

---

# 实验一 Vivado使用与组合电路的Verilog实现

## 实验目的

- (1) 熟悉 Vivado 的开发环境及开发流程;
- (2) 熟悉 Xilinx FPGA开发环境、EGO-1实验板的功能和使用方法;
- (3) 了解Verilog的结构描述、数据流描述和行为描述法;
- (4) 掌握组合电路的Verilog设计与实现

## 1 实验题目一：拨码开关控制led

### 1.1 实验内容

运用 Verilog HDL 语言, 在 Vivado 中实现用 8 位拨码开关控制 8 位LED 灯的电路, 并下载到 EGO-1开发板, 验证结果。

**注意:** EGO主芯片为 XC7A35T-CSGD324-1, 需要 64位的 Vivado (2015.4 及以后的版本)。本指导书中的配图以Vivado 2018.3 Webpack版展示Vivado环境下的基本操作。

### 1.2 实验步骤

#### (1) Vivado介绍与安装

Vivado是xilinx公司于2012年推出的开发套件EDA工具, 集成文本编辑器、逻辑函数库、布线/仿真工具、下载器等功能。早期的xilinx fpga开发软件是ISE, 后因xilinx FPGA架构和开发方法的变化而推出全新的vivado不再更新ISE。



Vivado有免费版的webpack版(lab版), 支持的芯片有限, 支持windows、Linux操作系统, 教学实验用webpack已经足够。Vivado每年都会更新, 对于数逻实验, 2018.3、2019.1、2019.2三个版本都可以, 建议安装较新版本。

Vivado分本地和在线安装两种方式, 最终安装好之后会占用20G左右的空间。本地安装需要先把完整的安装包下载到本地。在线安装是下载一个小的引导包web installer, 运行引导包边下载边安装, 对于网络要求高, 所以建议离线安装。Vivado安装包可以从官网<https://china.xilinx.com/support/download.html>下载, 但需要注册账号, 这里提供一个网盘下载的链接<https://pan.baidu.com/s/1UQBFCttvb9vg1KCOo1G3ug#list/path=%2F> 密码: hy96。实验室电脑安装的是2018.3的版本, 对于想在自己笔记本上安装的可下载安装包按照安装指引自行安装。

目前FPGA的两大主要厂商为xilinx和altera (已被Intel收购)。Xilinx市场份额接近50%, 主要产品包括: Sparten系列、Virtex系列、Artix系列、Kintex系列、Virtex系列等。

其第六代及以前的产品的开发工具为ISE，从第七代产品开始，已全部转移到vivado平台。[Altera](#)市场份额40%以上，主要产品包含：Max系列、Cyclone系列、Arria系列、Stratix系列等。开发工具为Quartus。此外，Lattice、Actel、Atmel等公司也有FPGA产品，由于市场份额小，市面上很少见到。

## (2) 创建新项目

在桌面双击  打开 Vivado，然后点击Create Project  创建一个新项目（或者在菜单栏选择 File->Project->New），图 1-1 所示。与Vscode等IDE使用方式类似，都需要新建项目，然后添加文件，通过项目管理一系列文件，且都是在File菜单栏下操作。

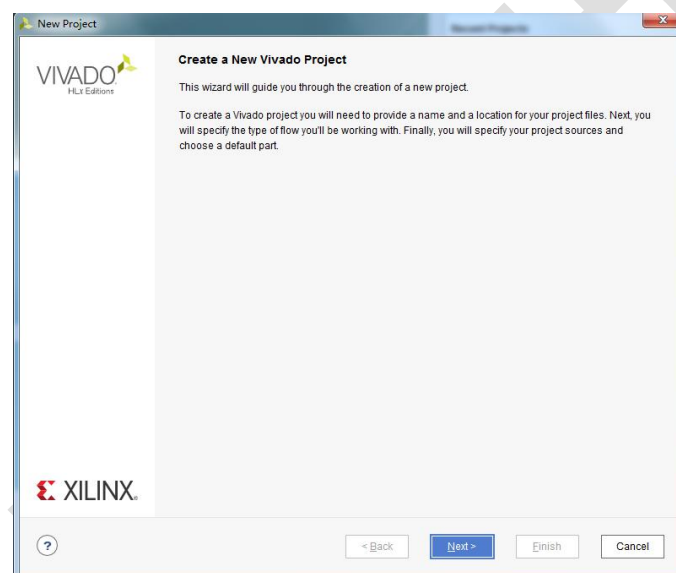


图 1-1 New Project

点击 **Next**，显示如图 1-2 的界面用于命名项目名称和路径。**Project Name**是项目名称，建议以字母、数字和下划线的组合命名，大小写不限，工程名称和存储路径中不能出现中文和空格。**Project location**是项目保存的路径，不要使得工程所在目录太深。Vivado会自动在该目录下生成多个子目录和配置文件。根据自己的盘符和使用习惯输入对应的名字。建议按图1-2所示命名，方便后面的操作。

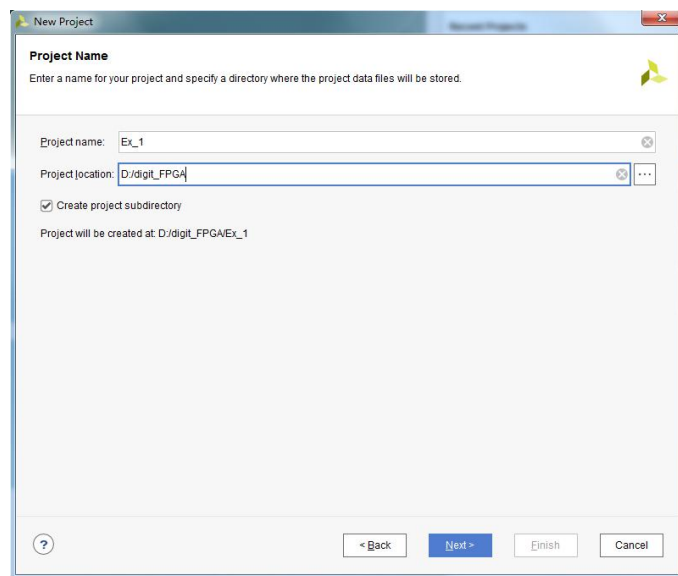


图 1-2 项目名称

这里项目名称为 Ex\_1，项目的位置是 D:/digit\_FPGA，点击**Next**。Vivado会自动在最后D:/digit\_FPGA目录下生成Ex\_1文件夹，整个项目将保存在D:/digit\_FPGA/Ex\_1目录。

如图1-3 所示选择项目类型为“RTL Project”。这里有多个类型可选，在后续课程中，我们都只用到RTL Project类型，其他类型不做了解。注意下面的勾选框，勾选“Do not specify sources at this time”，即在新建项目的过程不需要增加源文件（Verilog代码文件），而是在项目建好之后才添加，点击 **Next**进行下一步。

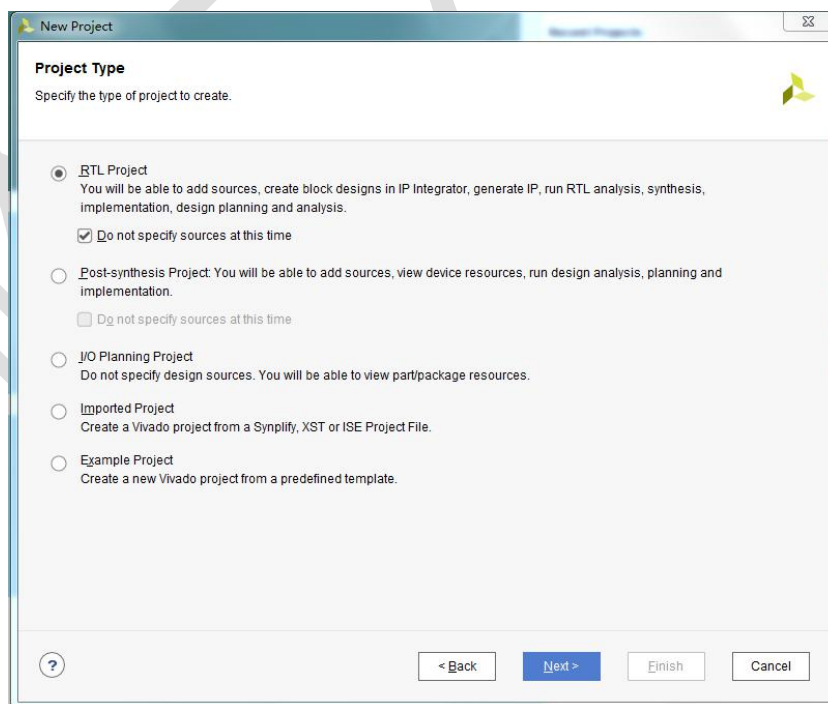


图 1-3 项目类型

接下来是选择项目运行的芯片，不同芯片LUT资源、引脚数和封装都不一样 ([xilinx芯](#)

片家族)。同一个项目，选择不同的芯片，综合和实现的结果都会不同。按图 1-4 改变下拉列表中的选项，可以直接在search框中搜索芯片型号，也可以先根据Family进行大类筛选，再根据Package、Speed进行细选。Ego开发板上的芯片是Artix-7 xc7a35tcsg324-1 (family是Artix-7, package是csg324, speed是-1)，直接搜索型号即可，然后点击 Next。当然项目建好后也可以在项目的Settings中随时修改所用的芯片型号。

