

Vivado 使用指南

简介：

本使用指南将指导读者在 Xilinx Vivado 环境下，使用 Verilog HDL 语言设计一个简单的数字电路样例。一个典型的设计流程包括创建 model，创建用户约束文件，创建 Vivado 项目，导入已创建的 model，编译约束文件，选择性调试运行时的行为仿真，综合你的 design，实现 design，生成 bitstream 文件，最后将 bitstream 文件下载到硬件中，并确认硬件能否正确的实现功能。读者即将学习的设计流程将基于 Artix-7 芯片的 Basys3 基板和 Nexys4 DDR 基板。一个典型的设计流程如下图所示，画圈数字的顺序将和本指南中的指导步骤的顺序一致。

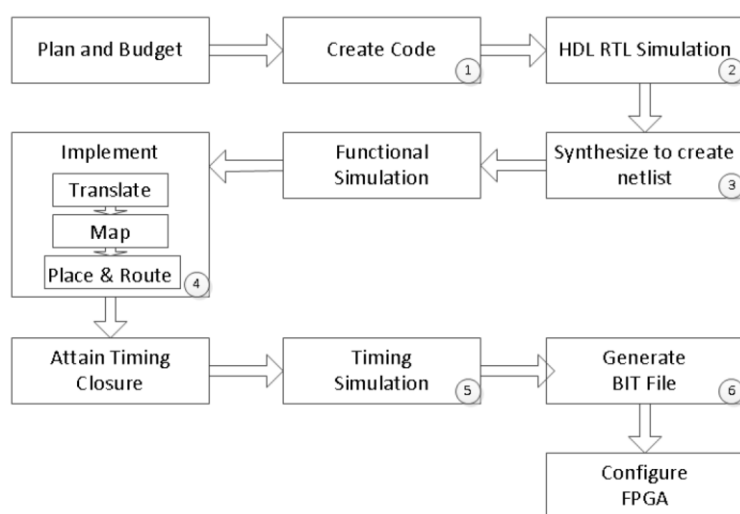


图 1 典型的设计流程

目标：

在完成了本指南的所有内容后，你应该具备以下能力：

- 创建一个采用 HDL 模型的 Vivado 项目，并针对位于 Basys3 和 Nexys4 DDR 板上的特定

FPGA 器件进行开发

- 使用提供的已部分完成的 Xilinx Design Constraint (XDC)文件来约束某些引脚的位置
- 使用 Vivado 的 Tcl 脚本功能来增加额外的约束
- 使用 XSim 仿真器来仿真你的设计
- 综合并实现你的设计
- 生成 bitstream 文件
- 使用已生成的 bitstream 文件配置 FPGA 设备并确认功能

