换为 FPGA(Xilinx)特定的底层编程信息和特定结构;

Map: 将翻译后的底层结构和编程信息映射到指定的 FPGA 器件上, 如 XC7K160T, 如果在建立工程时芯片型号选择不对, 在这一步会出错;

Place & Route: 根据用户约束和物理约束要求对设计模块进行布局分配和布线连接, 生成 FPGA 的配置(编程)文件。

运行设计实现只要在 View 选项中选择 Implementation 并选中 xor.v 文件, 在进程运行管理(Processes Running)窗口双击 Implement Design 菜单, ISE 自动运行翻译、映射和布局布线。

完成运行后可以查看布局布线后 FPGA 引脚分配结果及内部的互连结构、Slice 资源使用情况、时序等相关信息和布线后仿真和优化等操作。在 ISE 的 Design Summary/reports 中可以查看 Synthesize 和 Implement Design 的详细信息, 也可通过 PlanAhead 深入分析和优化。

A.2.7 生成 FPGA 编程的流文件

FPGA 编程上通过 JTAG 边界扫描协议下载编程的, 因此要将设计实现(Implement Design)后的编程信息文件转换为.bit 格式的流文件。

完成 Implement Design 后, 保持设计(Design)窗口选项不变, 在进程运行管理(Processes Running)窗口双击 Generate Programming File(产生编程文件), ISE 自动生成 FPGA 编程用的流文件, 如图 A.2-27 所示。

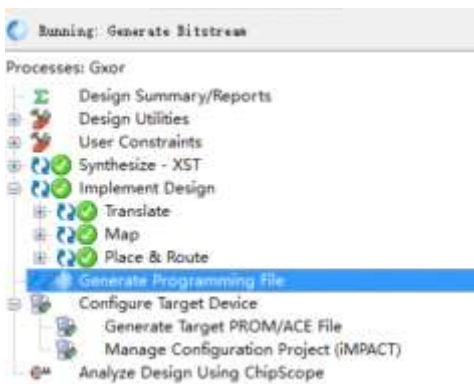


图 A.2-27 生成 FPGA 下载时的流文件

A.2.8 FPGA 编程

FPGA 编程有二种方案：

1. 直接通过 JTAG 协议下载.bit 文件到 FPGA 进行 FPGA 硬件编程，掉电后下载的编程内容即掉失；

2. 通过 JTAG 协议下载.bit 文件到 FPGA 配置存储器 SPI Flash(W 25Q64F)，每次加电或 FPGA 复位时对 FPGA 进行现场硬件编程。

二种方案都需要通过 JTAG 下载调试器，大部分支持 USB 的 Xilinx JTAG 都支持 SWORD 开发平台，如图 A.2-28 所示是早期使用和现代常用的 Xilinx USB 下载调试器。



图 A.2-28 支持 SWORD 开发平台的 JTAG 下载器

编程步骤：

1) 用 USB 线连接 PC 计算机和 SWORD 开发板 J11 或 CN7 的 JTAG 接口，查检无误后接通电源并打开开关 DSW16。

2) 在进程运行管理(Processes Running)窗口用鼠标点击“+ **Configure Target Device**”前的“+”号扩展配置目标器件菜单视图，如图 A.2-29所示。

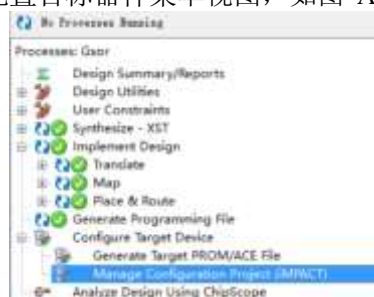


图 A.2-29 展开后的Configure Target Device菜单

3) 双击 **Manage Configuration Project (iMPACT)** 选项，弹出打开管理配置工程窗口，弹出如图 A.2-30的*iMPACT* 集成管理界面，也可直接双击**Configure Target**

Device。注意：不同版本略有不同。

4) 点击iMPACT Flows内部任意处激活窗口面板，再双击“Boundary Scan(边界扫描)”选项，此时在右边下载管理窗口由灰变白，如图 A.2-31所示激活了Boundary Scan管理窗口。

5) 在Boundary Scan管理窗口点鼠标右键，选中Initialize Chain点击，iMPACT工具自动初始化JTAG链并识别目标器件，如图 A.2-32所示，同时弹出是否需要分配下载文件对话框。

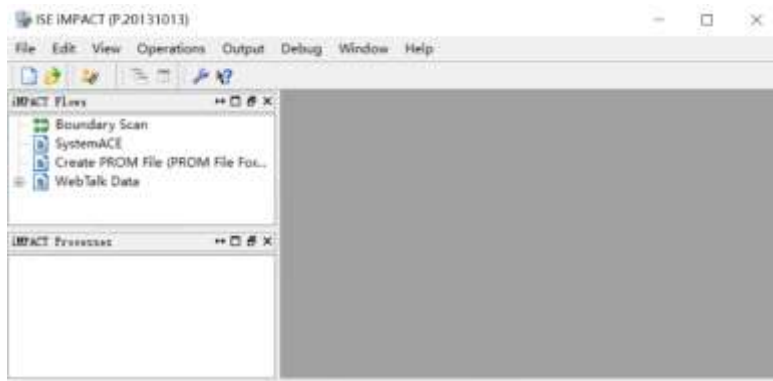


图 A.2-30 iMPACT集成管理界面



图 A.2-31 激活Boundary Scan管理窗口

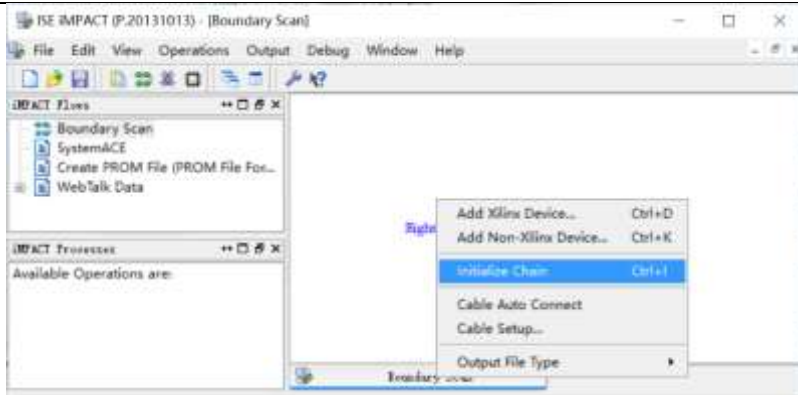


图 A.2-32 初始化JTAG扫描链

6) 分配下载文件。在Auto Assign Configuration Files Query Dialog对话框中，点击Yes，可以看到已经识别的目标芯片和下载链关系。



图 A.2-33 完成JTAG扫描链自动初始化

同时弹出请求分配FPGA新配置文件窗口，如图 A.2-34所示，选择正确目录下需要下载的FPGA编程流文件gxor.bit，点击Open打开流文件，此时gxor.bit文件出现在JTAG链中FPGA芯片的下方。

A.2.8 FPGA 编程

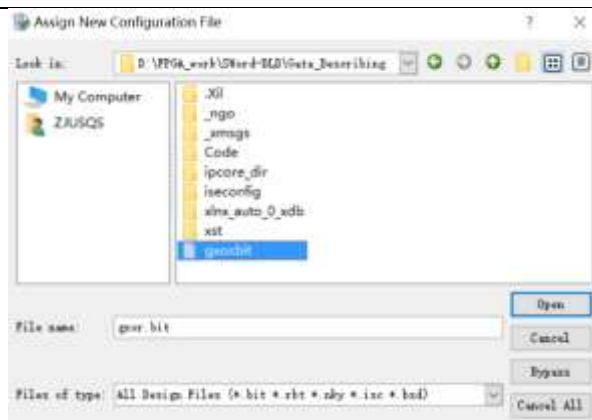


图 A.2-34 分配新的FPGA编程配置文件

同时又弹出Attach SPI or BPI PROM对话框，如图 A.2-35所示，询问是否需要同时附带下载配置存储器文件。

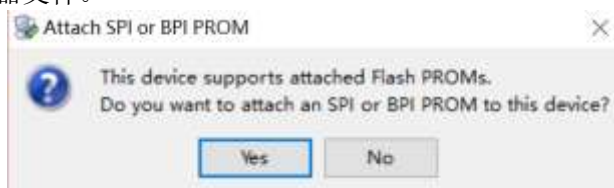


图 A.2-35 附带配置存储器对话框

目前不需要下载配置存储器文件，故点击“**No**”跳过PROM文件选择，弹出编程属性对话框，如图 A.2-36所示。.bit下载不需要校验，故直接点OK即可。

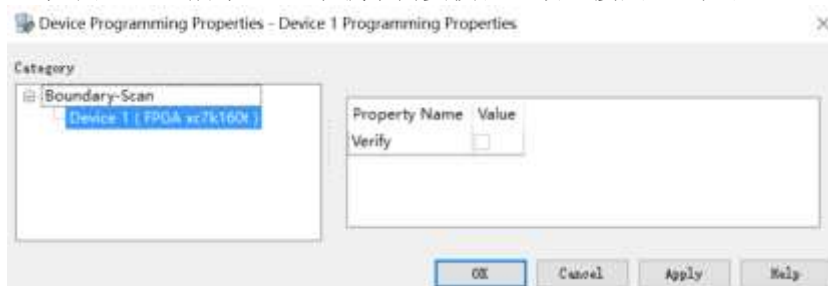


图 A.2-36 编程属性对话框

7) FPGA编程下载。

在Boundary Scan管理窗口用鼠标选中FPGA芯片图标点鼠标右键，弹出浮动菜单窗口，选中Program(编程)选项，如图 A.2-37(a)，iMPACT开始通过JTAG下载器下载.bit文件，对FPGA直接进行硬件编程，如图 A.2-37(b)所示。

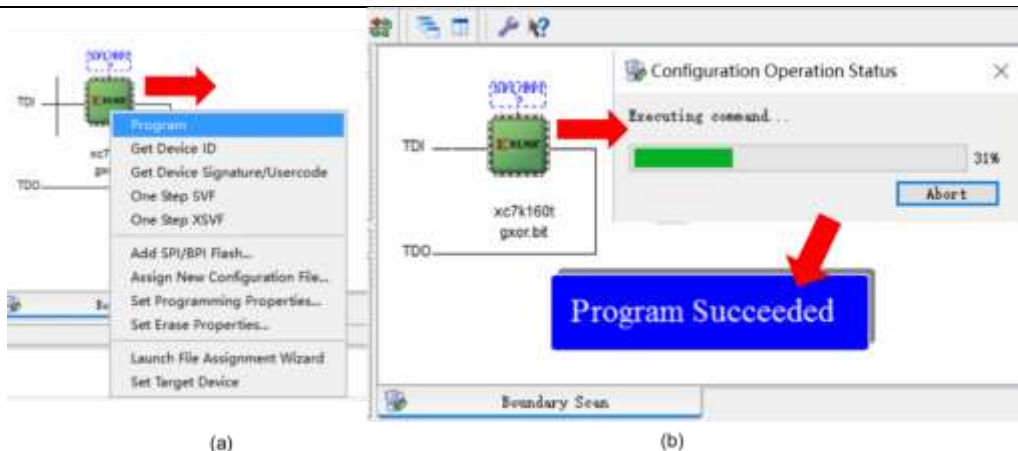


图 A.2-37 iMPACT通过JTAG对FPGA编程过程

完成FPGA下载编程后，在Boundary Scan管理窗口会出现“Program Succeeded”表明编程正确，否则就出现“Program Failed”，表示没有正确完成FPGA编程。

A.2.8.1 物理验证

根据 I/O 用户约束时定义的交互按钮和显示，在 SWORD 开发板通过拨动开关，如图 B-38 所示，观察 LED 灯颜色变化物理验证设计是否正确成功，验证结果填入错误!未找到引用源。。