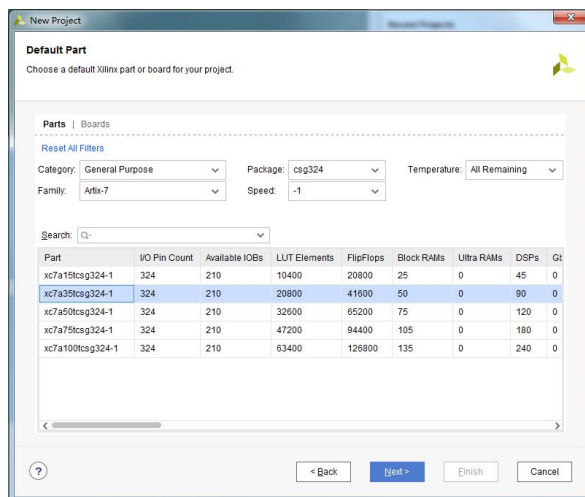


图 1-4 选择器件

可以看到如图 1-5 所示的新项目概览，点击 **Finish**完成工程创建。

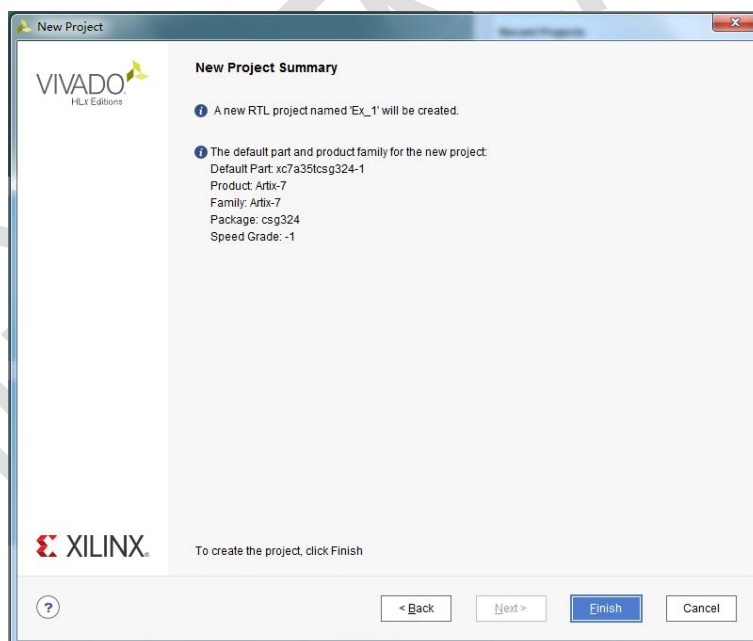


图 1-5 新项目概览

### (3) 添加源代码

工程创建之后，默认打开的界面如图1-6所示，界面比起软件开发的IDE如vscode要复杂一些。

对于刚接触vivado的同学对vivado的界面布局和功能难以理解，可以在多次使用之后再

回过头来看这部分的介绍，当前我们只要关注5个部分。第一部分是最上边的菜单栏，是所有IDE都会有的。第二部分是左侧的flow navigator即流程向导，在vivado编写代码到最终的上板的所有流程都集成在这个区域。第三部分是中间的sources小窗口，即源文件管理，包括添加、删除、编辑等操作。第四部分是右边的workspace工作区，这块区域显示内容是会变的，与当前所执行的操作有关，如果是编辑文件则是编辑窗口，如果是仿真则是波形显示窗口。第五部分是最下面的结果输出窗口，运行日志、错误日志等运行结果都会在这里显示。

先添加设计源代码，在图 1-6 所示的Source窗口中右键点击 Design Sources，在弹出的菜单中选择 Add Sources...，出现图1-7 所示的对话框。

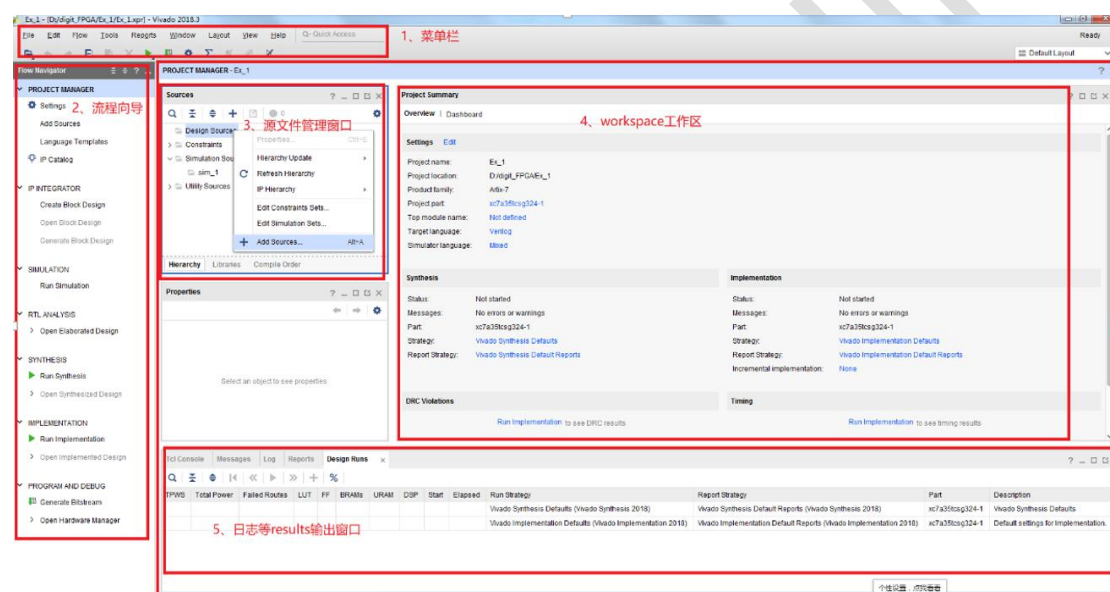


图 1-6 创建新项目后的界面

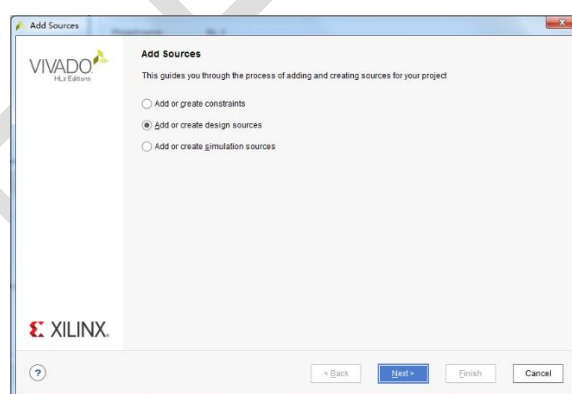


图 1-7 添加源程序对话框

图1-7里面有三个选项，分别对应约束、设计、仿真文件，约束和仿真后面会介绍，这里选择第二个即添加设计文件（功能实现文件），选择后点击Next。Verilog代码都是以“.v”为后缀名的文件，可以在其他文件编辑器里写好，再添加到新建的工程中，也可以在工程中

新建一个再编辑。如果已经有Verilog文件，则是add, 没有则是create。在接下来打开的对话框中直接点击中间的“Create File”新建文件。

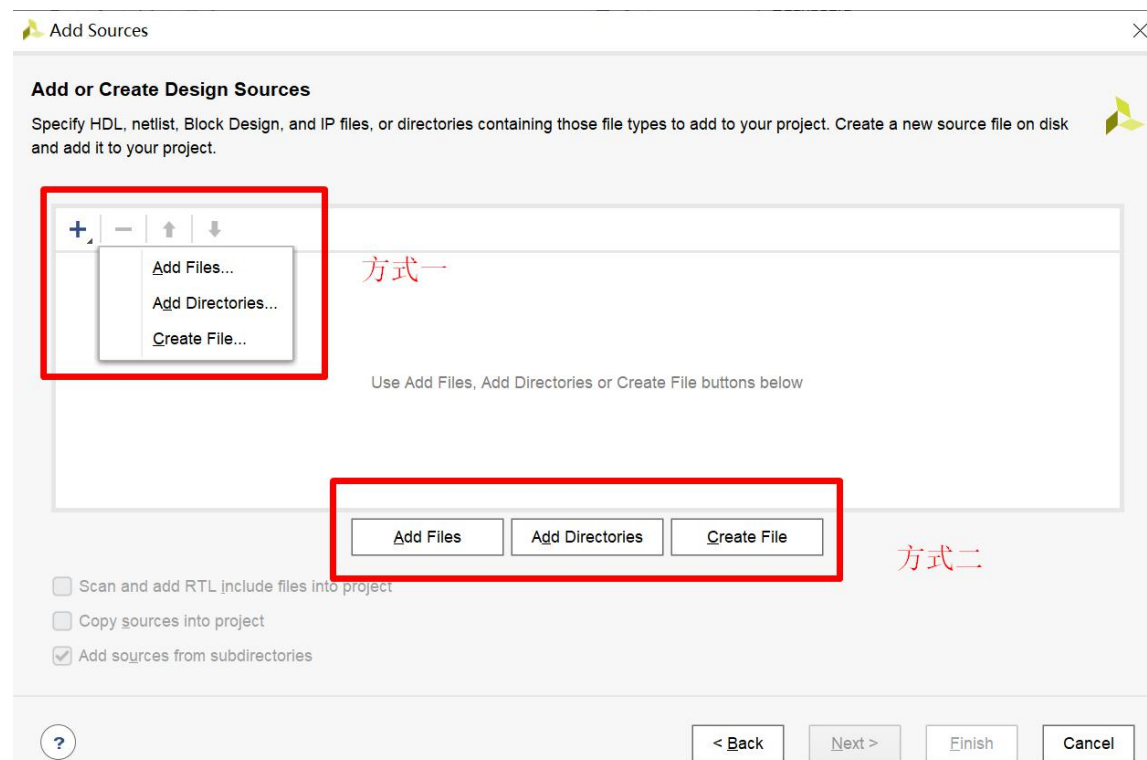


图 1-8 添加或创建设计文件对话框

此时会看到 Create Source File 对话框，File type即文件所用开发语言，默认是Verilog不用修改。File location即创建的文件存储路径，使用默认方式。中间的File name即文件名根据功能或需求命名，文件名称和位置路径中不能出现中文和空格。该实验按照图 1-9 所示填写后点击 OK。