流程

本指南分为几个步骤，每个步骤由一般概述语句和提供后续详细指示的信息组成，请按照这些详细知识逐步完成本指南

设计说明

该设计由一些直接连接到相应 LED 的输入构成，其他输入在向其余 LED 上输出结果之前进行了逻辑操作，如 Figure2 所示。

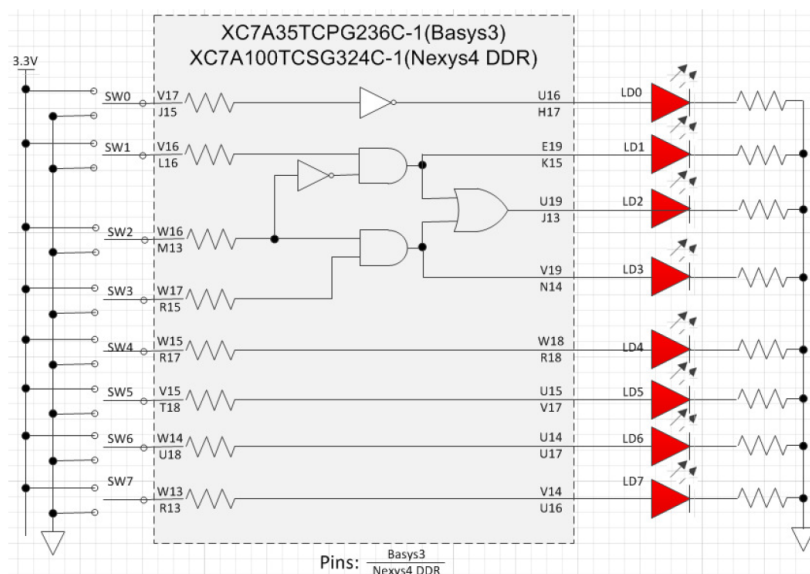


图 2 完成后的设计

本指南的一般流程：

- 创建 Vivado 项目并分析源文件
- 使用 XSim 仿真器仿真设计
- 综合你的设计
- 实现你的设计
- 执行时序仿真
- 使用 Basys3 或 Nexys4 DDR 基板从硬件上验证功能

使用 IDE 创建 Vivado 项目

Step 1

1-1. 启动 Vivado 并创建一个针对 xc7a35tcbg236-1 (Basys3) 或者 xc7a100tcsg324-1 (Nexys4 DDR)设备的项目 ,并使用 Verilog HDL 语言。

使用在 sources / tutorial 目录中提供的 tutorial.v 和

Nexys4DDR_Master.xdc or Basys3_Master.xdc 文件。

- 1-1-1. 打开 Vivado。Start > All Programs > Xilinx Design Tools > Vivado 2015.1 > Vivado 2015.1
- 1-1-2. 单击“Create New Project”以启动向导。你将看到“Create A New Vivado Project”对话框。点击 Next。
- 1-1-3. 单击 New Project 窗体的 Project location 字段的 Browse 按钮，浏览到 c : \ xup \ digital，然后单击 Select。
- 1-1-4. 如 Figure3 在 Project name 中输入 tutorial。确保选中“Create Project Subdirectory”框，点击 Next。

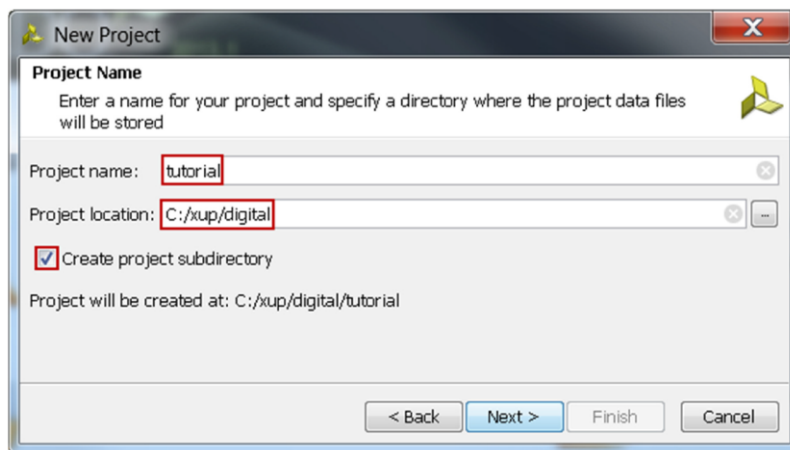


图 3 项目名称和路径

- 1-1-5. 在 Project Type 表单中选择 RTL Project，点击 Next
- 1-1-6. 在 Add Sources 表单中选择 Verilog 作为 Target language 和 Simulator language
- 1-1-7. 单击 Green Plus 按钮，然后单击 Add Files...按钮，浏览到 c : \ xup \ digital \ sources \ tutorial 目录，选择 tutorial.v，单击 Open，并确认已选中 Copy sources into project，然后单击 Next
- 1-1-8. 因为我们没有在此设计中使用任何预先固定的 IP，故单击 Add Existing IP form 表

单中的 **Next** ,

- 1-1-9. 在 *Add Constraints* 表单中, 单击 **Green Plus** 按钮, 然后单击 **Add Files ...**按钮, 浏览到 `c : \xup \digital \sources \tutorial` 目录, 选择 *Basys3_Master.xdc* (对应 Basys3) 或 *Nexys4DDR_Master.xdc* (对应 Nexys4 DDR), 单击 **Open** , 然后单击 **Next**。