三色 LED 物理验证应符合：红色 = 0、蓝色 = 1，如果反之将是什么结论？

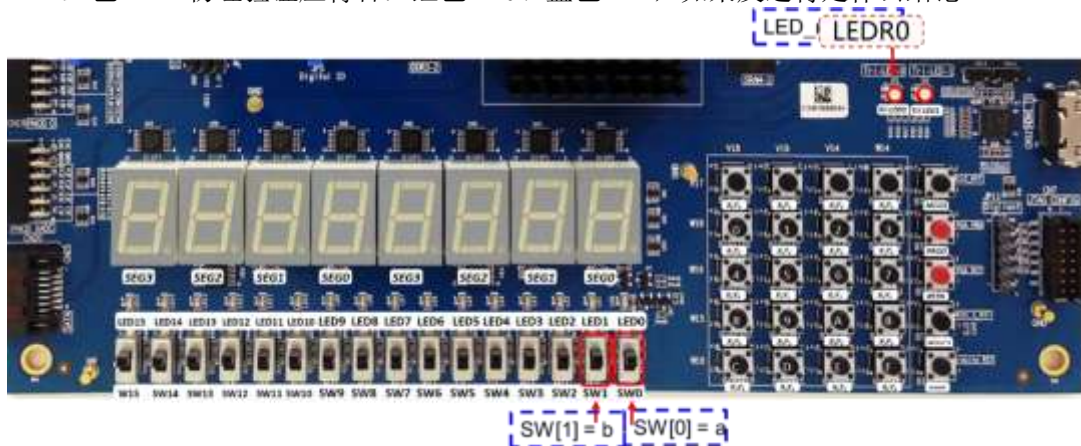


图 B-38 异或门输入输出接口分配

A.3 逻辑图描述开发操作与实现

操作与实现

电路实验实现操作与【实验四】类同，主要区别是原理图输入编辑部分，此处对相同部分仅说明参数，重点叙述不同部分。

A.3.1 【步骤1】建立 ISE 工程

同【实验四】。点击左上方 file 菜单，选择 new project 新建一个 project，弹出如图 A.2-1 所示新工程建立向导 New Project Wizard。

命名和参数设置：

Name: Lamp_C，不能作用中文字符，C 表示组合电路实现；

Location: “工程位置”。可建立或选择工程所在目录，缺省是上次填写的位置，建议同一个项目所有工程在一个目录下；

Working Directory: “工作目录”，改变 Location 时会自动改变；

Description: 描写工程内容，可省略；

Top-level source type: 此工程顶层使用原理图描述，与【实验四】不同下拉选择 Schematic。

完成设置后点击“Next”进入到设备属性窗口，如所示图 A.2-2 所示。与【实验四】完全相同，设备属性主要是 FPGA 芯片参数，重列如下：

Family: 根据实验板的型号来选择，(选择 Kintex7/SPARTAN-3)。

Device: 根据实验板上的 FPGA 芯片型号来选择，(选择 XC7K160T/XC3S200)。

Package: 根据实验板上的 FPGA 芯片型号来选择，(选择 FFG676/FT256)

Speed: 根据实验板上的 FPGA 芯片型号来选择，(选择-2L/-4)

Simulator: 选择 ISim(缺省)，ISE 自己的仿真模拟软件。也可选择第三方，但要另行安装。

完成详细的属性设置后，点击 NEXT，弹出工程属性核对窗口如图 A.3-1 所示，注意与【实验四】不同处。

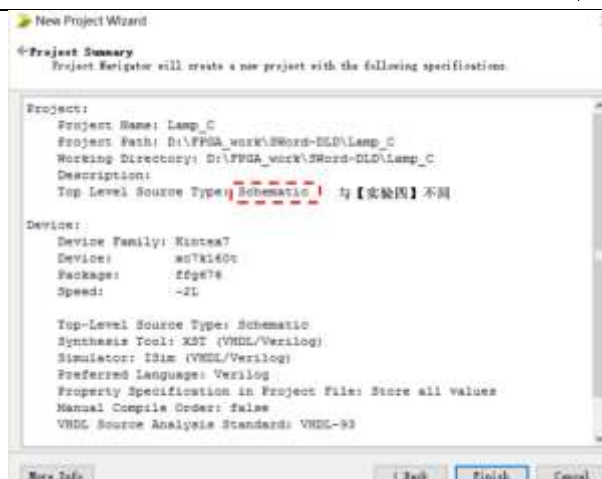


图 A.3-1 Lamp_C 工程属性核对窗口

核对无误后点击 Next，系统弹出工程模板窗口，除工程名以外均与错误!未找到引用源。相同，如图 A.2-4 所示。

A.3.2 【步骤 2】创建编辑模板及输入设计

这一步创建原理图编辑调试模板并输入原理图，模块命名：Lamp_C.sch。

具体操作如下：

1. 创建原理图编辑调试模板

1) 点击左上方 Project 菜单选择 **New Source** 或在 **Source** 窗口下空白处用鼠标右键点击 **New Source**，如错误!未找到引用源。图 A.2-5 所示。

2) 点击后，出现如图 A.3-2 所示新 Source 模板向导窗口，选择 **Schematic** 作为源类型、输入文件名 **Lamp_C**、确认 **Add to project** 被选中。

A.3.2 【步骤2】创建编辑模板及输入设计

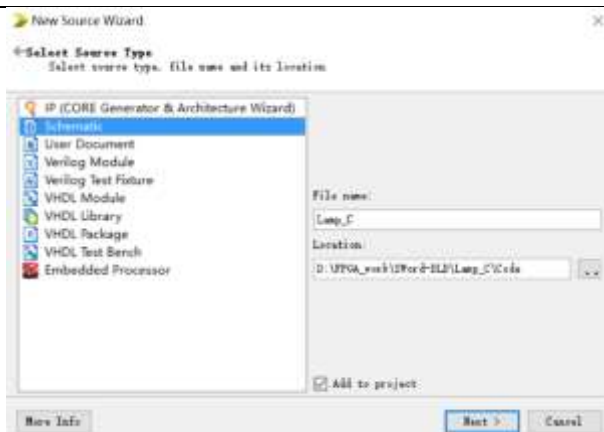



图 A.3-2 选择 Schematic 作为输入源类型

点击 Next, 接着点 Finish 弹出如图 A.3-3 所示原理图编辑窗口, 与错误!未找到引用源。不同的是过程中没有出现模块 I/O 输入窗口, 模块接口信号需要用工具栏图标  在原理图中标记。工具栏图标在图 A.3-3 作了给出了标注说明, 详细说明参考第一章相关内容。

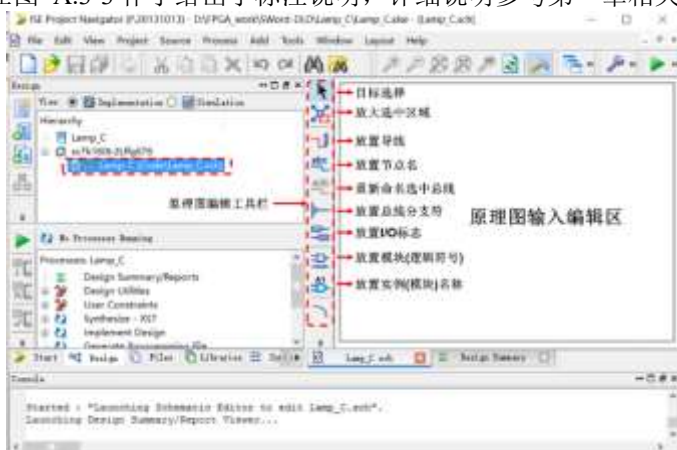


图 A.3-3 原理图编辑模板窗口

在原理图模板编辑窗口选用 ISE 器件库(相当 HDL 内置门)设计并编辑输入错误!未找到引用源。所示的楼道灯控制电路, 并标记模块 I/O 端口信号, 输入编辑完后保存 Lamp_C.sch 文件。

2. 原理图输入与编辑

输入 ISE 器件库模块或自制模块在原理图模板编辑窗口布局。自制模块需要创建模块逻辑符号并复制到工程根目录或加入到 ISE 器件库。后续设计中将介绍如何设计自制模块符号并调

用。

1) 逻辑模块输入(基本门)。在工程管理状态栏, 点击右向箭头选中 Symbols 界面, 进入选择器件窗口。在符号名过滤(Symbol Name Filter)处分别键入 inv、and3 或 or4, 选择电路所需器件“非门”、“与门”和“或门”, 拖入原理图编辑窗口, 如图 A.3-4 所示。

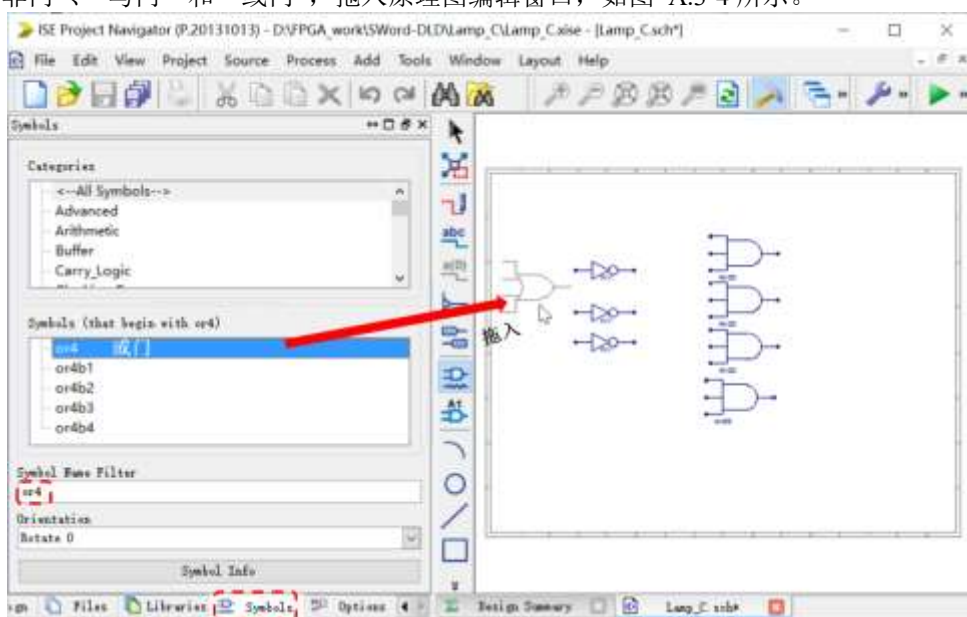



图 A.3-4 原理图输入与编辑-逻辑器件

在器件窗口中选择所需的器件拖到图形输入窗口, 当一个器件被安置后按下 ESC 键或点击  取消, 以便选择下一个器件。用鼠标右键点击模块符号可以相看修改属性, 如图 A.3-5 (a) 所示。在图 A.3-5 (b) 属性对话框中一般需要修改实例名(InstName)作为原理图中器件索引(ID), 如将自动生成的 XLXI_40 修改为 GAND1, 此例比较简单明了可以不修改。选中 Visible 中的方框对应属性可在原理图上显示, 并在原理图上移动和修改单项属性。

属性中还有一个关键项就是 VeriModel, 此处是 AND3, 这是逻辑符号对应的调用实例模块名, 对应 Verilog HDL 描述模块, ISE 综合时就是通过这模块名来调用模块实例的。本例调用的是 ISE 库描述器件, 系统自动生成, 如果是自制逻辑符号, 需要在此项指出调用的模块名。VhdlModel 也同样只是对应 VHDL 语言。

A.3.2 【步骤2】创建编辑模板及输入设计

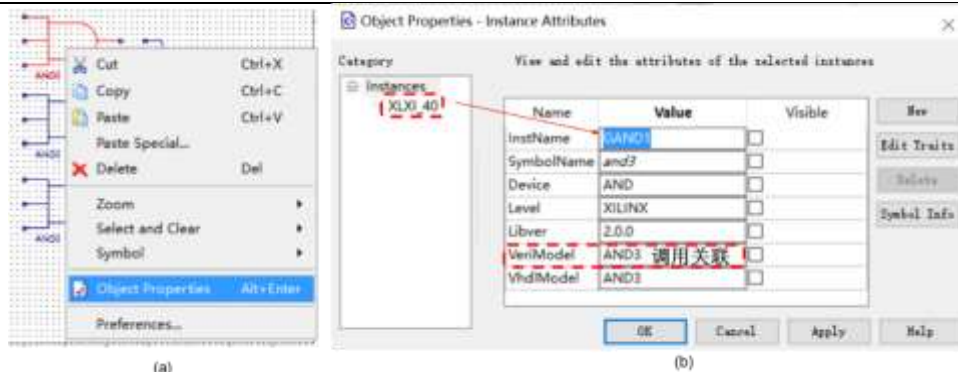





图 A.3-5 模块符号属性

2) 模块连接和 I/O 标记

点击  图标进入模块连接状态，用导线(add wire)连接逻辑模块端口，对顶层输入/输出需要加上相应的缓冲标记(add IO Marker, buffer 和 Iomake) 并修改命名。