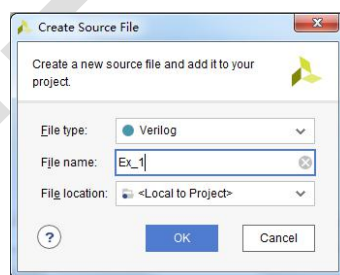


图 1-9 Create Souese File 对话框

此时会看到图 1-10 所示的界面，即显示待添加的所有文件列表，继续添加设计文件或者修改已添加设计文件设置，一次可以添加多个文件和多个目录，该实验我们只需要添加一个。

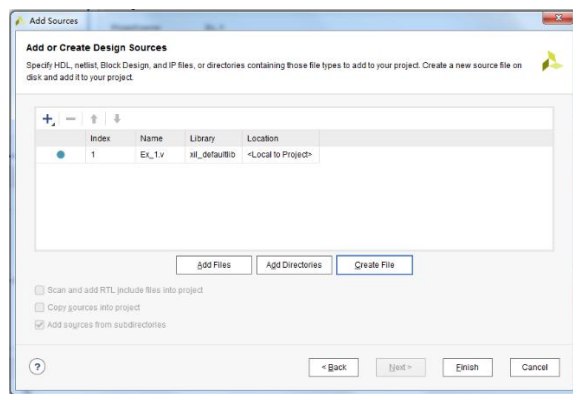


图 1-10 创建设计文件后的添加或创建设计文件对话框

点击 Finish，会弹出如图1-1所示的Define Module对话框，此处module即Verilog里面定义的module，这里可以通过图形化设置module的输入输出端口，设置好会自动生成对应的端口列表。当然也可以不进行配置而直接写代码即可以在这一步直接点“OK”跳过。这里参照图1-11进行设置，然后点击 OK(特别要注意Direction的设置)。如果端口为总线型，勾选“Bus”选项，并通过“MSB”和“LSB”确定总线宽度。

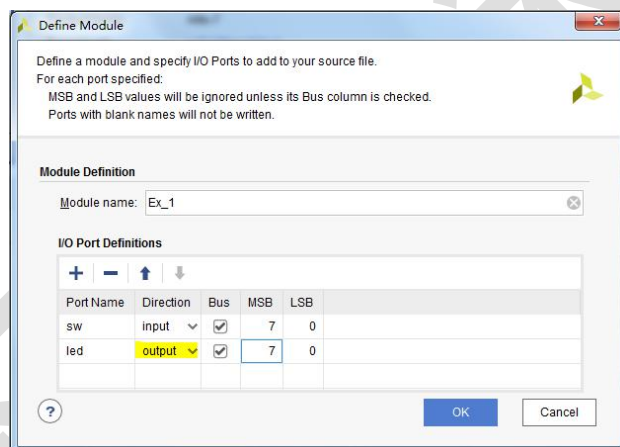


图 1-11 Define Module 对话框

接下来在图 1-12 所示界面上双击 Ex\_1 文件就会在右边显示初始的 Ex\_1 文件内容。

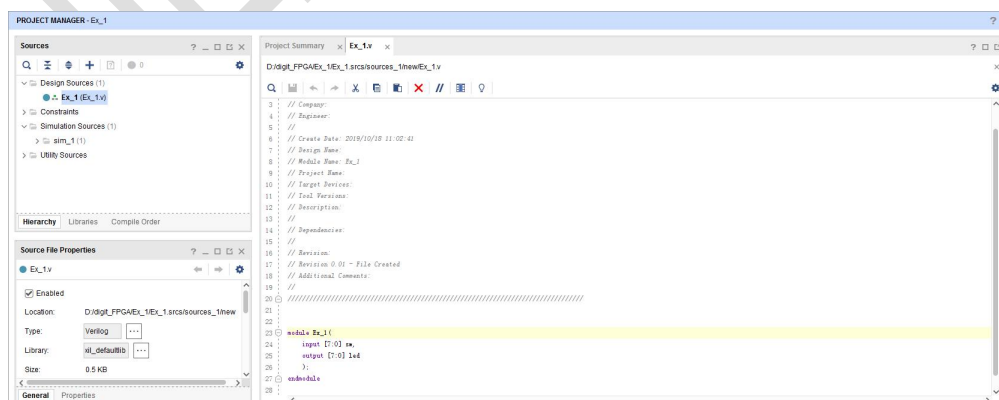


图 1-12 打开Ex\_1.v



可以看到文件中 Ex\_1 的模块是空的，但在Define Module设置了输入输出所以自动生成了sw和led信号。

```
module Ex_1(  
    input [7:0] sw,  
    output [7:0] led  
);  
  
endmodule
```

用下面的程序将这个空模块替代掉。

```
module Ex_1(  
    input [7:0] sw,  
    output [7:0] led  
);  
  
    assign led=sw;  
  
endmodule
```

这个模块很简单，就是将拨码开关上的高低电平赋值给 LED。

#### (4) 仿真

仿真就是为了验证电路设计是否正确，通过给待验证的设计输入测试信号（添加激励），观察它的输出是否符合预期。添加仿真文件的过程同“（3）添加源文件”。右键点击 **Project Manager** 下面 **Sources** 中的 **Simulation Sources**，在弹出的菜单中选择Add Source…。按照图1-13 所示的设置点击**Next**。

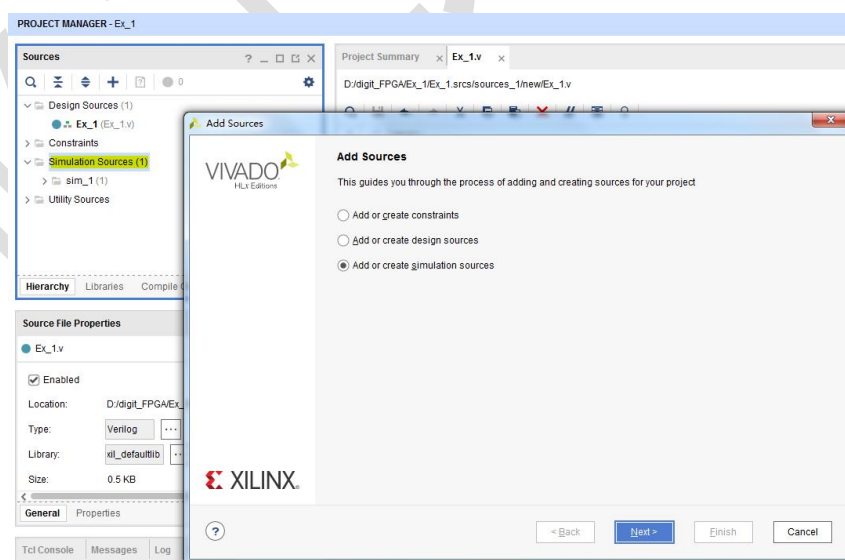


图 1-13 添加仿真源程序

这里注意图中的红框，指定了仿真集sim\_1，仿真集即一个子目录。该机制可以让用户将不同设计阶段所用到的不同仿真文件添加到各自的仿真集中。比如一个仿真集可以添加用