XDC 约束文件将 FPGA 上的物理 IO 位置分配给主板上的开关和 LED。这些信息可以通过电路板的原理图或电路板的用户指南获得

- 1-1-10. 如 Figure 4 ,在 *Default Part* 表单中 ,使用 **Parts** 选项和 **Filter** 部分的各种下拉字段, 选择 **xc7a35tcbg236-1** part(对于 Basy3)或 **xc7a100tcsg324-1** part(对于 Nexys4 DDR)。 单击 **Next**。

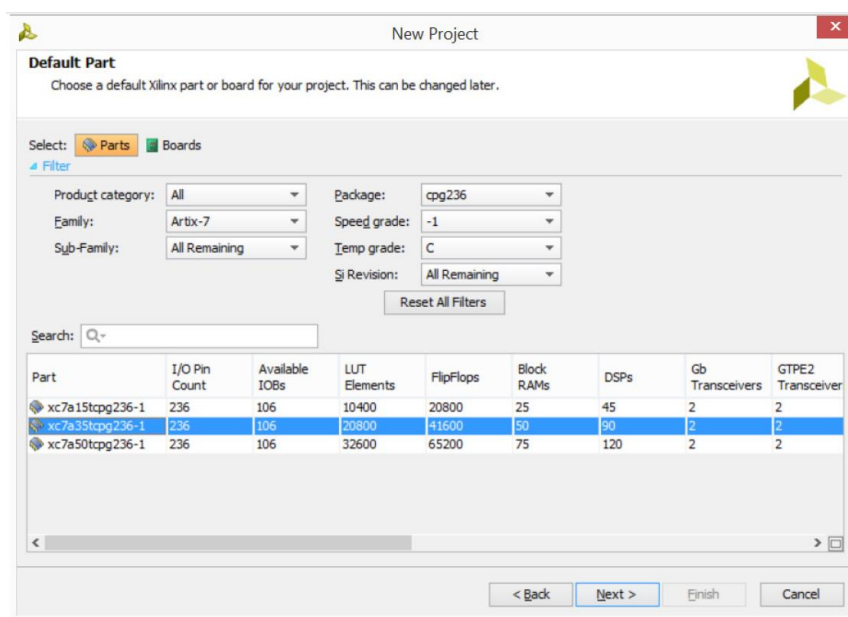


图 4 Basys3 器件选择

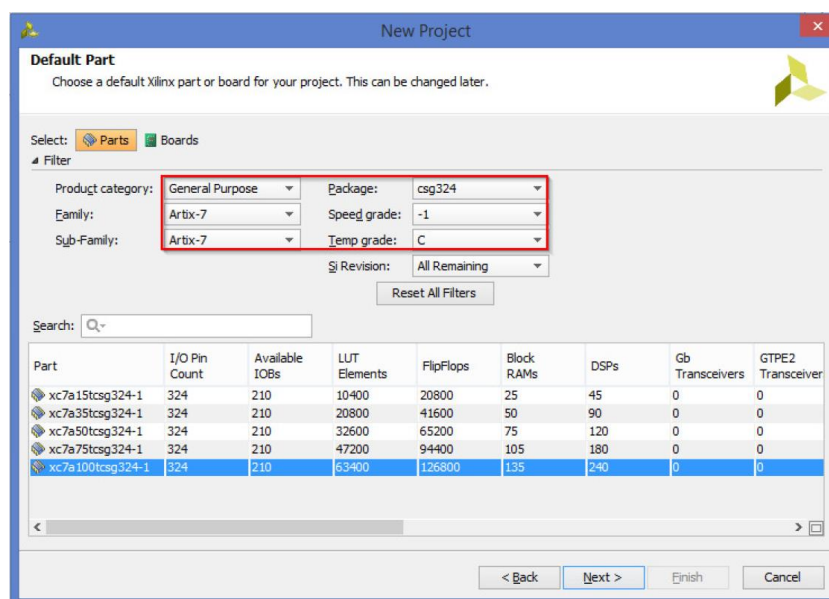


图 4 Nexys4 DDR 器件选择

1-1-11. 单击 **Finish** 以创建 Vivado 项目。

使用 Windows 资源管理器并查看 `c:\xup\digital\tutorial` 目录。你将看到 `tutorial.srcs` 和其他目录以及 `tutorial.xpr` (Vivado) 项目文件已创建。在 `tutorial.srcs` 目录下创建了两个子目录 `constrs_1` 和 `sources_1`; 在它们的下方, 分别放置了复制的 `Nexys4DDR_Master.xdc` 或 `Basys3_Master.xdc` (约束) 和 `tutorial.v` (源) 文件。