当鼠标靠近模块可导线端口，端口处会自动出现口(针对 Net)或  (针对总线)图形符以表示可以连接。用鼠标左键点住口或  并移动鼠标至另一端口或适当位置释放鼠标左键，系统自动连接二个端口点或布线至鼠标位置。

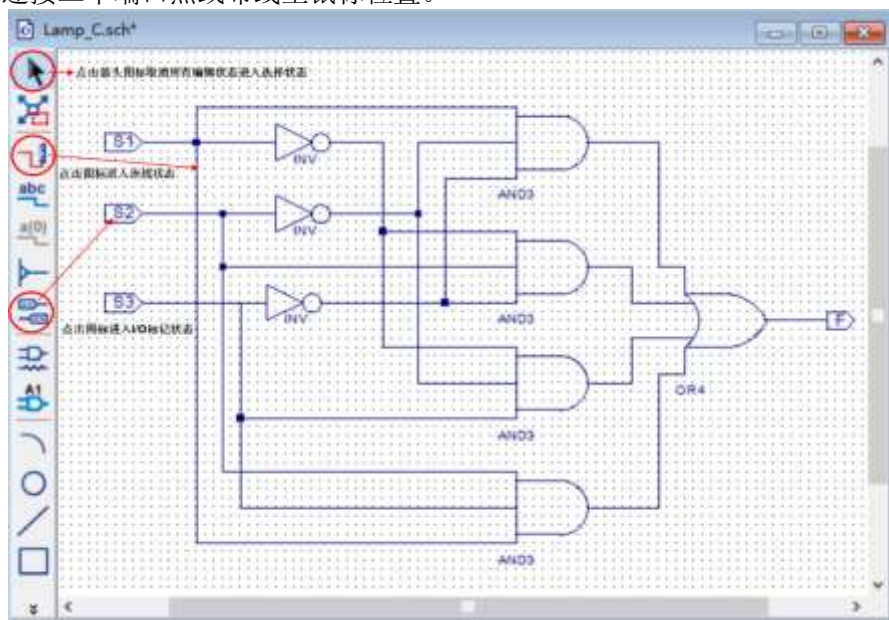







图 A.3-6 模块之间用导线(Net)连接并标记端口信号

模块端口连线相当于 HDL 描述时的端口信号列表，在原理图上需要做 I/O 标记，以告知 ISE 综合器。点击  图标进入端口信号标记状态，当鼠标靠近需要标记端口处会自动出现  (针对 Net) 或  (针对总线) 图形符以表示可以标记，点击即可标记，系统自动标记  或 ，同时以导线名作为标记名，点击标记符可以修改标记名，但导线名同时修改。

布线标记完毕的楼道灯控制器电路如图 A.3-6 所示，在 Tools 菜单选择 Check Schematic 可检查连线规则，检查无错误后保存原理图文件。


原理图中一位 wire 也称为 Net，多位 wire 也是 Net，称为 BUS，用  图标分离出子总线。双击导线(Wire)可以修改名称和宽度(大于 1 位成为总线)，如果是端口 Net 则增加了导线极性(PortPolarity)属性，如图 A.3-7 所示。



图 A.3-7 导线 Wire(Net)属性

注意：与 HDL 描述总线用 “[]” 不同，在原理线中总线宽度是用圆括号“()”表示，HDL 语法检查无法查出错误。总线的连接在后续设计用到再介绍。

3) 设计实用工具(Design utilities)

在 Processes Running 进程运行管理窗口还有“设计实用工具(Design utilities)”子项，点击 Design utilities 前的+号展开菜单，提供了五项很有用的小功能，如图 A.3-8 左边所示：

- 建立电路符号(Create Schematic Symbol)：自动生成选中模块的逻辑电路符号(Symbol)，符号文件后缀是.sym，可以打开此文件修改，同 ISE 内置库门电路一样在原理图输入时调用。比直接用 Tools 菜单下的 Symbol Wizard(符号生成向导)要方便。
- 查看命令行日志文件(View Command Line Log File)：保存了 ISE 操作的命令，可以查看学习命令行。

A.3.2 【步骤2】创建编辑模板及输入设计

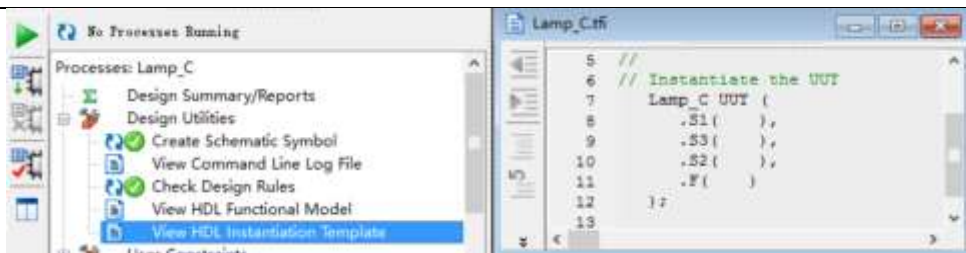


图 A.3-8 设计实用工具与实例模板

- c. 电路设计规则检查(Schematic Design Rule Check, DRC): 检查输入/输出标准合法性、时钟输入脚分配、参考电压(Vref)、工作电压(Vcco)是否兼容等。可以在 **Edit → Preferences** 打开首选项对话框下的 **category → Schematic Editor → Check** 对话框设置电路检查选项。如果不仅要检查当前电路, 而且还要检查相关电路(子模块), 需要通过运行 **Tools → Hierarchical Check**; 如果只检查当前电路运行 **Tools → Check Schematic** 即可。
- d. 查看 HDL 实例模板(View HDL Instantiation Template): 此项可以查看实例 HDL 调用结构, 提供 HDL 描述时调用格式, 在使用系统 IP 核或原理图描述模块时非常有用。
- e. 查看 HDL 功能模块(View HDL Functional Model): 提供了原理图描述模块对应的 HDL 描述, 对学习 HDL 门级、原语或结构化描述非常有帮助。此原理图对应的 HDL 描述代码如下:

```
module Lamp_C(S1,  
              S2,  
              S3,  
              F);  
  
  input S1;  
  input S2;  
  input S3;  
  output F;  
  
  wire XLXN_9;  
  wire XLXN_11;  
  wire XLXN_19;  
  wire XLXN_27;  
  wire XLXN_28;  
  wire XLXN_29;  
  wire XLXN_30;
```

```

INV  XLXI_28 (.I(S1),
               .O(XLXN_11));
INV  XLXI_29 (.I(S2),
               .O(XLXN_9));
INV  XLXI_30 (.I(S3),
               .O(XLXN_19));
AND3  XLXI_31 (.I0(XLXN_19),
               .I1(XLXN_9),
               .I2(S1),
               .O(XLXN_27));
AND3  XLXI_32 (.I0(XLXN_19),
               .I1(S2),
               .I2(XLXN_11),
               .O(XLXN_28));
AND3  XLXI_33 (.I0(S3),
               .I1(XLXN_9),
               .I2(XLXN_11),
               .O(XLXN_29));
OR4   XLXI_34 (.I0(XLXN_30),
               .I1(XLXN_29),
               .I2(XLXN_28),
               .I3(XLXN_27),
               .O(F));
AND3  XLXI_38 (.I0(S1),
               .I1(S3),
               .I2(S2),
               .O(XLXN_30));

```

endmodule

可以看出此 HDL 描述与**错误!未找到引用源。**的 HDL 内置门描述完全一样。实际上所有描述在顶层电路综合后都会生成 HDL Functional Model 代码，其文件后缀是.vf，综合后错误定位就是在这个文件上。

注：不要将.vf 文件拿来直接作为 Verilog HDL 设计文件，以免和系统的.vf 文件冲突，同时.vf 中的描述是一种门级结构描述，可以参考，但不是一种好的结构化描述。

A.3.3 【步骤 3】原理图设计综合

在模块管理窗口 View 选择 Implementation 并选中 Lamp_C.sch 模块文件，在 Processes Running 进程运行管理窗口双击“Synthesize-XST”，ISE 自动进行设计综合。

此操作与**错误!未找到引用源。**完全一样，同样也可双击 View RTL Schematic 查看分析 Lamp_C.sch 综合后的 RTL 层电路结构。

A.3.4 【步骤4】时序仿真测试

A.3.4 【步骤4】时序仿真测试

此操作与 A.2 的【实验四】例子完全相同，仅需设计楼道控制器 Lamp_C.sch 模块的仿真激励代码 Lamp_CTest.v。仿真激励代码参考如下：

```
.....  
integer i;                                //数值变量，用于循环变量  
initial begin  
    S1 = 0;  
    S3 = 0;  
    S2 = 0;  
  
    for(i=0;i<=7;i=i+1)begin  
        {S1,S2,S3} <= i;  
        #20;  
    end  
end
```

楼道灯控制器用上述激励仿真结果如图 A.3-9 所示，请分析仿真结果与设计功能是否一致。





图 A.3-9 楼道控制器 Lamp_C.sch 仿真结果

仿真波形表明此电路是一个三输入异或门,也可以调用【实验四】的二输入异或门来实现。

A.3.5 【步骤 5】建立用户约束

除文件名与开发板接口引脚分配不同,此操作与【实验四】完全相同。

首先在 ISE 设计(Design)窗口的 View 选项选中 Implementation,从 Simulation 窗口切换回 Implementation 窗口,并选中 Lamp_C.sch 模块文件。

点击左上方 Project 菜单选择 New Source 或在 Source 窗口下空白处用鼠标右键点击 New Source 如图 A.2-5 所示。

在新 Source 模板向导窗口,选择 Implementation Constraints File 作为源类型、输入文件名 Lamp 并确认 Add to project 被选中,类似【实验四】图 A.2-19 所示,点击 Next 弹出核对窗口。

检查无误后点击 Finish,此时在设计模块管理窗口的 Lamp_C.sch 下面出现关联文件 Lamp.ucf,同时打开文档编辑。

分配楼道控制器三个输入开关信号映射到 16 位开关(SWich)最低三位 SW[2:0],输出信号映射到三色 LED 红色端,其余暂时不用引脚用“#”注释掉。实际需要配置的约束信息如下:

```
NET "F"          LOC = U21    | IOSTANDARD = LVCMOS33 ;
NET "SW[0]"      LOC = AA10   | IOSTANDARD = LVCMOS15 ;
NET "SW[1]"      LOC = AB10   | IOSTANDARD = LVCMOS15 ;
```

编辑完成后的引脚约束如图 A.3-10 所示,保存编辑结果即可。

A.3.6 【步骤 6】设计实现

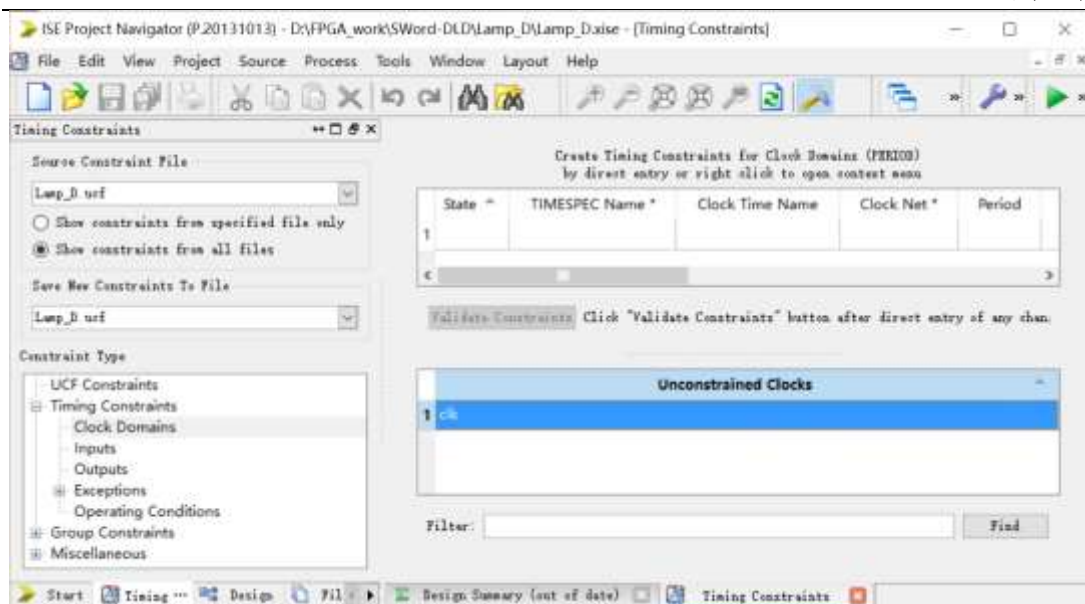


图 B-11 Timing Constraints 操作界面

时钟约束主要(Timing Constrains)主要包含三项:

- Clock Domains、Inputs、Outputs

A.4.1 时钟周期约束

现在首先对时钟做约束,选中 Clock Domains, 双击 unconstrained clock 窗口下面的 clk 信号, 将弹出如图 B-12 所示时钟周期约束对话框, 用来设置时钟周期、占空比和初始时钟边沿(上升沿(Rising)或下降沿(Falling))等时钟 clk 的参数, 缺省值周期是 20, 初始采用上升沿(Rising)、占空比是 50。

SWORD 平台输入时钟频率是 100MHz, 周期是 10ns, 故时钟参数修改为:

Time = 10

Rising Duty Cycle = 50

Initial clock edge = Rising(HIGH)

A.4.2 输入偏移时间约束

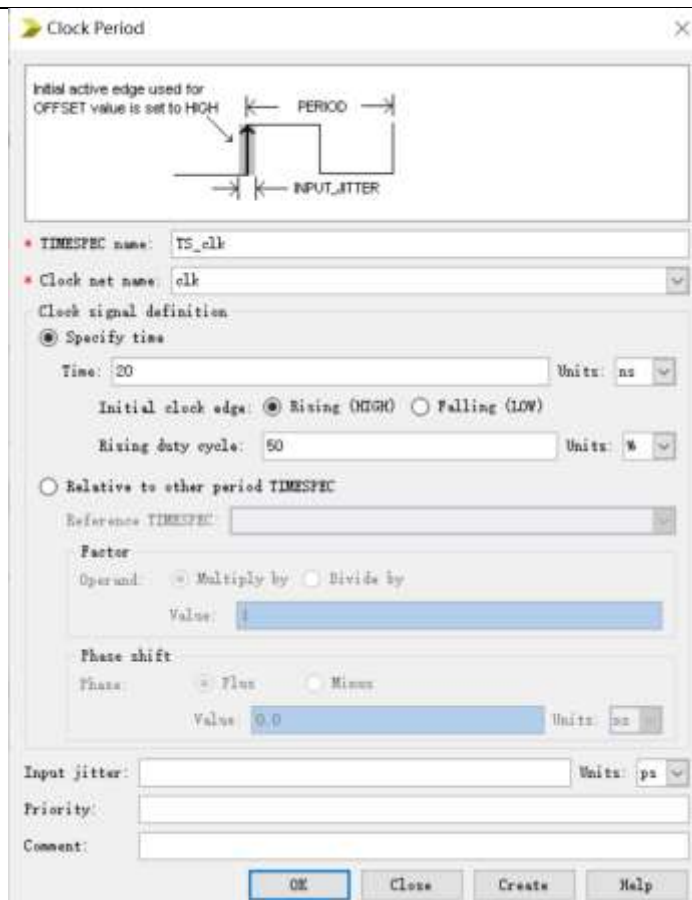


图 B-12 时钟约束对话框界面

点击 OK，在工作区上方显示设计好的时钟周期参数具体信息，并且可选择约束进行逐一修改。再点击保存图标系统将自动生成或在已有的 ucf 文件内增加如下时钟约束参数：

```
#Created by Constraints Editor (xc7k160t-ffg676-2l) - 2016/02/24
NET "clk" TNM_NET = clk;
TIMESPEC TS_clk = PERIOD "clk" 10 ns HIGH 50%;
```

如果时钟是从其他指定的时钟生成的，也可以指定生成时钟(Relative to other period TIMESPEC)的关系、输入抖动和优先级等。

A.4.2 输入偏移时间约束

在 Timing Constrains 菜单下, 选中 Input 约束项, 在右边出现窗口出现 unconstrained Input Port 将列出全部输入信号, 双击输入信号, 可以设置输入信号的时序约束, 双击“S1”弹出如图 B-13 所示输入信号建立时间(Setup)约束对话框。

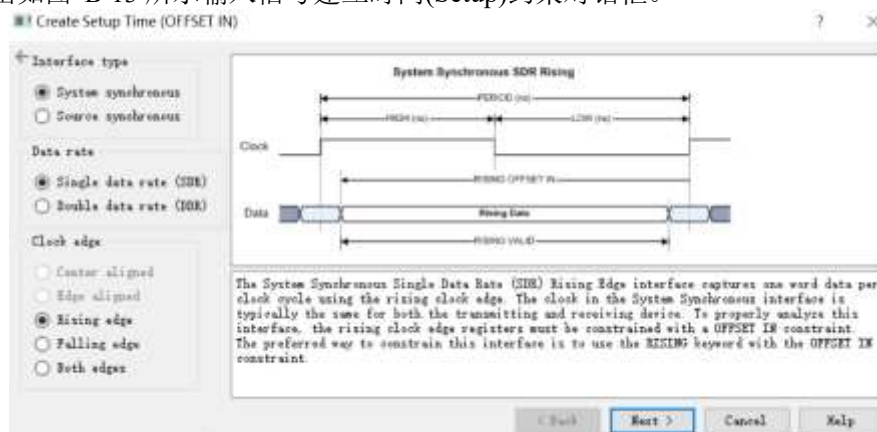


图 B-13 建立时间(Setup Time): 指定输入偏移(OFFSET IN)约束类型

这个对话框是指定同步类型, 缺省是 System Synchronous、Single data rate 和 Rising。

输入信号同步类型: System Synchronous(系统同步)或 Source Synchronous(源同步, 针对时钟的不同来源);

数据速率: Single data rate(单数据速率, SDR)或 Double data rate(双数据速率, DDR); 时钟边沿: Rising、Falling 或 Both edge。

点击 Next, 弹出输入信号 S1 输入偏移约束参数设定, 如图 B-14 所示。偏移约束主要包括输入(或输出)偏移, 约定时钟和数据与输入(或输出)管脚之间的时序关系, 不是用来约束内部逻辑的。

A.4.3 输出偏移时间约束

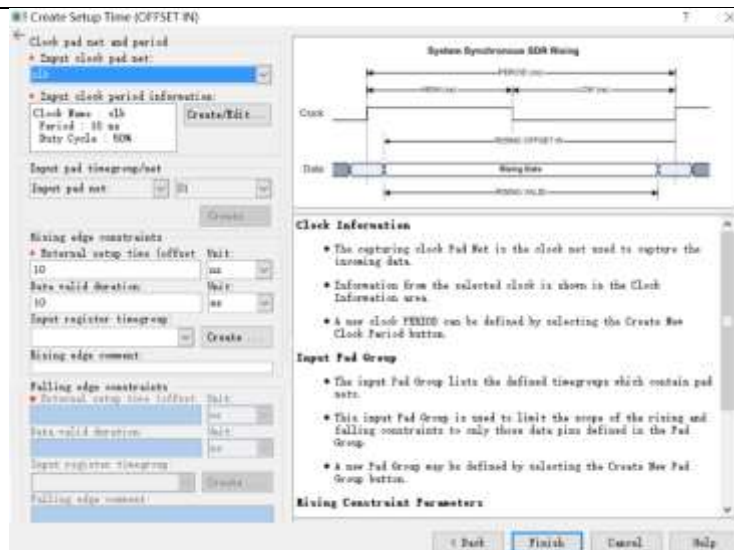


图 B-14 建立时间(Setup Time): OFFSET IN 约束设置

这时要设置的是:

Input pad timegroup/net: S1 不是 Group 选择 Input pad net;

External setup time(offset in): 建议设置 6-10 之间。这是 OFFSET_IN_BEFORE, 约束输入数据(S1)比有效时钟边沿提前多长时间建立好。FPGA 芯片内部与输入引脚(S1)相连的组合逻辑的延迟不能大于这个时间, 否则当时钟边沿来到时数据还没有建立或稳定有效, 触发器就不能正确采样到有效数据。这个值也不能设置太小, 否则 ISE 运行设计实现(Implement Design)时间会很长且有可能找不到优化路径而失败;

Data Valid duration: 这个时间是指输入数据有效保持时间, 在这个例子中 S1 相对于系统时间是非常长的, 可以用缺省值。

完成后点击保存图标, ucf 文件内增加以下内容:

```
NET "S1" OFFSET = IN 6 ns VALID 100 ns BEFORE "clk" RISING;  
NET "S2" OFFSET = IN 6 ns VALID 100 ns BEFORE "clk" RISING;  
NET "S3" OFFSET = IN 10 ns VALID 100 ns BEFORE "clk" RISING;
```

A.4.3 输出偏移时间约束

在 Timing Constrains 菜单下, 选中 Output 约束项, 在右边出现窗口出现 unconstrained Output Port 将列出全部输出信号, 双击输出信号, 可以设置输入信号的时序约束, 双击“F”弹出如图 B-15 所示输入信号建立时间(Setup)约束对话框。

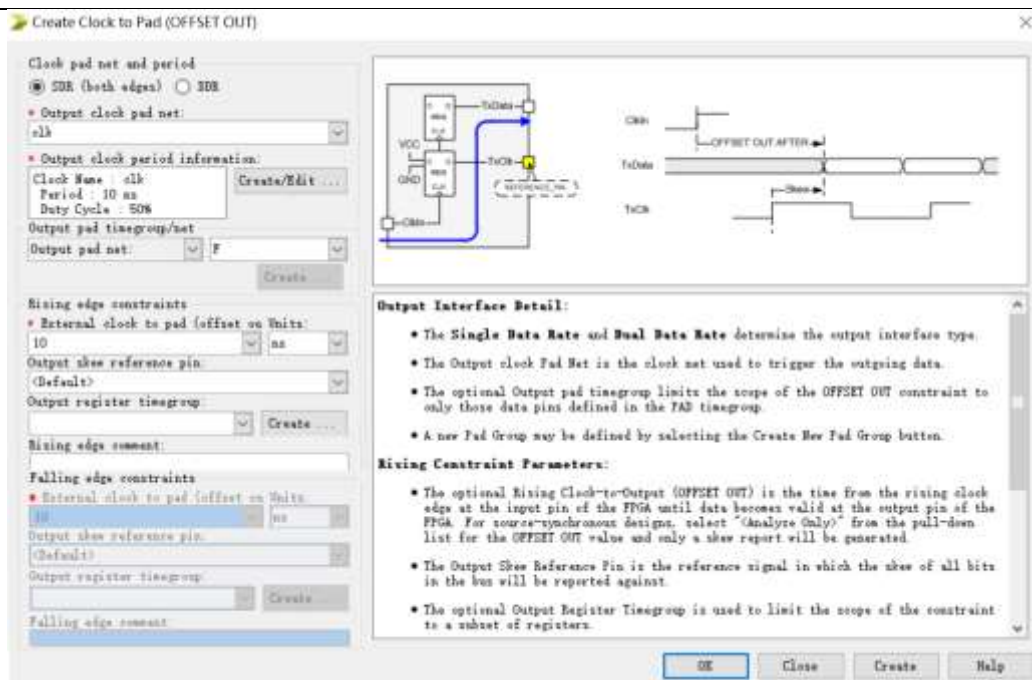


图 B-15 建立时间(Setup Time): OFFSET OUT 约束设置

这时要设置的是:

Output pad timegroup/net: F 不是 Group 选择 Output pad net;

External setup time(offset out): 建议设置 6-10 之间。这是 OFFSET_OUT_AFTER, 约束输出数据(F)在有效时钟边沿之后多久稳定下来, FPGA 芯片内部的输出延迟必须小于这个数值。如果输出信号连接到下一个输入端, 这个时间将影响下一级的输入端的延迟时间。在这个例子中将影响计数器的控制(参考综合 RTL 电路)。完成设置后 ucf 文件内增加下列内容:

```
NET "F" OFFSET = OUT 10 ns AFTER "clk" REFERENCE_PIN "F";
```