于RTL行为级仿真的源文件，另一个仿真集添加用于实现后时序仿真的源文件。或者不同的仿真集提供对一个设计的不同测试方法。

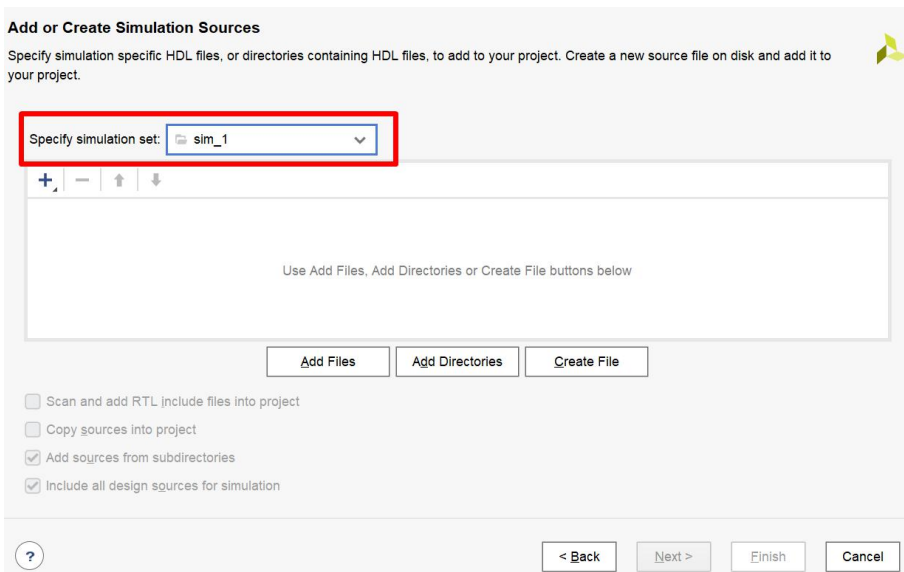


图 1-14 增加仿真源程序第二步——Add Sources 对话框

选择 Create File, 然后设置仿真源文件的文件名为 Ex\_1\_sim如图 1-15 , 完成之后点击ok。仿真文件推荐使用\_sim结尾，命名更清晰。

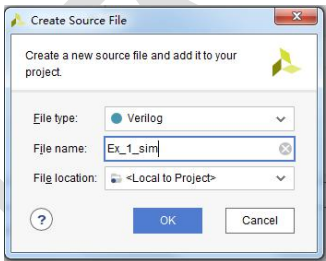


图 1-15 设置仿真源文件文件名

现在回到了 Add Sources 对话框，点击 Finish。由于激励测试文件不需要有对外的接口，在接下来的 Define Module 窗口中直接点击OK，接下来弹出的窗口（如图 1-16）点击 Yes。

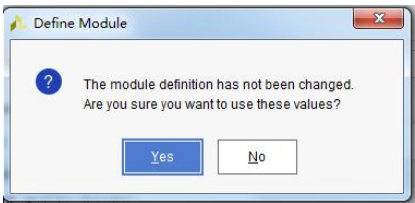


图 1-16 Define Module 窗口

如图 1-17 所示，现在在项目中看到了这个仿真源文件（图中高亮部分）。所有的仿真文

件都会自动放在Simulation Sources下面，并自动生成Sim\_1目录。

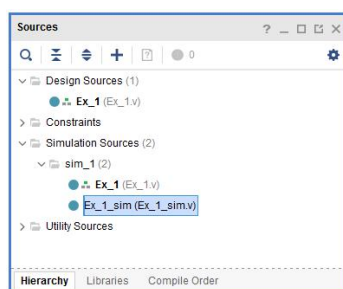


图 1-17 项目中的仿真源文件

双击图 1-17 中的高亮部分，打开该文件，完成对将要仿真的module的实例化和激励代码的编写。可以看到目前的模块定义为：

```
module Ex_1_sim(  
  
);  
  
Endmodule
```

用下面的代码替代。

```
`timescale 1ns/1ps          //1ns表示延时单位，1ps表示时间精度  
  
module Ex_1_sim();  
    reg [7:0] sw = 8'h00;    //input  
    wire [7:0] led;          //output  
    Ex_1 uut(  
        .sw(sw),              // 第一步实例化被测试对象  
        .led(led)             // 激励信号sw连接到被测试模块的sw  
    );  
    always #10 sw =sw+1;      // 第二步添加激励，每隔10个单位时间将sw加1  
endmodule
```

仿真代码编写分两步：第一步是对被测试设计的顶层接口进行实例化，表示调用哪个模块。第二步是给被测试设计的输入接口添加激励，即改变输入信号。上面的代码首先例化了 Ex\_1 模块，对 sw 初始化为 0，然后改变sw的取值。

设计文件新建完成后，在Design Sources和Simulation Sources中都有，而仿真文件只会出现在Simulation Sources文件夹中。设计文件可以用于仿真，也可以用于产生最终烧写进开发板的比特流，而仿真文件仅用于仿真。

保存好 Ex\_1\_sim.v 文件后，得到的项目文件层次如图 1-18 所示



图 1-18 编辑完 Ex\_1\_sim.v 后的项目文件层次

在如图 1-19 所示的 Flow Navigator->Simulation 中点击 Run Simulation，在弹出的菜单中选择 Run behavior Simulation即行为仿真也叫功能仿真。

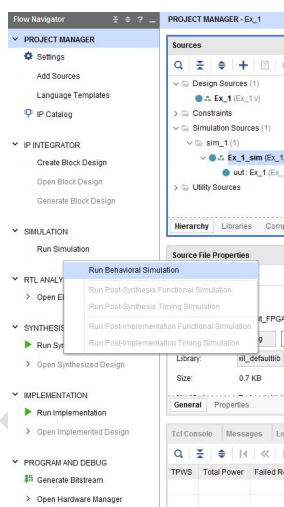



图 1-19 点击 Run Simulation

仿真完后，会自动弹出仿真波形图。整个仿真窗口包扩scope、object、波形窗口三部分，波形窗口又包含Name、value、波形三部分。点击 ，使Cursor回到0时刻，再按住Ctrl键向下滚动滚轮缩放时间轴，得到如图 1-20 所示的波形图。从图上可以明显地看到，每隔 10ns，输入数据 sw 加 1，输出数据 led 与 sw 同步变化。仿真工具栏上有很多快捷操作按钮，将鼠标放到对应的图标上会弹出说明。

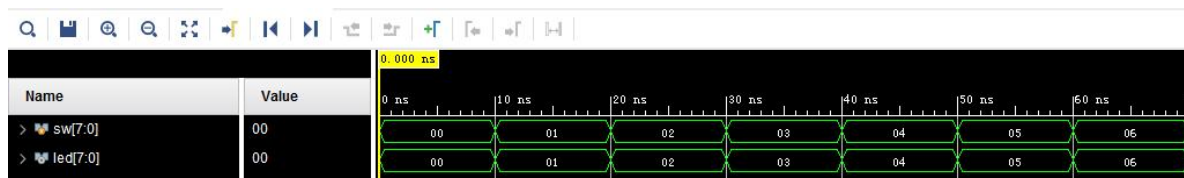


图 1-20 仿真波形图

## (5) 添加约束文件

在该实验中，用到了sw信号表示拨码开关输入，led信号用于led输出。为了上板运行，

还需将sw和led信号与Ego开发板上的FPGA芯片的引脚对应起来，即物理约束。

单击右上角，退出Simulation界面，在弹出的Confirm Close中点选OK，回到Project Manager界面。右键点击 Project Manager 下面 Sources 中的 Constraints，在弹出的菜单中选择 Add Source…。按照图 1-21 所示的设置点击 Next。

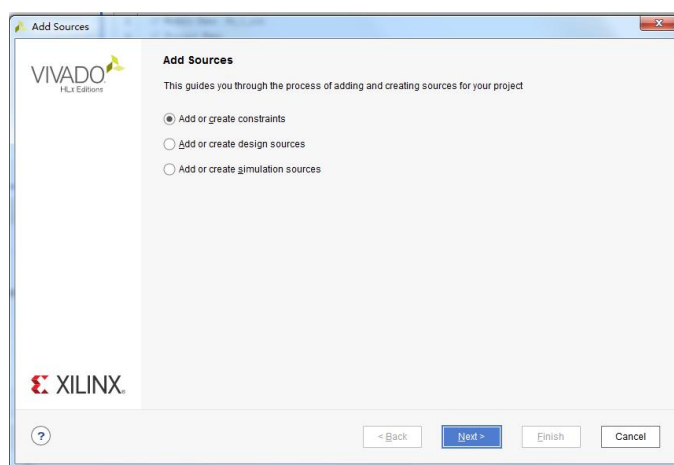


图 1-21 添加约束文件

约束文件初始为空，拷贝Ex\_1.xdc的内容完成约束文件添加，下面截取约束文件的部分内容进行说明。

```
set_property PACKAGE_PIN P5 [get_ports sw[0]]
set_property PACKAGE_PIN F6 [get_ports led[0]]
#语法格式为set_property PACKAGE_PIN (pin location) [get_ports (port name)]
set_property IOSTANDARD LVCMOS33 [get_ports sw[0]]
set_property IOSTANDARD LVCMOS33 [get_ports led[0]]
#语法格式为set_property IOSTANDARD (level:LVDS,LVCMOS18,LVCMOS33 etc.) [get_ports (port name)]
```

约束文件主要包含两部分内容，第一部分是信号和管脚的对应关系，比如第一行就是将fpga芯片的P5管脚和信号sw[0]信号绑定。第二部分是管脚的属性设置，IOSTANDARD全部设置成 LVCMOS33即IO电平标准是CMOS 3.3v。完整的约束根据表 1-1，对每个管脚进行分配。（管脚分配既可以参考实验板的说明手册，也可以直接在实验板丝印上读出，如SW0-R1说明开关SW0对应管脚R1）。