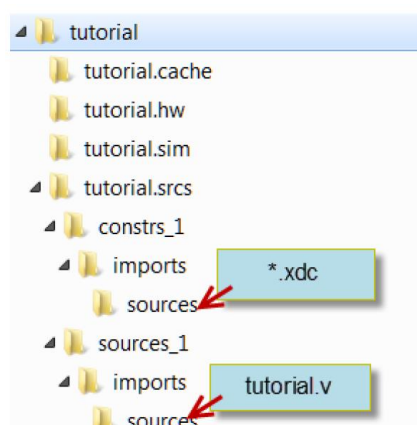


图 5 生成的目录结构

1-2. 打开 tutorial.v 文件并分析内容

1-2-1. 如 Figure6，在 *Sources* 窗格中，双击 **tutorial.v** 条目以在文本模式下打开文件。

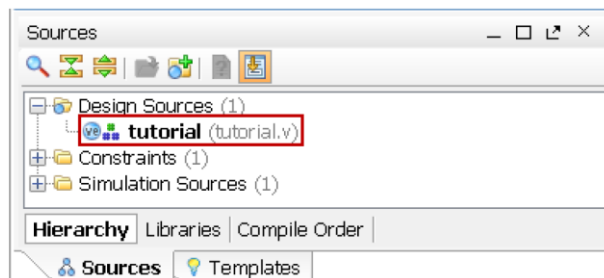


图 6 打开源文件

1-2-2. 请注意，在 Verilog 代码中，第一行定义了仿真器的 timescale 指令。第 2-5 行是描述模块名称和模块用途的注释行

1-2-3. 第 7 行定义了开头（用关键字 **module** 标记），第 19 行定义了模块的结尾（用关键字 **endmodule** 标记）。

1-2-4. 第 8-9 行定义输入和输出端口而第 12-17 行定义实际功能。

1-3. 打开 *Basys3_Master.xdc* 或 *Nexys4DDR_Master.xdc* 源，分析内容并编辑文件。

1-3-1. 如 Figure7，在 *Sources* 窗格中，展开 *Constraints* 文件夹，然后双击 *Basys3_Master.xdc* (*Basys3*) 或 *Nexys4DDR_Master.xdc* (*Nexys4 DDR*) 条目以在文本模式下打开文件。

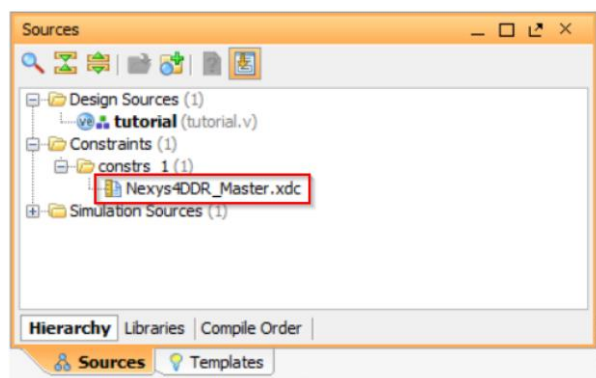


图 7 打开约束文件

1-3-2. 如 Figure8 ,通过删除 # 符号或突出显示 SW [7 :0]并按 CTRL /来取消注释 SW [7 :