A.4.4 全局偏移约束

在选中输入输出偏移约束功能时, 在 unconstrained input/Output Port 窗口右边还会出现针对全局输入输出偏的时钟移约束窗口 “is constrained by a Global OFFSET IN/OUT”, 建立全局偏移约束, 是对所有输入输出信号做的约束, 这个过程与一般偏移约束相同。完成所有约束配置后打开 ucf 完整的时序约束如下:

```
#Created by Constraints Editor (xc7k160t-ffg676-2l) - 2015/07/24
```

A.4.4 全局偏移约束

```
NET "clk" TNM_NET = clk;                                #周期约束
TIMESPEC TS_clk = PERIOD "clk" 10 ns HIGH 50%;          #周期约束
NET "S1" OFFSET = IN 6 ns VALID 100 ns BEFORE "clk" RISING; #输入偏移约束
NET "S2" OFFSET = IN 6 ns VALID 100 ns BEFORE "clk" RISING; #输入偏移约束
NET "S3" OFFSET = IN 10 ns VALID 100 ns BEFORE "clk" RISING; #输入偏移约束
NET "F" OFFSET = OUT 10 ns AFTER "clk" REFERENCE_PIN "F";  #输出偏移约束
OFFSET = IN 6 ns VALID 100 ns BEFORE "clk" RISING;      #全局输入偏移约束
OFFSET = OUT 10 ns AFTER "clk";                          #全局输出偏移约束
```

这个例子中除了周期约束外其他都可以省略，一般主频不高或要求不高的时序电路可以不做偏移约束。

A.5 存储器 IP 核设计

在 FPGA 上实现存储器模块有多种方式，最简单的是用 HDL 描述二维寄存器变量，由 ISE 综合器帮助实现，这里以 8×16bit RAM 为例介绍用 ISE 核生成器实现。

1. 存储器核生成

【第一步】激活 ISE 核生成器

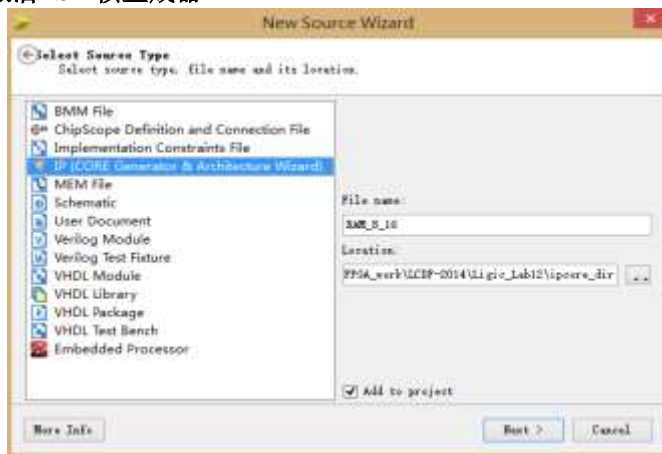


图 A.5-1 New Source 生成向导

参考新建源操作，在 ISE 设计窗口击鼠标右键或在集成菜单上从 Project 选择 New Source，弹出如图 A.5-1 所示新模块源生成向导窗口，选择 IP(CORE Generator & Architecture Wizard)。

在 File Name 处填入存储模块名 RAM_8_16(注意用小写字母)，选中 Add to project 左边的方框，点击 Next 激活核生成向导。

【第二步】选择 IP 核种类

在 Select IP 窗口，选择需要核的类型或体系框架，如图 A.5-2 所示，此时选择 RAMs & ROMs 下的“Block Memory Generator”。ISE 提供 Xilinx FPGA 支持的两种存储器核：Block Memory (块存储器)和 Distributed Memory(分布式存储器)，前者是独立的存储器模块，后者是利用分布在 FPGA 可配置模块(CLB)中的存储单元构成的模块，如查表器中的 RAM 资源。

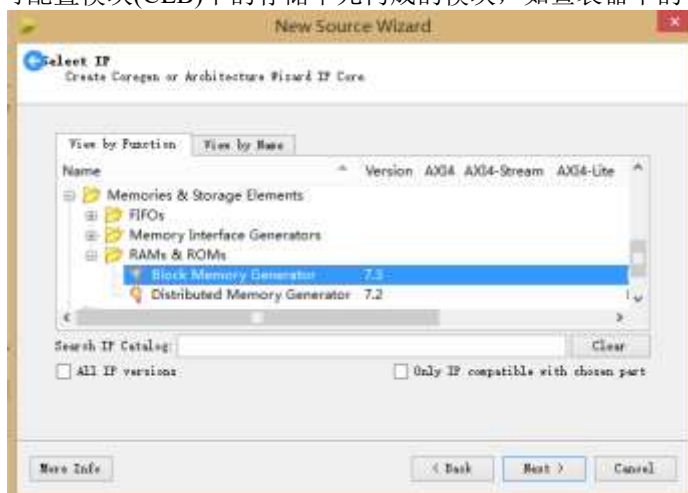


图 A.5-2 IP 核类型选择窗口

这里选择了块存储生成器，点击 Next 直接弹出参数设置窗口第 2 页，进入核参数配置窗口。

【第三步】存储器核参数设置：核类型

A.4.4 全局偏移约束



图 A.5-3 选择单端口块存储器类型

在 Memory Type 下拉选项中选择“Single Port RAM”,其余缺省设置, 点击 Next 弹出窗口第 3 页。

注意：块存储器存在地址锁存时钟。

【第四步】存储器核参数设置：空间和字宽

在参数设置窗口第 3 页根据所需存储器要求设置空间和字宽, 本例设计 8×16 的存储模块, 设置如下:

Write Width: 16(bit);

Write Depth: 8 (Word, 也称单元);

Write First: 选中;

Use ENA Pin: 选中。

其余选项默认, 这里也选中了使能项, 用于字扩展, 如图 A.5-4 所示。

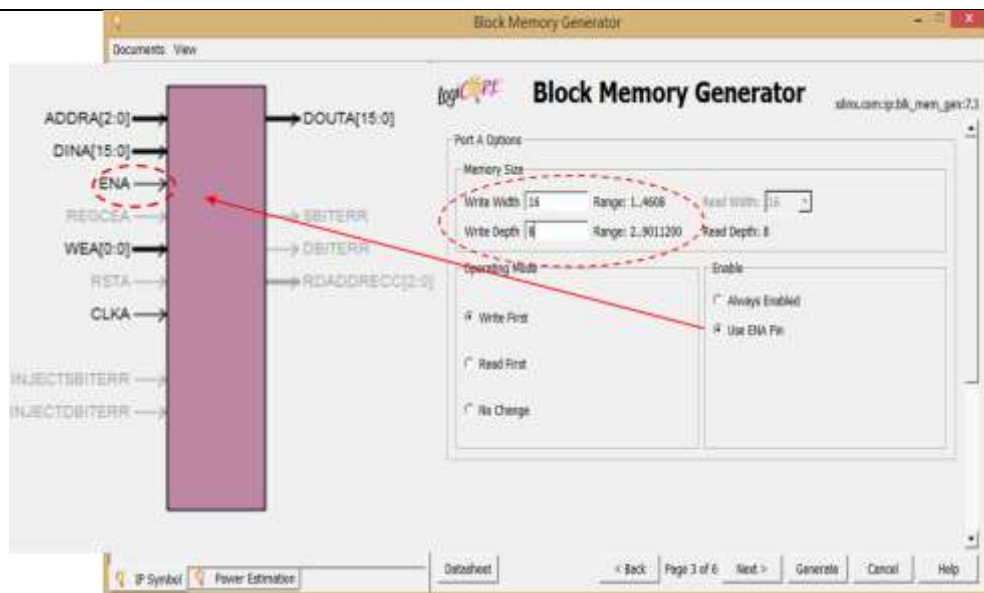


图 A.5-4 配置存储模块空间和字宽

【第五步】RAM IP 核初始化

完成存储参数配置后点击 Next，弹出窗口第 4 页，点击“Browse...”选择初始化关联文件（RAM 也可以初始化，这对调试非常有帮助，本实验无），其余不用修改，如图 A.5-5 所示。初始化内容可以点击 Show 选项查看，同时可以点击 Datasheet 查看核的相关资料，了解核使用的时序等参数。

【第六步】生成与调用

大多数存储器核只要设置上述基本配置就可以点击“Generate”，核生成器自动生成所需的存储器 IP 核，这个过程时间较长。核生成后自动加入工程并在设计窗口出现 RAM...xco 模块，ISE 生成的核调用与.v 模块调用完全相同。在 ISE 设计窗口选用该核，可以在进程窗口点击 View HDL Instantiation Template 查看核调用结构，如图 A.5-6 所示。

A.4.4 全局偏移约束

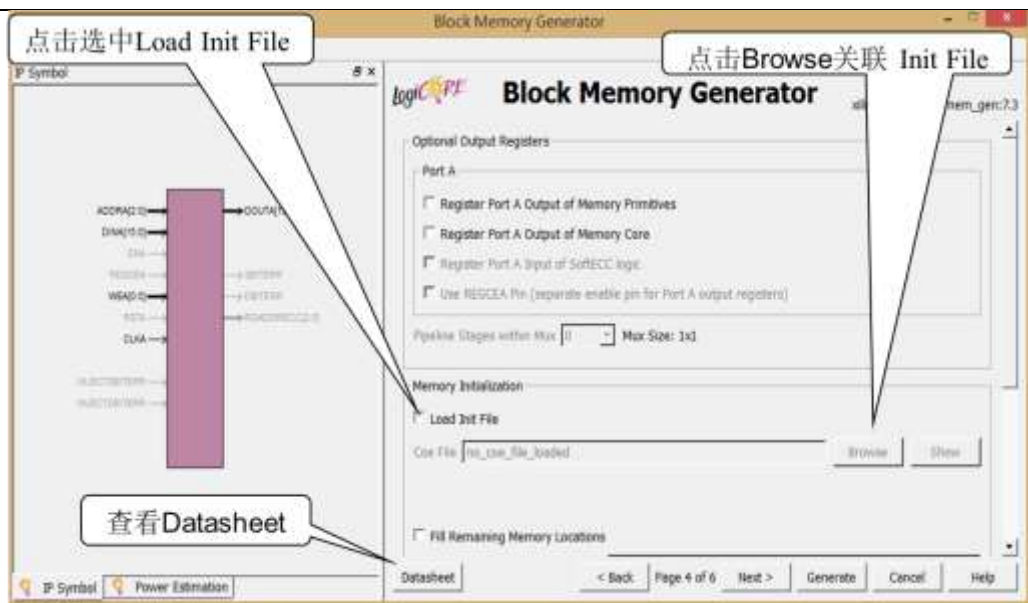


图 A.5-5 存储器核初始化

核相关的所有内容在当前工程的 ipcore_dir 目录下，核的逻辑符号是核同名.sym 文件 (RAM_32_32.sym) 也在这个目录中。这个逻辑符号图非常大，在实际使用时需要修改，修改后的逻辑符号最好另行保存，使用时自制到工程根目录，以免在重新生成核时被覆盖。

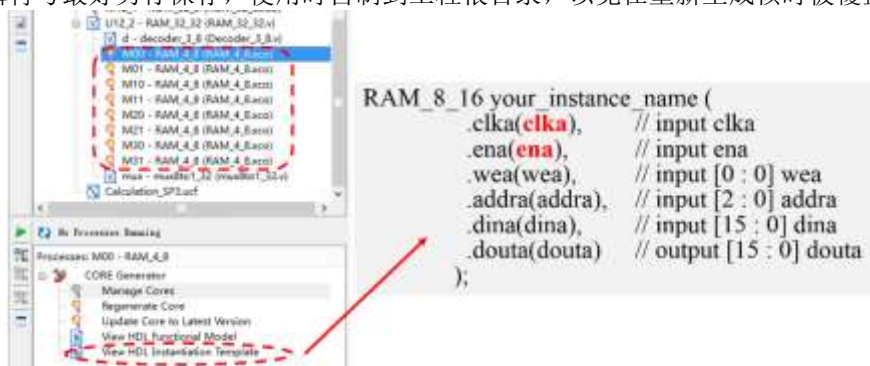


图 A.5-6 查看核调用结构

2. 存储器初始化文件

存储器这初始化关联文件后缀是.coe，可以用 ISE 打开编辑，也可以用普通文本编辑工具，其格式如下：

```
memory_initialization_radix=16;
```

```
memory_initialization_vector=
00000000, 11111111, 22222222, 33333333, 44444444, 55555555, 66666666, 77777777,
88888888, 99999999, AAAAAAAAAA, bbbbbbbb, cccccccc, dddddddd, eeeeeeee, ffffffff,
557EF7E0, D7BDFBD9, D7DBFDB9, DFCFFCFB, DFCFBFFF, F7F3DFFF, FFFDF3D, FFFF9DB9,
FFFFBCFB, DFCFFCFB, DFCFBFFF, D7DB9FFF, D7DBFDB9, D7BDFBD9, FFFF07E0, 007E0FFF,
03bdf020, 03def820, 08002300;
```