约束文件除了物理约束还有时序约束，暂不做过多讲解，可以参考[约束功能概述](#)。

表 1-1 Ex\_1 管脚分配表

信号	部件	管脚	信号	部件	管脚
led[0]	D1	F6	sw[0]	SW0	P5
led[1]	D2	G4	sw[1]	SW1	P4
led[2]	D3	G3	sw[2]	SW2	P3
led[3]	D4	J4	sw[3]	SW3	P2
led[4]	D5	H4	sw[4]	SW4	R2
led[5]	D6	J3	sw[5]	SW5	M4
led[6]	D7	J2	sw[6]	SW6	N4
led[7]	D8	K2	sw[7]	SW7	R1

需要注意的是板子标注有两种，如下图所示，如果拿到的板子是图1-22（a）所示的，后续实验需要自行调整约束信号的顺序，该实验不用调整。



(a)



(b)

图1-22 Ego开发板两种标注

## (6) 综合(Synthesis)

**综合：**将高级抽象层次的语言描述转化成较低层次的电路结构。即将verilog代码映射为门级网表（与非门等基本逻辑单元的互联关系）。综合的过程有翻译、映射、优化。翻译即translate，将代码转成基本的与或非等器件无关的逻辑电路。映射即map，将逻辑电路映射成FPGA基本单元，比如LUT，RAM。优化包括去除冗余的电路结构，或者复用功能相同的电路结构。

在Flow Navigator中点击Synthesis -> Run Synthesis并在弹出的列表中点选Run Synthesis。综合如果没有问题，会弹出“Synthesis sucessfully completed”窗口，并给出后续步骤的三个选项，这里直接点击“Cancel”。

按下面步骤查看管脚分配结果：在左侧Flow Navigator中展开RTL ANALYSIS, 点击Open Elaborated Design，如果弹出Elaborated Design对话框，直接跳过，点选OK，由此进入

Elaborated Design界面，调出该界面后再点击菜单栏Layout->I/O Planning，在界面下部可以看到管脚分配情况如下图所示：

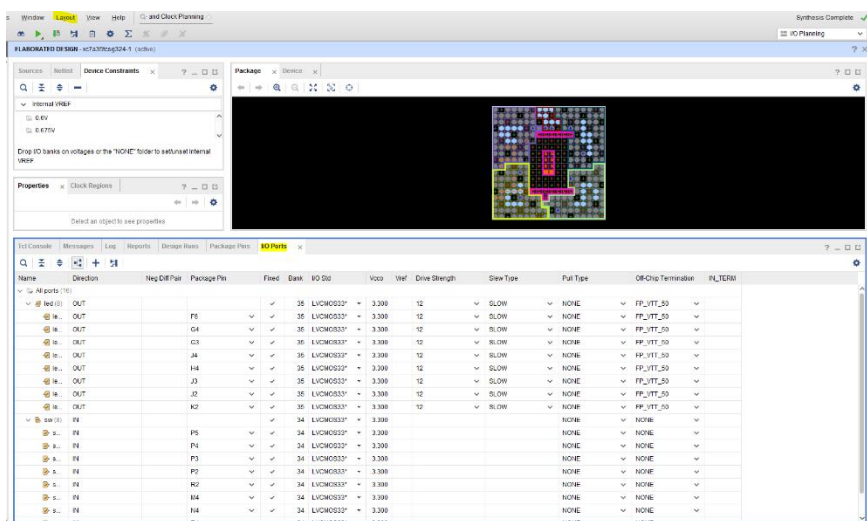


图 1-23 管脚分配后查看结果图

因为前面手动添加过约束文件，所以这里信号和管脚的对应关系已经建立。如果之前没有添加约束文件，则可以在这里进行物理约束设置。显然这个方式不够高效，如果输入输出信号有50个，那需要重复50次设置，文件编辑的速度显然更快。

### (7) 实现(Implementation)

综合后生成的门级网表只是表示了门与门之间的虚拟的连接关系（逻辑连接），并没有规定门的位置、走线等。Implementation则是将门级网表转化为布局布线后的版图，即implementation包括布局和布线两步。布局，把综合后的基本单元放到器件的各个位置。布线，把各个单元连接起来。布局布线需要考虑是时序优先、资源优先（面积）、功耗优先。

关闭Elaborated Design界面，在Flow Navigator 中点击Implementation ->Run Implementation 来对设计进行实现。实现完后出现图 1-24 的窗口。默认是进行下一步生成比特流文件，可以选择cancel取消。这里我们选择“Open implemented Design”查看工程实现结果。可能会出现综合过时的提示，如果出现需重新运行。

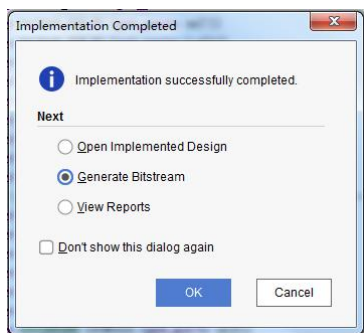


图 1-24 实现后的窗口

## (8) 产生比特流文件并下载

FPGA上板的最后一步是将项目生成bit文件，用于配置FPGA的配置比特流。按照图 1-24 的设置点击 OK，或者在 Project Manager 中点击 Program and Debug-> Generate Bitstream。比特流生成后会出现图 1-25 所示的对话框。

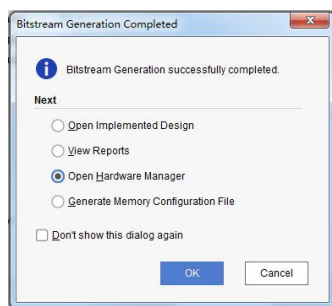


图 1-25 比特流生成完成

用USB\_JTAG线将EGO1实验板的JTAG（J22）与PC机的USB相连，**打开EGO-1 板的电源**（如果不打开电源则无法连接！！这里注意拿板子的方式，手握板子边缘不要用手直接接触开发板的元器件，避免静电烧坏芯片）。按照图 1-25 所示选中 Open Hardware Manager。点击OK，出现Hardware Manager界面。

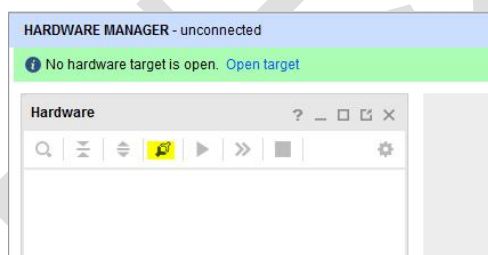


图 1-26 Hardware Manager

在左侧 Flow Navigator 中点击 Program and Debug->Open Hardware Manager->Open Target->Auto Connect,进行自动连接。

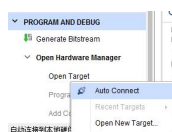


图 1-27 连接硬件

硬件连接上后，出现如图 1-28 所示界面。

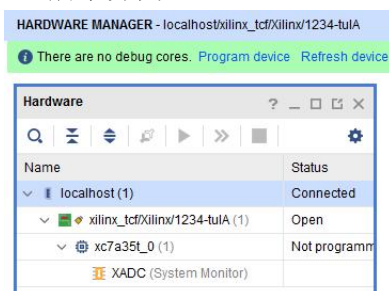




图 1-28 连接上硬件后

点击 Program devices->xc7a100t\_0。出现的 Program Device 窗口（如图 1-30）中点击 Program。

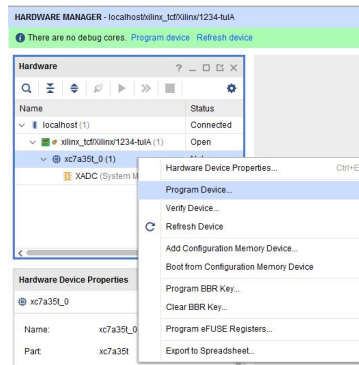


图 1-29 调出Program Device 窗口

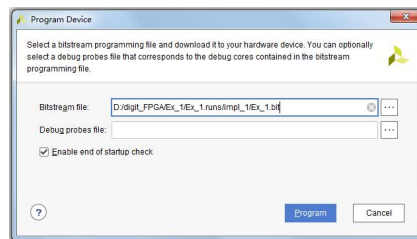


图 1-30 Program Device 窗口

出现如图 1-31 所示的正在下载的进度条。



图 1-31 下载进度条

在上板调试环节如果发现Vivado不能正常识别开发板时，请按照下列步骤排查：

（a）检查FPGA开发板是否上电即电源指示灯是否亮？检查FPGA配置下载器的USB电缆是否与电脑以及适配器正常连接？

（b）检查Cable Driver驱动是否正常安装。若驱动正常安装，你可以在windows设备管理器的界面中查看到“Programming cables→Xilinx USB Cable”。如果找不到，说明下载器USB Cable的驱动未正常安装，参考d)进行安装。

（c）如果发现下载器USB Cable的驱动未正常安装，请参考<https://china.xilinx.com/support/answers/59128.html>进行驱动的安装。如果该方法不行，推荐Google或在xilinx官网上找寻针对你的问题的解决方法。

（d）在Vivado Tcl Console面板下输入disconnect\_hw\_server命令并在Open target的时候选择Auto Connect。

- 
- (e) 重启Vivado软件，重复上面的步骤。
  - (f) 重启你的电脑，重复上面的步骤。
  - (g) 找一个已经证实能被其它电脑识别出的实验箱，如果用你的电脑连上后能识别，拿着你手头不能识别的那个实验箱来找助教和老师。
  - (h) 如果上面的手段尝试了都没有效果，找助教和老师。