0]。取消注释 LED [7 :0] ,引脚名称需要进行更改,以匹配 *tutorial.v* 文件中的引脚

名称。将 *sw* 更改为 *swt* ,将 *LED* 更改为 *led*。

```
## LEDs
set_property PACKAGE_PIN U16 [get_ports {led[0]]}
set_property IOSTANDARD LVCN0533 [get_ports {led[0]]}
set_property PACKAGE_PIN E19 [get_ports {led[1]]}
set_property IOSTANDARD LVCN0533 [get_ports {led[1]]}
set_property PACKAGE_PIN U19 [get_ports {led[2]]}
set_property IOSTANDARD LVCN0533 [get_ports {led[2]]}
set_property PACKAGE_PIN V19 [get_ports {led[3]]}
set_property IOSTANDARD LVCN0533 [get_ports {led[3]]}
set_property PACKAGE_PIN W18 [get_ports {led[4]]}
set_property IOSTANDARD LVCN0533 [get_ports {led[4]]}
set_property PACKAGE_PIN U15 [get_ports {led[5]]}
set_property IOSTANDARD LVCN0533 [get_ports {led[5]]}
set_property PACKAGE_PIN U14 [get_ports {led[6]]}
set_property IOSTANDARD LVCN0533 [get_ports {led[6]]}
set_property PACKAGE_PIN V14 [get_ports {led[7]]}
set_property IOSTANDARD LVCN0533 [get_ports {led[7]]}
#set_property PACKAGE_PIN V13 [get_ports {led[8]]}
#set_property IOSTANDARD LVCN0533 [get_ports {led[8]]}

## Switches
set_property PACKAGE_PIN V17 [get_ports {sw[0]]}
set_property IOSTANDARD LVCN0533 [get_ports {sw[0]]}
set_property PACKAGE_PIN V16 [get_ports {sw[1]]}
set_property IOSTANDARD LVCN0533 [get_ports {sw[1]]}
set_property PACKAGE_PIN W16 [get_ports {sw[2]]}
set_property IOSTANDARD LVCN0533 [get_ports {sw[2]]}
set_property PACKAGE_PIN W17 [get_ports {sw[3]]}
set_property IOSTANDARD LVCN0533 [get_ports {sw[3]]}
set_property PACKAGE_PIN W15 [get_ports {sw[4]]}
set_property IOSTANDARD LVCN0533 [get_ports {sw[4]]}
set_property PACKAGE_PIN V15 [get_ports {sw[5]]}
set_property IOSTANDARD LVCN0533 [get_ports {sw[5]]}
set_property PACKAGE_PIN W14 [get_ports {sw[6]]}
set_property IOSTANDARD LVCN0533 [get_ports {sw[6]]}
set_property PACKAGE_PIN W13 [get_ports {sw[7]]}
set_property IOSTANDARD LVCN0533 [get_ports {sw[7]]}
#set_property PACKAGE_PIN V2 [get_ports {sw[8]]}
#set_property IOSTANDARD LVCN0533 [get_ports {sw[8]]}
```

图 8 编辑 Basys3 约束文件

```
set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCN0533 } [get_ports {swt[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCN0533 } [get_ports {swt[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCN0533 } [get_ports {swt[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCN0533 } [get_ports {swt[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCN0533 } [get_ports {swt[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCN0533 } [get_ports {swt[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCN0533 } [get_ports {swt[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCN0533 } [get_ports {swt[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
#set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVCN0533 } [get_ports {swt[8] }]; #IO_L24N_T3_34 Sch=sw[8]
#set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCN0533 } [get_ports {swt[9] }]; #IO_25_34 Sch=sw[9]
#set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCN0533 } [get_ports {swt[10] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
#set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCN0533 } [get_ports {swt[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
#set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVCN0533 } [get_ports {swt[12] }]; #IO_L24P_T3_35 Sch=sw[12]
#set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCN0533 } [get_ports {swt[13] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
#set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVCN0533 } [get_ports {swt[14] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
#set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCN0533 } [get_ports {swt[15] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]

## LEDs
set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCN0533 } [get_ports {led[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCN0533 } [get_ports {led[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCN0533 } [get_ports {led[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCN0533 } [get_ports {led[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCN0533 } [get_ports {led[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCN0533 } [get_ports {led[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCN0533 } [get_ports {led[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCN0533 } [get_ports {led[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
#set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCN0533 } [get_ports {led[8] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
```

图 8 编辑 Nexys4 DDR 约束文件

1-3-3. 将 *sw* [*]名称更改为 *swt* [*],将 *LED* [*]更改为 *led* [*],因为模型中的端口名称是 *swt*

和 led。

1-3-4. 关闭并保存文件。

1-4. 对源文件执行 RTL 分析

1-4-1. 展开 *Flow Navigator* 窗格的 *RTL Analysis* 任务下的 *Open Elaborated Design* 条目，然后单击 **Schematic**。