第一行说明是初始化参数向量采用 16 进制（也可以 2 进制）

第二行是初始化向量名

第三行开始是初始化向量元素，用逗号“,”分隔，分号结束。文件头、尾部可以用“#”号加注释，中间不可以加注释。存储器片内单元译码实现

为了简化存储器内部结构，这里采用[错误!未找到引用源。](#)(c)开关加三态门模拟存储单元，用 74LS139 实现实现[错误!未找到引用源。](#) (a)的存储器。

附录B 实验规范及要求

B.1 实验规范与要求

实验课与理论课有较大的区别，理论课教与学的互动是老师为主，主要的互动是讨论、演讲等，主角是老师。实验课则不同，教学互动的主角是学生，老师是指导学生根据理论知识和实践规律实现电路，学生自主设计、调度和实现电路，当实验发现问题时，教师引导学生检查电路故障。

现代电路实验大多采用计算机辅助设计工具和 EDA 芯片实现知识点的某个电路模块或需求的某种电路功能、部件或系统，其难度已经大大下降，没有繁琐的焊接、实物连线等机械工作，取而代之的是仿真、调试、测试和物理验证，基本不会出现物理器件损坏或电路板断路、短路及接触不良等问题，而造成在短时间排除不了故障而使实验完成不了。

由于硬件电路的特殊性、不可预测性及需要对电路实物进行操作，EDA 工具不可能做到发现所有错误或对错误准确定位，其实验过程与软件实验有很大的区别，因此完成实验时间还会存在不确定性，不同能力或经验积累的人实验时间也不一样。这就需要实验人员在实验课前预习，做到进实验室前做好电路物理验证调度的准备工作，实验课后整理完

善工作，否则仅靠实验课的时间是难以完成实验任务的。

同时硬件电路实验需要接触电、电器和电路板等，需要遵守实验规范和安全操作。

B.1.1 实验室规则

1. 实验前必须做好实验内容的充分预习，完成要求的预习任务，EDA 实验需要在进实验室前完成电路的预设计，一般要求做到 HDL 描述无语法错误或原理图描述无规则错误。写出简要的预习报告（可以将预习要求直接填写在本实验指导书的相关内容中）。实验前教师要求对学生的预习报告进行检查，没有预习的同学不能进行实验。
2. 使用仪器前，必须了解其性能、使用方法及注意事项，严格遵守掌握各种仪器的选择和使用。
3. 学生进入实验室，必须严格遵守实验室的各项规章制度，听从指导，服从管理。实验时，要遵守纪律，不迟到，不做与实验无关的事情，不动与本次实验无关的仪器和设备。实验时保持室内安静。
4. 实验结束后，每个学生都必须按要求写一份实验报告。并将实验报告和预习报告、实验中的记录的数据一起及时提交。

B.1.2 实验操作要求

1. 实验时按实验方案连接实验电路，认真连线，并经过检查确认无误后，才能接通电源。实验中接线，拆时应插拔芯片，应先关闭电源。
2. 接通电源后应首先观察有无破坏性异常现象（如仪器设备、无器件冒烟、发烫或有异味），如果发现应该立即关闭电源，保护现场，报告直到老师，只有在查清原因，排除故障后，才能继续做实验。在实验报告中认真记录实验数据、波形并加以分析，发生故障后应独立思考，耐心排除，同时记下故障排除的方法和过程。认真地分析故障原因。并说明故障排除的过程和方法。
3. EDA 实验按照预习时设计方案进行电路的仿真调试和下载物理验证，发现故障时首先根据电路原理分析故障排除，如果还是无法排除向老师或助教描述故障，说明已经采用过的排查过程，在老师或助教的指导下自行排除。
4. 实验完成后，提交老师或助教演示实验结果，经老师或助教同意后方可结束实验。

如果在实验课没有完成实验任务，经老师或助教同意在课余时间预约实验室完成。

5. 实验结束后，经教师同意方可拆除线路，先关断仪器电源，然后再拆线，并将仪器设备恢复原状，整理好实验桌及。

B.2 正确的撰写实验报告

要想圆满的完成一项实验，必须把握三个环节，即实验前的预习，实验中正确的操作及实验后撰写一份完整的报告。写好实验报告十分重要，他不仅仅是完成实验一个不可缺少的部分，还是今后写好技术报告和科技论文的基础。因此，每一位同学都必须认真对待。

B.2.1 实验报告的内容

一份完整的实验报告应该包含实验标题、实验目的、实验原理、实验仪器和设备、实验方法和步骤、数据记录和分析、问题讨论这几部分构成。

实验标题：包含实验题目，实验者的姓名，实验合作者的姓名，实验日期，提交报告的日期等内容。

实验目的：实验目的叙述要简洁明了恰如其分。切忌用过于笼统的语言来描述它，更不要把它扩大到超出实验中实现的范围，例如写“研究 LC 振荡器的特性”就不如写“研究 LC 振荡器的起振条件”更贴切。

实验内容和原理：简要说明实验内容，实验原理主要简略描述该实验的主要原理，其中应包括原理图、主要公式等。如果是设计型实验需要写出详细的设计方案和实验的技术过程(路线)。

实验仪器和设备：列出实验中所使用的仪器仪表设备和主要元器件的型号、规格，生产制造厂家也是极其必要的。因为别人能据此对你的实验结果的可靠性和精确程度做出初步的判断，也可供想重做该实验并得到相同的结果的人参考。

实验方法和步骤：实验方法和步骤要用实验者本人的语言简明扼要的描述，而不是照抄实验指导书上的内容。

实验结果：实验结果包括实验中仿真结果(含仿真激励和条件)、实际测得的数据（包括波形）和根据测得的数据及计算得到的数据两种。这里需要注意以下几个问题：

1. 测得的数据要根据误差的要求，正确选取其有效数字的位数，而计算所得的数据的精

度（有效数字的位数）要与测得数据的精度相吻合，若存在差异，要分析原因。

2. 检测得到的数据由表格表示时，每张表格都应加上说明标题，表中所列的数据应附有单位，同一类型的量采用的单位应力求相同。计算所得的数据也可列在同一表格中对比，但要清楚地加以说明它是计算所的数据。
3. 有时需要将测得的数据绘制成图形或曲线。那么，每条曲线的水平和垂直坐标轴都应标明所代表的量、采用的单位及刻度大小。每张图上的数据点应使用记号清晰表示，每根曲线都应画的光滑，连续(除离散量)，而不是机械地用每个数据点连成的多线段折线或多段弧线来表示。有时，如需将实验结果与理论曲线进行比较，则理论曲线也应画在同一张图上，所使用的单位和刻度也应相同。
4. 任何一次实验的结果都存在误差。认真地对实验所得的结果进行误差分析，不仅仅是提高实验质量的关键，还能提高对实验过程的洞察力以及培养分析问题和解决问题的能力。

分析实验误差应包括：误差的计算，误差变化规律的探寻，误差产生原因的分析以及对误差影响大小不同的因素的区分等。如果本次实验是验证某项理论，那么就应把实测数据与理论计算结果进行对比分析，说明产生误差的原应以及误差的大小。

B.2.2 思考题及讨论

思考题及讨论包括两个部分：实验教材中思考题的讨论；实验结果或实验过程中奇异现象和结果的讨论。

如果实验数据揭示出一种未曾预料到的结果，那么，应在讨论部分加以陈述，并力争对此种新现象做出初步的解释与判断。最后提出对该项实验改进建议，如实验方案的修正，内容的增删，步骤的改进，精度的提高等。

除了以上所阐明的那些内容要求外，一份优秀的实验报告，还应做到文理通顺，字迹工整，图形美观，页面整洁。

B.2.3 实验报告模板

实验×—××实验报告

姓名：_____ 学号：_____ 专业：_____

课程名称：_____ 数字逻辑设计 _____ 同组学生姓名：_____

实验时间：_____ yyyy-mm-dd _____ 实验地点：紫金港东 4-509 _____ 指导老师：_____

一、实验目的和要求

简要介绍本次实验的目的和要求。

二、实验内容和原理

本小节详细说明实验内容和实验原理，必要时应有图片、表格等。如果内容比较多，可以分节描述，小节的格式如下：

2.1 实验任务

<正文>

2.2 实验原理

<正文>

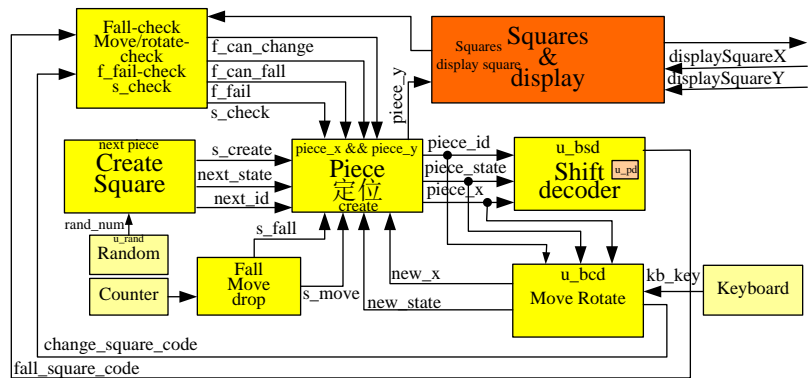
2.3 XX（其他相关内容）

<正文>

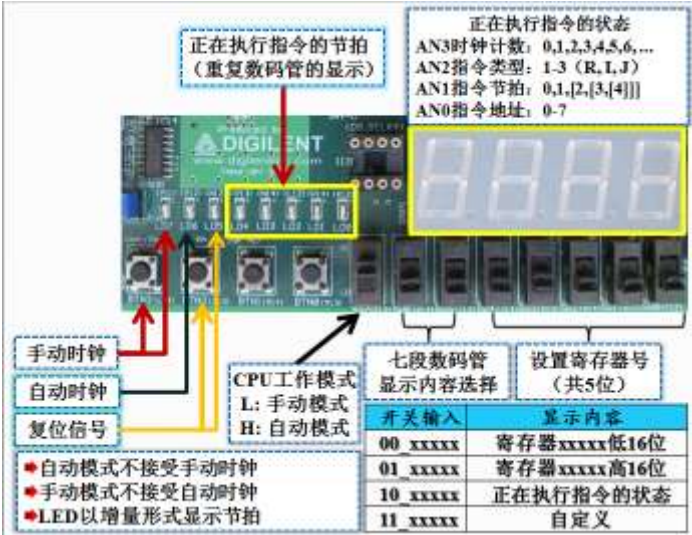
图片和表格的格式要求示范如下：

!!! 非实物图片用 Microsoft Visio 工具画；实物图片上的说明也用 Microsoft Visio 工具标注！

B.2.3 实验报告模板



图表 1 XX 系统原理框图（图的名称）



图表 2 XX 系统操作接口说明（图的名称）

图表 1 表格的名称

标题	栏目 1	栏目 2	栏目 3	栏目 4

插入题注的方法: office 20xx, “插入”菜单, 引用, 然后找需要费用的内容。office 20xx,

“引用”工具栏选项卡，在“题注”中选择“插入题注”。

要求：按照默认的格式，在报告中不区分图、表的标签，即“图 1”和“表 1”不作区分，一律简单设置为“图表 1”。所不同的是，对于图，题注在图的下方且位置居中；对于表，题注在表格的上方，且左对齐。

截图请注意把重点信息体现出来，图片中不要把无关的部分如任务栏等包括进来，注意可读性和图片清晰度。

表格的样式根据实际需要选择；表格最好不要跨页。

本节中，内容和原理不要原样复制课件的内容。

三、主要仪器设备

必须采用编号样式，设备的数量和单位应对齐。示范如下：

1. xx 开发板 1 套
2. xx 配置 PC 机 1 台
3. ISE12.4 ...
4.