下载结束后，会看到 EGO-1 板上的 LED 灯会随着拨码开关的变化而变化，正确的现象为：开关拨上，其上面对应的贴片LED亮起；开关拨下，其上面对应的贴片LED熄灭。

实验题目一结束，关闭 Hardware Manager。

## 2 实验题目二：3-8译码器的Verilog实现

### 2.1 实验内容

使用 Verilog 实现如图2-1所示的3-8译码器（同教材74x138，信号名不同），拨码开关作为译码器的输入，输出驱动led显示，运行仿真来验证你的实现，并上板验证。

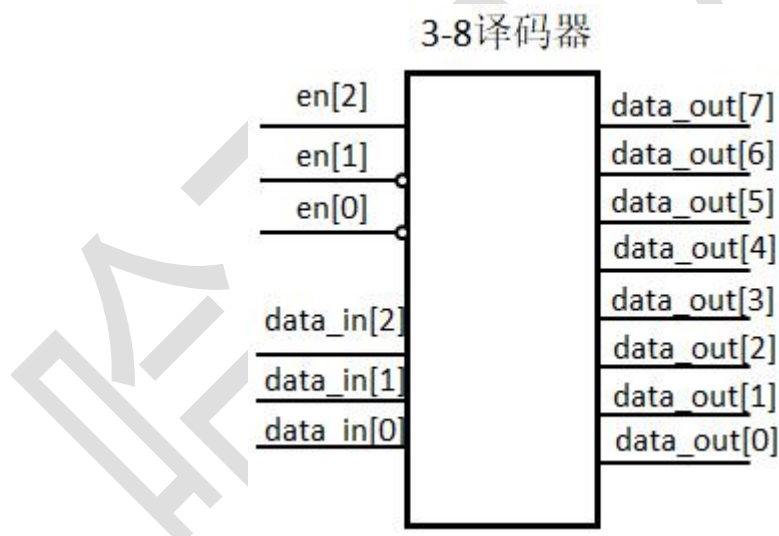


图2-1 3-8译码器电路框图

### 2.2 实验原理

3-8译码器真值表如下：

表2-1 3-8译码器真值表

en_i[2:0]			data_i[2:0]			data_o[7:0]							
0	x	x	x	x	x	1	1	1	1	1	1	1	1
X	1	x	x	x	x	1	1	1	1	1	1	1	1
X	x	1	x	x	x	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0	0	1	0
1	0	0	0	1	0	0	0	0	0	0	1	0	0
1	0	0	0	1	1	0	0	0	0	1	0	0	0
1	0	0	1	0	0	0	0	0	1	0	0	0	0
1	0	0	1	0	1	0	0	1	0	0	0	0	0
1	0	0	1	1	0	0	1	0	0	0	0	0	0
1	0	0	1	1	1	1	0	0	0	0	0	0	0

各信号与FPGA管脚对应如下表，信号名可以使用其他名称，但需保证逻辑功能与描述一致。

表2-3 引脚分配图

信号	部件	管脚	信号	部件	管脚
data_out[7]	D7	F6	data_in[0]	SW0	R1
data_out[6]	D6	G4	data_in[1]	SW1	N4
data_out[5]	D5	G3	data_in[2]	SW2	M4
data_out[4]	D4	J4	en[0]	SW3	R2
data_out[3]	D3	H4	en[1]	SW4	P2
data_out[2]	D2	J3	en[2]	SW5	P3
data_out[1]	D1	J2			
data_out[0]	D0	K2			

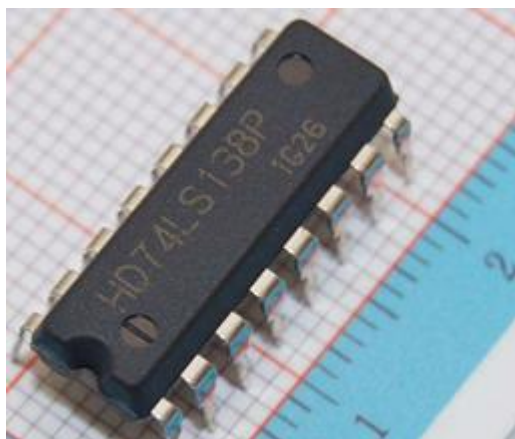


图2-2 3-8译码器实物图

## 2.3 实验步骤

### (1) 项目创建和设计源文件添加

参照指导书“实验题目一”的内容独立完成项目的创建和源文件的添加，新建工程decoder\_38, 设计源文件命名为decoder\_38.v。

### (2) 功能实现

Verilog实现一个电路有多种描述方法，不同的描述方法应用场景不同。对于3-8译码器，有三种实现方式。

**方式一：结构描述法**, 通过调用库中的元件（基本的与非门）或已设计好的模块来完成设计。

根据真值表得到3-8译码器的电路结构图，如下图，然后使用对应的与非门进行描述。

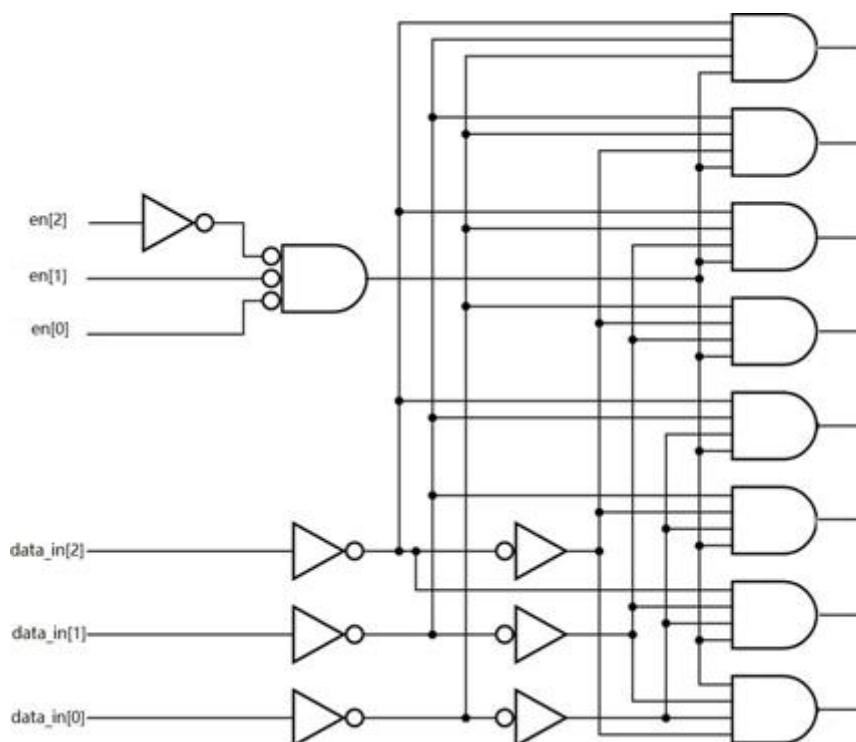


图2-3 3-8译码器电路结构图

下面以处理控制信号为例，因Verilog基本元件限制，所用到的基本门与上图稍有不同。

```
module decoder_38_struct (
    input      [2:0]  data_i,
    input      [2:0]  en_i,
    output reg  [7:0]  data_o
);

    // 内部连线定义
    wire en_out, en_not[1:0];

    // 调用基本库元件完成电路连接
    not U_not_0(en_not[0], en_i[0]);
    not U_not_1(en_not[1], en_i[1]);
    and U_and_0(en_out, en_not[0], en_not[1], en_i[2]);

    // 其他电路连接
    .....

endmodule
```

图2-4 结构描述法

**方式二：**数据流描述法，又称为寄存器传输级（RTL）描述法，从数据的变换和传送角度描述模块，主要使用持续赋值语句，多用于组合逻辑电路。这里根据真值表推导出每个输出信号与输入的逻辑函数。

比如  $\text{data\_out}[5] = \text{data\_in}[2] \cdot \sim \text{data\_in}[1] \cdot \text{data\_in}[0]$ （教材给的逻辑关系是  $Y5\_L = G1 \cdot G2A\_L \cdot G2B\_L \cdot C \cdot B' \cdot A$ ），使用assign语句完成赋值。依次推导出其他输出位与输入信号的关系，并一一用assign语句实现即可。

```
module decoder_38_rtl (
    input      [2:0]  data_i,
    input      [2:0]  en_i,
    output reg  [7:0]  data_o
);

    assign data_o[5] = data_i[2] & ~data_i[1] & data_i[0];
    // or assign data_o[5] = (data_i == 3'd5);

    // 其他电路连接
    .....
endmodule
```