1-4-2. 如 Figure9，单击 **OK** 以运行分析。

将详细说明模型（设计）并显示设计的逻辑视图。

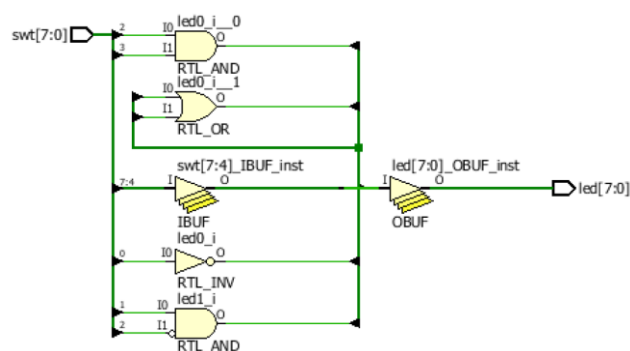


图 9 设计的逻辑视图

请注意，某些开关输入会通过逻辑门后再被输出到 LED，而其余部分将和文件中的模型一样直接输出到 LED。

1-5. I/O 约束（注：与添加 xdc 文件同等作用）

1-5-1. 如 Figure 10，执行 RTL 分析后，可以使用另一种称为 *I/O Planning* 的标准布局。

单击下拉按钮，然后选择 *I/O Planning* 布局。

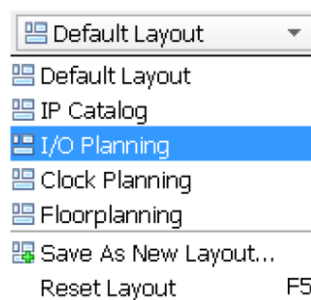


图 10 I/O Planning 布局选项

如 Figure11 ,请注意 ,Package 视图显示在 Auxiliary View 区域中 ,选择了 RTL Netlist 选项卡 , I / O 端口选项卡显示在 Console View 区域中。另请注意 , I / O Ports 选项卡中列出了设计端口 (led 和 swt) , 两者都具有多个 I / O 标准。

将鼠标光标移动到 Package 视图上 ,突出显示不同的引脚。请注意 , 引脚编号以及引脚类型 (用户 IO , GND , VCCO ...) 及其所属的 I / O bank 显示在 Vivado GUI 的底部。