四、实验实现方法、步骤与调试

本节重点介绍实验的具体过程，包括：实现思路（技术路线）、代码设计层次结构图及说明、源代码（包括注释）、PC 机上进行的关键步骤截图及说明、调试过程等，这部分的内容应当与实际操作过程和结果相符。

同时说明实现过程中的关键点和难点

本节也可以再细分小节，要求同上。

代码的格式要求：源代码应对齐（采用 tab 键、空格）。代码字体选择 courier/courier new，字号：10，行距：最小值、0。不需要源代码着色和行号。建议采用表格，1 行 1 列，表格边框设为无，并采用浅色底纹。

正文中需要对核心代码或重要代码作分析示例，不需要使出所有代码。完整代码放入附件中。特别注意：**源代码禁止采用图片格式（贴图）！**

代码示范如下：

```
`timescale 1 ps / 1 ps

module glbl ();
```

B.2.3 实验报告模板

```

parameter ROC_WIDTH = 100000;
parameter TOC_WIDTH = 0;
wire GSR;
wire GTS;

.....
//----- JTAG Globals -----
wire JTAG_TDO_GLBL;
wire JTAG_TCK_GLBL;
wire JTAG_TDI_GLBL;
wire JTAG_TMS_GLBL;
wire JTAG_TRST_GLBL;

.....

assign (weak1, weak0) GSR = GSR_int;
assign (weak1, weak0) GTS = GTS_int;
assign (weak1, weak0) PRLD = PRLD_int;

initial begin
GSR_int = 1'b1;
PRLD_int = 1'b1;
#(ROC_WIDTH)
GSR_int = 1'b0;
PRLD_int = 1'b0;
end

.....

endmodule

```

仿真结果示例:

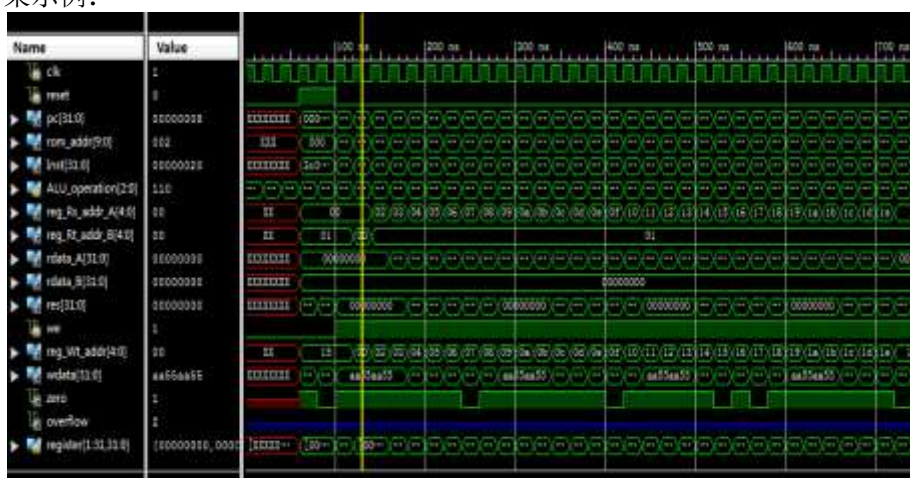


图 3 XXXX 仿真时序

五、实验结果与分析

这里应给出详实的实验结果。

分析应有条理，要求采用规范的书面语。

六、讨论、心得

简要地叙述一下实验过程中的感受，以及其他的问题描述和自己的感想。

报告的第二、四、五部分是重点，请认真书写。

B.2.4 课程设计报告模板

1. 封面

浙江大学

课程设计报告

中文题目： 基于数字系统的俄罗斯方块

英文题目： _____

姓名/学号： _____

指导教师： _____

参加成员： _____

专业类别： _____

所在学院： _____

论文提交日期____年__月__日

2. 摘要

摘要

3. 关键词

关键词：

4. 目录

目录

.....

5. 图目录

单独一页。

图目录

.....

6. 报告实体

由 5~6 部分组成，具体内容如后。

第1章 绪论

1.1 ……设计背景

（内容要点：选择这个设计的思想背景和意义，有什么特点，希望达到什么目的等）

!!! 非实物图片用 Microsoft Visio 工具画；实物图片上的说明也用 Microsoft Visio 工具标注！
!!! 不要原样复制课件的内容。
!!! “……” 要用课程设计相关内容代换

插入题注的方法：office 20xx，“插入”菜单，引用，然后自己找找。office20xx，“引用”工具栏选项卡，在“题注”中选择“插入题注”。其他编辑器如 Microsoft office 请自行定义，格式参考如下示例。

要求：按照默认的格式，在报告中区分图、表的标签，即“图 1”或“表 1”，对于图，题注在图的下方且位置居中；对于表，题注在表格的上方，且左对齐。

截图请注意把重点信息体现出来，图片中不要把无关的部分如任务栏等包括进来，注意可读性和图片清晰度。

表格的样式根据实际需要选择；表格最好不要跨页。

图片和表格的格式要求示例如下：

错误!不能通过编辑域代码创建对象。

图 1 XX 系统原理框图（图的名称）

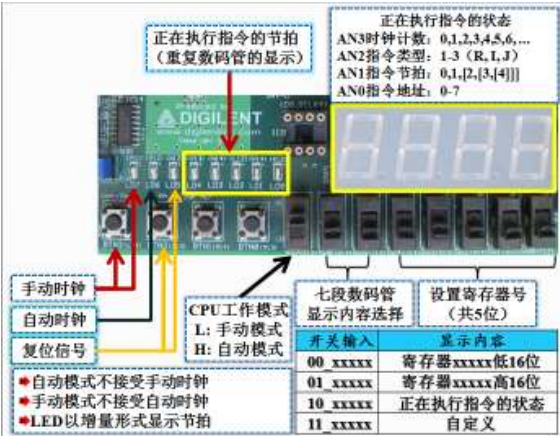


图 2 XX 系统操作接口说明（图的名称）

表 1 表格的名称

标题	栏目 1	栏目 2	栏目 3	栏目 4

1.2 国内外现状调查

（内容要点：查阅文献和资料，国内外高校是否有相同和类似的设计，现状、特点和结果与本设计的异同等。鼓励增加国内外“应用与研究”的现状分析）。

1.3 主要内容和难点

（内容要点：本设计要完成的主要内容（任务）、功能、技术要求和目的，以及实现的重点或难点；）

第2章 ……设计原理

2.1 ……设计相关内容

（内容要点：课程设计用到的理论要点、技术工具和方法。如果有课程外理论或技术需要另分节展开简述）

2.2 ……设计方案

（内容要点：课程设计的技术需求、整体设计方案(含结构框图及分析说明)、技术路线(实现上的技术思路)）

2.3 ……硬件设计

（内容要点：详细的硬件设计分析过程，包括顶层及各电路模块结构、逻辑电路图、硬件描述代码等）

2.4 系统软件设计（若有）

（内容要点：详细的软件设计分析过程，包括测试代码，主程序和各功能子模块代码、软件框图和流程图等）

第3章 ……设计实现

3.1 实现方法

（内容要点：实现的具体思路和方法、重点难点在实现中的经验和方法等）

3.2 实现过程

（内容要点：详细的实现过程，包括步骤，模块层次结构，信号定义、综合后 RTL 逻辑图，那些模块需要做仿真等）

3.3 仿真与调试

（内容要点：详细的仿真与调试过程和内容，包括出现的问题与解决方案，仿真结果图等）

第4章 系统测试验证与结果分析

4.1 功能测试

（内容要点：根据系统设计的功能设计测试方法，验证是否达到设计功能目标及功能正确性和完备性等）

4.2 技术参数测试

（内容要点：根据系统设计的参数要求设计测试方法，验证是否达到设计技术目标及技术参数等）

4.3 结果分析

（内容要点：分析验证结果、存在的问题及原因、最终的成果内容等）

4.4 系统演示与操作说明

（内容要点：分析验证结果、存在的问题及原因、最终的成果内容；系统操作说明(使用说明)，演示的主要结果截图（要有说明）等）

Final PROJECT 需另附 3~5 分钟视频：

1. 自我介绍（含成员, 10 秒）；
2. 简要阐述设计方案和设计实现过程
3. 操作演示

第5章 结论与展望

（内容要点：简要讨论并叙述 Project 过程中的感受，以及其他的问题和自己的感想。）

6.1 前面已经记录了一些组成 Verilog 的基本组成，可以用这些基本组成来构成表达式。这一节，就来记录一下把这些表达式构成一个文件的各种行为描述语句。①这里用 Verilog 基本要素进行的行为描述主要是针对综合来的，也就是可以设计出实际电路来的（行为描述语句有两大子集，一个是面向综合，一个是面向仿真）。②行为描述语句一般指放在 always 语句中。内容提纲如下所示：

- 触发事件控制
- 条件语句（if 与 case 语句）
- 循环语句
- 任务和函数
- 编译预处理

一、触发事件控制

①电平敏感事件是指 指定信号的电平发生变化时发生指定的行为。

②边沿触发事件（信号跳变沿）是指 指定信号的边沿信号跳变时发生指定的行为，分为信号的上升沿（ $x \rightarrow 1$ 或者 $z \rightarrow 1$ 或者 $0 \rightarrow 1$ ）和下降沿 $x \rightarrow 0$ 或者 $z \rightarrow 0$ 或者 $1 \rightarrow 0$ ）。

③信号跳变沿触发电路对信号的某一跳变沿敏感名字一个时钟周期内，只有一个上升沿和一个下降沿，因此计算结果在一个周期内保持不变，而电平触发电路则可能会引起数据在一个时钟周期内变化一次或多次。

其他敏感列表的事项请查看这篇博文：<http://www.cnblogs.com/IClearner/p/7253002.html>。

二、条件语句

Verilog 的条件语句包括 if 语句和 case 语句。

（1）if 语句

①if 语句中的条件判断表达式（括号中的那个）一般为逻辑表达式或者关系表达式或者就一个变量。如果表达式的值是 0、X 或者 Z，则全部按照“假”处理；若为 1，则按照“真”处理。

②在应用中，else if 分支的语句数目由实际情况决定；else 分支可以省略，但在描述组合逻辑中，会综合得到锁存器。

（2）case 语句

①case 语句，case 语句是一个多路条件分支的形式，常用于多路译码、状态机以及微处理器的指令译码等场合，有 case 分支、casez 分支、casex 分支这三种形式。

②case 语句首先对**条件表达式**求值，然后同时并行对各分支项求值并进行比较；当 case 语句跳转到某一支后，**控制指针**将转移到 endcase。

③case 分支，case 分支语句在执行时，条件表达式和分支之间进行的比较是一种**按位**进行的全等比较，也就是说，只有在分支项表达式和条件表达式的每一位都彼此相等的情况下，才会认为二者是相等的；此外 x、z 这两种逻辑状态也作为合法的状态参与比较（ $1 == 1$ ， $0 == 0$ ， $x == x$ ， $z == z$ ）。

④当分支的结果以常数出现时，如果没有指定位宽，则 Verilog 编译器默认其具有与 PC 字长相等的位宽。

⑤在 casez 分支中，如果分支取值的某些位为高阻 z，则这些位的比较就不予以考虑，只关注其他位的比较结果。

⑥在 casex 分支中，则 x、z 都不予以考虑。

（3）if 与 case 的区别

①if 语句指定了一个有优先级的编码逻辑，而 case 语句生成的逻辑是并行的，不具有优先级。

②通常 if 结构得出的电路速度较慢，但是占用面积较小，由于速度慢，因此不适合构建特别长的 if 语句。

③case 结构较 if 结构的速度快，但是占用面积大。

三、循环语句

循环语句有四种：for 循环、repeat 循环、while 循环、forever 循环。但是 forever 循环不能进行综合，而其他三种在一定情况下可以进行综合，因此这里记录可以综合的这三种。一般在电路设计中，不是经常用到循环语句，因为循环语句不好进行优化，占用的资源多，即使用到也是用到 for 循环。