图2-5 数据流描述法

**方式三：行为级描述法**，描述清楚输入与输出信号的行为，无需知道具体电路结构，抽象程度最高。除了用case语句，也可以用if-else语句实现。

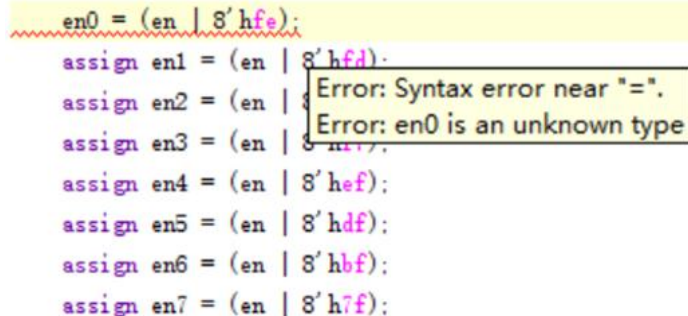
```
module decoder_38 (
    input      [2:0]  data_i,
    input      [2:0]  en_i,
    output reg  [7:0]  data_o
);

    always @(*) begin
        case (data_i)
            3'b000: data_o = 8'b0000_0001;
            3'b001: data_o = 8'b0000_0010;
            3'b010: data_o = 8'b0000_0100;
            3'b011: data_o = 8'b0000_1000;
            3'b100: data_o = 8'b0001_0000;
            3'b101: data_o = 8'b0010_0000;
            3'b110: data_o = 8'b0100_0000;
            3'b111: data_o = 8'b1000_0000;
            // default:
        endcase
    end
    // 其他电路连接
    .....
endmodule
```

图2-6 行为描述法

请思考这三种方式的优缺点，并自行选择其中的一种方式完成最终的实现，以上代码只是参考，需自行补全控制信号和边界条件的处理。

Verilog代码如果出现语法错误，会以波浪线提示，鼠标放上去会弹出具体的提示。



```
en0 = (en | 8'hfe);  
assign en1 = (en | 8'hfd);  
assign en2 = (en | 8'hfd);  
assign en3 = (en | 8'hfd);  
assign en4 = (en | 8'hfe);  
assign en5 = (en | 8'hdf);  
assign en6 = (en | 8'hbf);  
assign en7 = (en | 8'h7f);
```

图2-7 错误提示

### (3) 仿真与约束

功能实现之后第一步仍然是进行仿真，尽管该模块功能很简单。一个好的仿真应该能覆盖到功能模块的所有场景，就跟软件测试一样，测试用例既要覆盖到正常功能场景，也要覆盖所有的边界条件。对译码器的仿真应包含使能端有效和使能端无效两种场景，使能端有效的场景应该覆盖所有的输入情况，使能端无效的场景可以覆盖所有的输入也可以只覆盖部分输入。

仿真文件内容如下，添加仿真文件后运行仿真，看输出是否与真值表预期一致。

```
`timescale 1ns/1ps // 1ns 表示延时单位, 1ps 表示时间精度  
  
module decoder_38_sim ();  
  
    // 输入端口  
    reg [2:0] data_in;  
    reg [2:0] en;  
  
    // 输出端口  
    wire [7:0] data_out;  
  
    // 结合自己的实现完成实例化  
    decoder_38 U_dec38_0 (  
        .data_i      (data_in),  
        .en_i        (en),  
        .data_o      (data_out)  
    );  
  
    initial begin
```



```

// 构造输入激励信号
#5 begin en = 3'b100; data_in = 3'b000; end
#5 begin en = 3'b100; data_in = 3'b001; end
#5 begin en = 3'b100; data_in = 3'b010; end
#5 begin en = 3'b100; data_in = 3'b011; end
#5 begin en = 3'b100; data_in = 3'b100; end
#5 begin en = 3'b100; data_in = 3'b101; end
#5 begin en = 3'b100; data_in = 3'b110; end
#5 begin en = 3'b100; data_in = 3'b111; end
// 使能端无效
#5 begin en = 3'b101; data_in = 3'b000; end
// 结束仿真
#5 $stop;

end

endmodule

```

波形显示将data\_out设置为二进制。

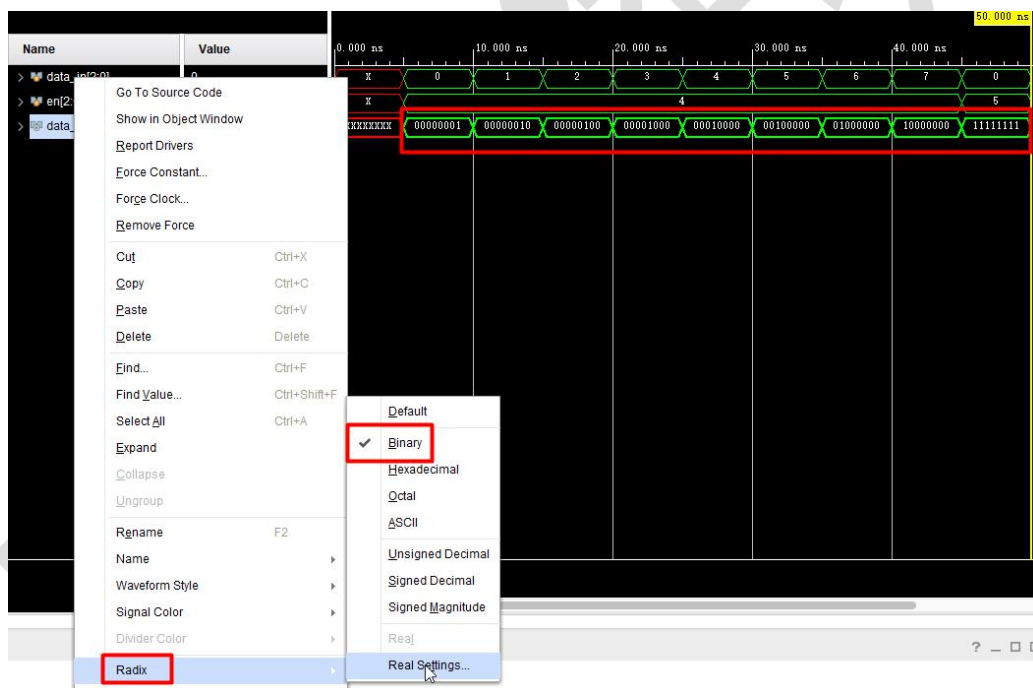


图2-8 3-8译码器仿真输出

添加约束文件，注意信号名和管脚的对应关系。

表2-1 3-8译码器管脚分配表

信号	部件	管脚	信号	部件	管脚
data_out[0]	D1	F6	data_in[0]	SW0	P5
data_out[1]	D2	G4	data_in[1]	SW1	P4
data_out[2]	D3	G3	data_in[2]	SW2	P3
data_out[3]	D4	J4	en[0]	SW3	P2
data_out[4]	D5	H4	en[1]	SW4	R2
data_out[5]	D6	J3	en[2]	SW5	M4
data_out[6]	D7	J2			
data_out[7]	D8	K2			

#### (4) 综合、实现与下载

独立完成综合、实现、生成比特流文件的流程，也可以直接双击“Generate BitStream”自动运行综合、实现和生成比特流文件。上板之后拨动拨码开关，验证与真值表的输出是否一致。

对于部分编码错误，不影响仿真，但在综合的时候会报错，仿真和综合都尽量看下Messages窗口如下图。重点关注是否有Errors和Critical Warnings。Errors意味综合失败只有综合成功的才能走后续的流程；Critical Warnings表示所设计的电路中存在潜在的不确定性或其他会引起电路不能正确运行的非语法因素；Warnings是较为次要的警告，一般可以忽略（除非出现0 error和0 critical warning，但仍然不能综合/实现/生成比特流。此时应当先检查Warnings，并在必要时重新建立工程）。

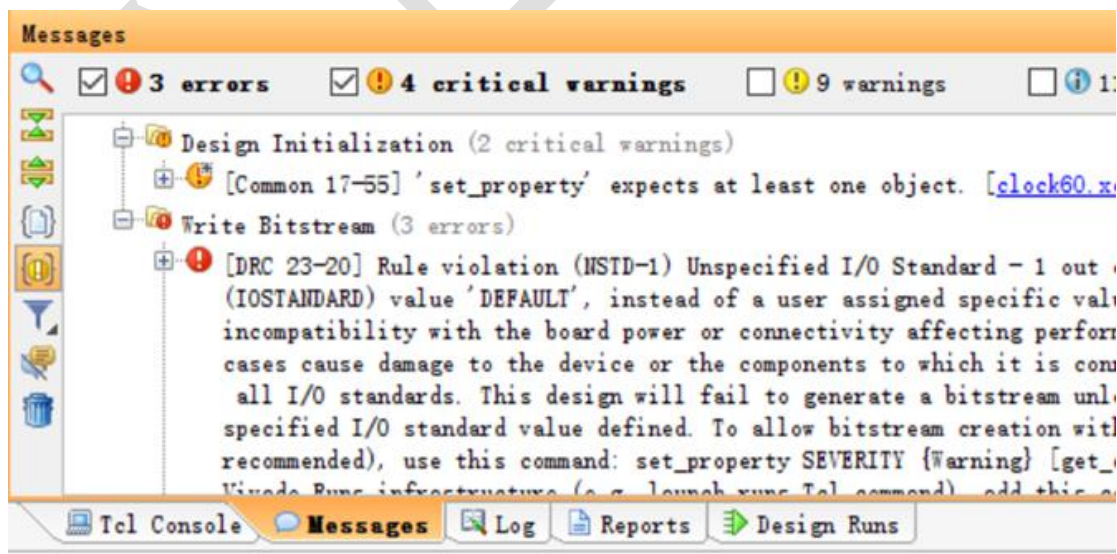


图2-9 vivado18.3版本的Messages窗口

为了进一步理解vivado和Verilog开发过程，Synthesis之后分别按图2-10和2-11所示打开

schematic。图2-10为RTL图（寄存器级传输图），从图中可以清晰看到三八译码器最终由一个ROM模块（只读存储）和一个多路选择组成。RTL图是语法分析得到的结果，类似于用原理图设计电路的样子，并非最终电路结构，是设计输入的最直接的体现,它的主要作用是帮助设计者检查设计输入中的问题。

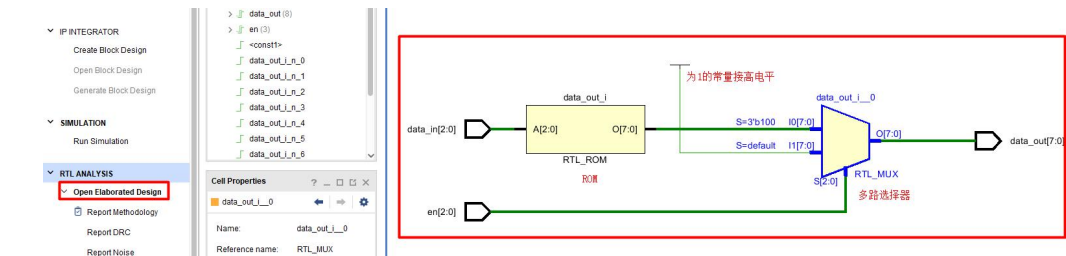


图2-10 RTL图

综合的电路图（图2-11）是由各种输入输出缓冲（ibuf）（obuf）以及LUT组成，FPGA内部就是由这些来组成一系列的电路，反映了实际的电路和资源使用情况。上面介绍的三种Verilog实现方式，综合之后得到的电路图是不一样的，可自行对比下。

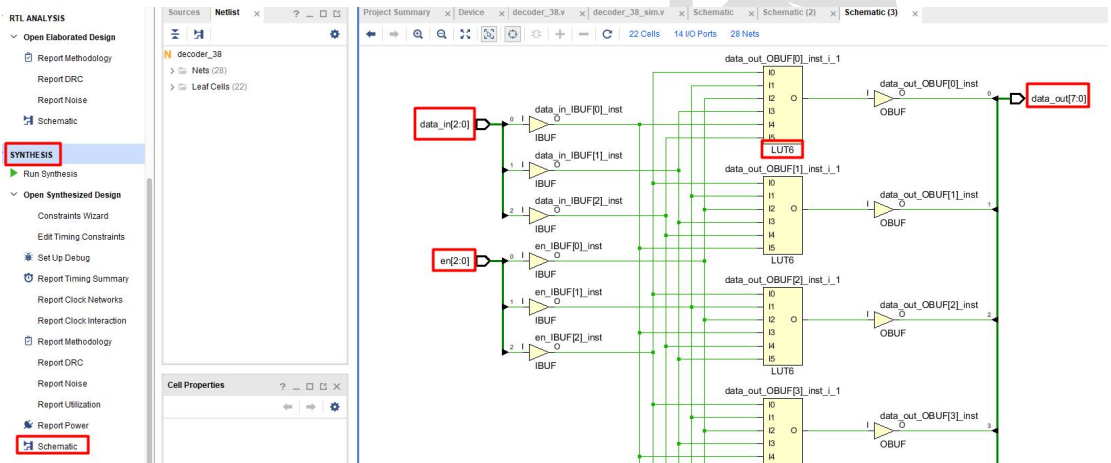


图2-11 综合后结果图

Implementation完成之后选择“Open Implemented Design”，打开device界面即器件视图，显示了封装引脚、晶圆焊盘和IO组，如下图所示。其中高亮出来的区域就是有器件使用的地方。

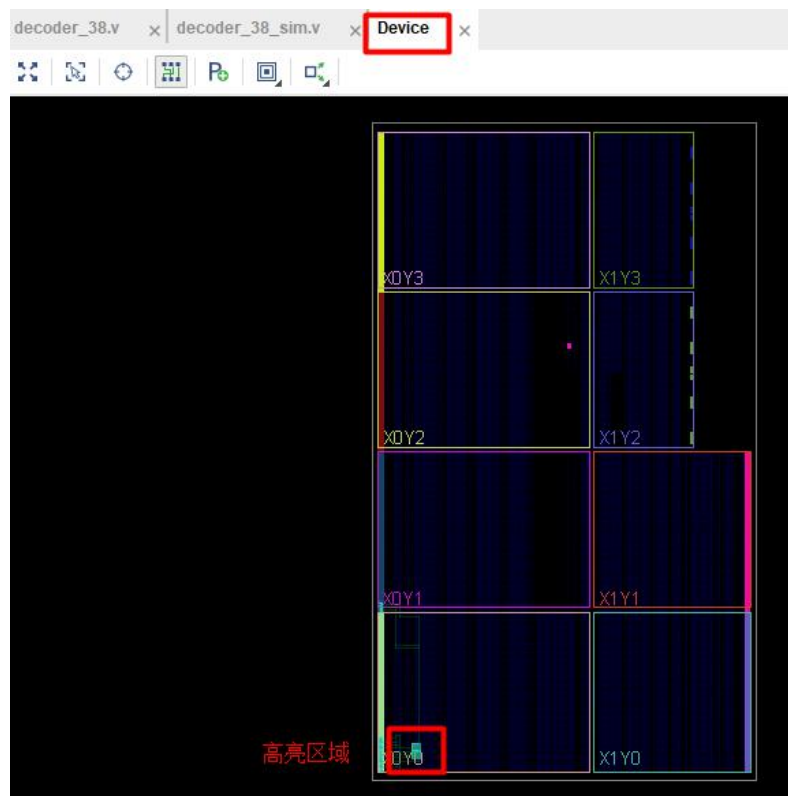


图2-12 FPGA器件内部结构

如下图所示，点击“显示布线资源”按钮，并将图放大到合适大小，可以看到具体LUT资源的利用和绿色的布线。

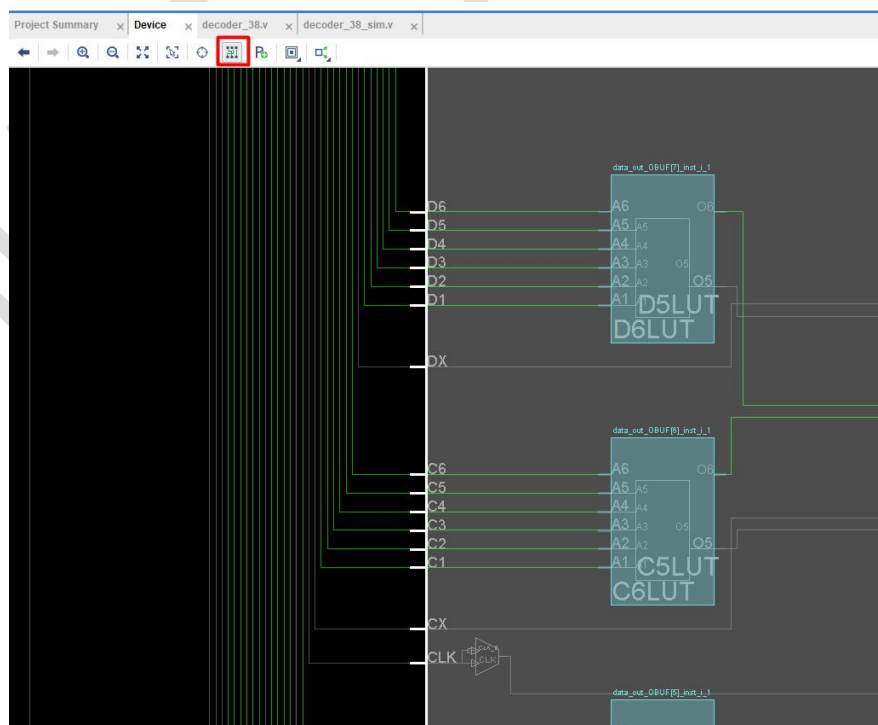


图2-13 布局布线图

### (5) 工程文件要求

工程文件名称: decoder\_38; 主模块文件名: decoder\_38.v;

仿真文件文件名: decoder\_38\_sim.v; 约束文件文件名: decoder\_38.xdc。

## 3、附加题

使用Verilog实现8位二进制代码到格雷码的转换器。4位格雷码和4位二进制码之间的关系如表3-1所示。

表3-1 格雷码与二进制码之间的关系

二进制	0000	0001	0010	0011	0100	0101	0110	0111
格雷码	0000	0001	0011	0010	0110	0111	0101	0100
二进制	1000	1001	1010	1011	1100	1101	1110	1111
格雷码	1100	1101	1111	1110	1010	1011	1001	1000

本题需下板验证，不作仿真要求。下板后，所设计的模块应当能够读取SW7-SW0的值作为输入，并将转换后得到的格雷码显示在LED7-LED0上。管脚分配详情请自行查阅EGO1开发板的数据手册。

本题不作硬性要求，学有余力的同学可自行完成。

## 4、实验检查与提交要求

实验题目一 检查实验现象（1分）；

实验题目二 提交仿真结果截图、工程文件（1分）。