(1) for 循环

①for 循环中，在循环次数确定的情况下，也就是循环结束条件是个常量下，才可以进行综合。主体注意要用 begin...end 来进行包括。

②在 Verilog 中，**不支持++，--这些运算**，需要用完整的 i=i+1 来完成。

(2) repeat 循环

①repeat 语句执行指定的循环次数，如果循环次数是 x 或者 z，那么循环次数将会按照 0 次处理。主体注意要用 begin...end 来进行包括。

②当循环次数在程序编译过程中保持不变时，可以进行综合。

(3) while 循环

①只有在指定的循环条件为真（0、x、z 都是假）时才会重复执行循环体，注意循环体要用 begin...end 来包括。

②在执行语句中，必须有改变循环执行条件表达式的值的语句，否则循环可能进入死循环。

循环语句使用注意事项：

①循环语句中出现的变量都采用阻塞赋值（时序中，即沿触发中），这是因为在 always 块中，使用非阻塞赋值时，只有在 always 结束后才会把右端的值赋给左边的寄存器，如果采用非阻塞赋值，则会造成循环语句只执行一次。

②在沿触发中，语句采用阻塞赋值，这是由于循环语句本质是由组合逻辑实现的，虽然整体是基于时序逻辑，但是循环部分确是组合逻辑。

③基于循环语句的 Verilog 显得相对精简，但是面向硬件设计的关注点是时序、面积、功耗等，在使用循环语句进行电路设计时要慎重。

四、任务和函数

(1) 任务语句（task）

任务(task)就是一段封装在“task-enttask”之间的程序。任务是通过调用来执行的，而且只有在调用时才执行，如果定义了任务，但是在整个过程中都没有调用它，则该任务不会被执行的。调用某个任务时可能需要它处理某些数据并返回操作结果，所以 Verilog 中的 task 存在输入端和输出端。

任务定义语法格式如下所示：

```
1 task<任务名>;                                // <= task task_id;
2 <端口及数据类型声明语句>                    // <= [declaration]
3 <语句 1>                                     // <= procedural_statement
4 <语句 2>                                     // <= procedural_statement
5 .....                                       // <=
6 <语句 n>                                     // <= procedural_statement
7 endtask                                     // <= endtask
```

上升的格式中，关键词 `task` 和 `endtask` 将它们之间的内容标志成一个任务定义，`task` 标志着一个任务定义结构的开始；`task_id` 是任务名；可选项 `declaration` 是端口声明语句和变量声明语句，任务接收输入值和返回输出值就是通过此处声明的端口进行的；`procedural_statement` 是一段用来完成这个任务操作的过程语句，如果过程语句多于一条，应将其放在语句块内；`endtask` 为任务定义结构体结束标志。一个任务的例子如下所示：



```
1 task find_max; //任务定义结构开头，命名为 find_max
2     input [15:0] x,y; //输入端口说明
3     output [15:0] tmp; //输出端口说明
4 if(x>y) //给出任务定义的描述语句
5     tmp = x;
6 else
7     tmp = y;
8 endtask
```



编写任务时，需要注意：

①调用某个任务时，可能需要它处理某些数据并返回操作结果，因此任务应当有接收数据的输入端和返回数据的输出端。

②任务的输入、输出和双向端口的数量不受限制，甚至可以没有输入、输出和双向端口。

③在任务定义的描述语句中，可以出现不可综合的语句，但是这样会引起任务不可综合。

④任务的**定义结构**定义不能在 `initial` 或者 `always` 语句中，在任务**定义结构内**不能出现 `initial` 和 `always` 过程块；但是任务的**调用**必须在这两个语句块中。

⑤任务定义中可以出现“`disable`”终止语句，但是这是不可综合的；在仿真中，如果出现的该终止语句，将中断正在执行的任务；任务被中断后，程序流程将返回到调用任务的地方继续向下执行。

⑥在第一行“`task`”语句中不能列出端口名称。

⑦可以使用出现不可综合操作符合语句（使用最为频繁的就是延迟控制语句，例如`#10ns`），但这样会造成该任务不可综合。

任务的使用（调用）：

`task_id` (端口 1,端口 2,...,端口 n);

`task_id` 是要调用的任务名，端口 1、端口 2，…是参数列表。参数列表给出传入任务的数据（进入任务的输入端）和接收返回结果的变量（从任务的输出端接收返回结果）。

任务调用的注意事项：

①任务调用中接收返回数据的变量必须是寄存器类型。

②任务调用语句和一条普通的行为描述语句的处理方法一致。

③可综合任务只能实现组合逻辑；也就是说调用可综合任务的时间为 0。而在面向仿真的任务中可以带有时序控制，如多个时钟周期或时延，因此面向仿真的任务调用时间不为 0。

④在任务中可以调用其他任务或者函数，也可以调用自身。

⑤当调用输入、输出和双向端口时，任务调用语句必须包含端口名列表，且信号端口顺序和类型必须和任务定义结构中的顺序和类型一致。任务的输出端口必须和寄存器类型的数据变量相对应。

（2）函数（function）

函数与任务类似，函数（`function`）是一段封装在“`function-endfunction`”之间的程序。函数的定义格式如下所示：



```

1 function<返回值的类型或范围>(函数名);
2     <端口说明语句>// input XXX;
3     <变量类型说明语句>// reg YYY;
4 begin
5     <语句>
6     .....
7     函数名= ZZZ; //函数名就相当于输出变量;
8 end
9 endfunction

```



函数编写的注意事项如下：

①定义函数时至少要有一个输入参数；可以按照 ANSI 和 module 形式直接定义输入端口；例如：

```
function[63:0] alu (input[63:0] a, b,input[7:0] opcode );
```

此外，函数至少有一个输入端口，可以有多个输入端口；不允许输出声明，也不允许双向端口，因为函数名本身充当一个返回值。

②和任务一样，函数的定义只能在模块中完成，不能出现在过程块中。

③函数结果中，不能出现任何形式的时间控制语句（如#，wait），也不能使用 disable。

④函数内部可以调用函数，但是不能调用任务。

⑤如果描述语句是可综合的，则必须所有分支均赋值，不允许存在不赋值情况，只能按照组合逻辑方式描述。

函数编写的一个例子如下所示，这是一个计算有符号数绝对值的函数：

```

1 function[width-1 : 0] DWF_absval;
2     input [width-1 : 0] A;
3 begin
4     DWF_absval = ((^(A ^ A) != 1'b0)) ? {width{1'bx}} : (A[width-1] == 1'b0) ? A : (-A);
5 end
6 endfunction

```

关于函数调用的注意事项：

①函数调用可以在过程块中完成，也可以在 assign 这样的连续赋值语句中出现。

②函数调用语句不能单独作为一条语句出现，只能作为赋值语句的右端操作数。

使用函数的一个完整实例：一个乘法电路——输入操作数 a 和 b，输出他们的乘积；这里使用移位相加的算法来实现乘法。使用函数的实现的代码如下：



```

1 module MAC #(parameter N=8)(
2     input clk,
3     input reset,
4     input [N-1:0]      opa,
5     input [N-1:0]      opb,
6     output reg[2*N-1:0] out
7 );
8
2

```

```

9 function[2*N-1:0] mult;//函数定义，mult 函数完成乘法操作
10     input[N-1:0] opa;    //函数只能定义输入端
11     input[N-1:0] opb;    //输出端口为函数名本身
12     reg[2*N-1:0] result;
13     integer i;
14 begin
15     result = opa[0]? opb : 0;
16     for(i= 1; i <= N-1; i = i+1)
17     begin
18         if(opa[i]==1) result=result+(opb<<i);
19     end
20     mult=result;
21 end
22 endfunction
23
24 wire[2*N-1:0] sum;
25 assign sum = mult(opa,opb) + out;
26 always@(posedge clk or negedge reset)
27 if(!reset)
28     out<=0;
29 else
30     out<=sum;
31
32 endmodule

```



(3) 关于任务和函数的对比和总结

①task 语句和 function 语句都是可以综合的，但是只能实现组合逻辑，具备组合逻辑的优点和缺点。

②task 和 function 都必须在模块内部定义，除参数个数不同外，还可以定义内部变量，包括寄存器、时间变量、整型等，但是不能定义线网变量。二者只能出现在行为描述中。

③区别：

比较点	任务	函数
输入、输出	可以有任意多个种类类型的参数	至少有一个输入，不能有输入和双向端口
调用	只能在过程语句中调用	可以在过程语句中调用，也可以作为赋值操作的表达

		式用在 assign 赋值语句中
触发事件控制	任务可以包含延迟控制语句等，但是只能面向仿真，不可综合。	不能有时间控制语句
调用其他函数和任务	可以调用其他函数和任务	只能调用函数
返回值	任务没有返回值	函数向调用它的表达式返回一个值
其他	任务调用语句可以作为一条完整的语句出现	函数语句只能作为赋值操作的表达式，不能作为一条独立的语句出现

五、编译预处理语句

①编译预处理是 Verilog 编译系统的一个组成部分，指编译系统会对一些特殊命令进行预处理，然后将预处理结果和源程序一起在进行通常的编译处理。编译预处理是可以综合的。

②在 Verilog 语言编译时，特定的编译器指令在整个编译过程中有效（编译过程可跨越多个文件），直到遇到其他的不同的编译程序指令。

③常用的编译预处理语句有：``define`、``undef` ； ``ifdef`、``else`、``endif` ； ``include` ； ``timescale`。

（1）``define`、``undef`

格式：``define` 宏的名称 宏的正文

①宏定义的名称可以是大写，也可以是小写，但是主要不要和变量名重复，此外宏定义的使用要注意带上```。

②和所有的编译器伪指令一样，宏定义在超过单个文件边界时仍然有效（对工程中的其他源文件），除非被后面的``define` 或者``undef` 伪指令覆盖，否则``define` 不受范围的限制。

③当用变量定义宏时，变量可以在宏正文中使用，并且在使用宏时可以用实际的变量表达式代替。

④通过用反斜杠\转移中间的换行符，宏定义可以跨越几行，新的行是宏正文的一部分。

⑤宏定义的行末不需要添加分号或者逗号表示结束

⑥宏正文不能分离的语言记号包括注释、数字、字符串、保留的关键字、运算符。

⑦编译器伪指令不允许作为宏的名字，此外宏的正文可以是一个表达式，并不仅英语变量名的替换。

⑧``define` 和 `parameter` 的区别:

区别点	<code>`define</code>	<code>parameter</code>
作用域	① <code>`define</code> 从编译器读到这条指令开始到编译结束	① <code>parameter</code> 作用于声明的当前文件，如果要让它

	都有效（除非遇到上面的提到的情况），可以应用于整个工程。 ②`define 可以写在代码的任何位置。	作用与整个项目，可以将这些声明单独列在一个文件中，然后用`include 进行包含。 ②parameter 必须放在应用之前，也就是你要用到某个 parameter 参数了，你必须先 parameter 它。
传递功能	`define 不能实现参数传递，但是不局限与定义变量，还可以定义	Parameter 可以用作模块例化时的参数传递，实现参数化调用，但是仅局限于定义变量，而不能定义表达式。

（2）条件编译命令`if 语句

①条件编译指令包括`ifdef 、`else 、`endif，其中`ifdef 、`endif 必不可少，`else 可选，且条件编译语句可以放在程序的任何地方调用。

②举例：

```
1 `ifdef ABCD
2     Parameter X1 = 1;
3 `else
4     Parameter X2 = 1;
5 `endif
```

意思是：如果用`define 定义的 ABCD，那么就会执行第一个模块（Parameter X1 = 1）；否则执行第二个模块（Parameter X2 = 1）。

（3）文件包含`include 语句

①当某个模块需要调用某一个文件时，但是这个文件不在当前目录下，那么就需要使用`include 语句进行包含，这样调用才不会出现错误。如 `include “.././xxxx.v”

②一个`include 指令只能指定一个被包含的文件。如果要完成 N 个文件的包含，需要 N 个`include 指令。

③可以将多个`include 指令写在同一行，在`include 命令行只能出现空格和注释。

④如果文件 A 同时包含了文件 B 和 C，那么文件 C 可以直接利用 B 的内容，而不需要在对 B 文件进行包括，同理，B 也可以直接利用 C 的内容。

(4) 时间尺度`timescale 语句

①`timescale 用于定义延时的单位和延时的精度，如`timescale 1ns/100ps 那么时间单位就是 1ns,精度就是 100ps。

②时间单位，表示了仿真时测量的单位，比如延时 1，1ns;精度则表示仿真器只识别的范围，比如精度是 100ps，那么如果你 1.3ns,编译器是识别，但是如果写 1.32，那么由于精度达不到那么细，所以 0.02 被四舍五入掉。

③`timescale 影响着全部模块，知道遇到另外的`timescale。