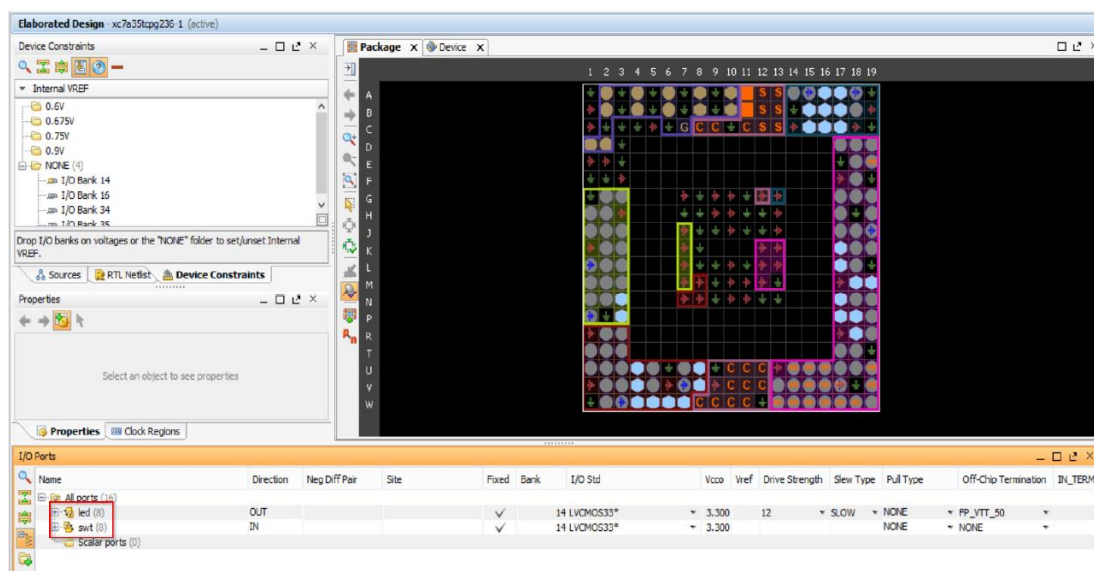


图 11 Basys3 IO Planning 布局视图

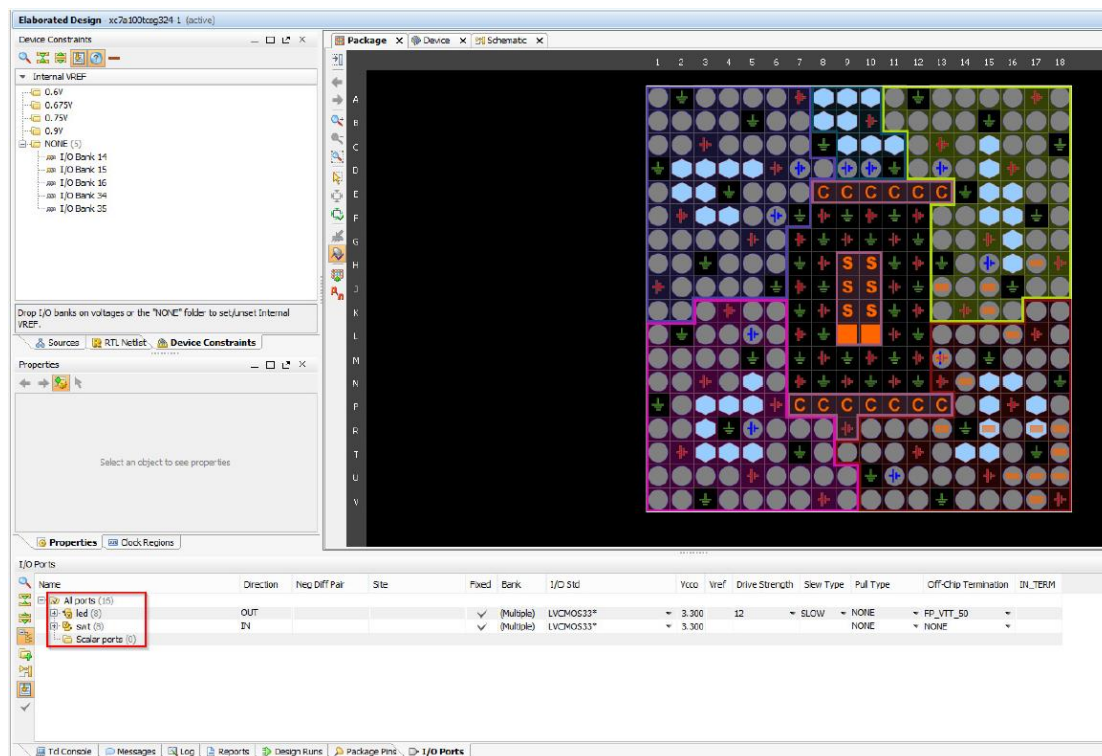


图 11 Nexys4 DDR IO Planning 布局视图

如 Figure12，您可以通过单击+框来展开 LED 和 swt 端口，并观察 led [7 : 0]和 swt [7 : 0]已分配引脚并使用 LVCMOS33 I / O 标准。要更改 I / O 标准，请单击所需端口的 I / OStd，然后选择适当的 I / O 标准。在步骤 1-3-3 中编辑文件时，主 XDC 文件已具有正确的 I / O 标准。

Name	Direction	Neg Diff Pair	Site	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM
All ports (16)													
led (8)													
led[7]	OUT		V14	✓		14 LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50		
led[6]	OUT		U14	✓		14 LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50		
led[5]	OUT		U15	✓		14 LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50		
led[4]	OUT		W18	✓		14 LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50		
led[3]	OUT		V19	✓		14 LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50		
led[2]	OUT		U19	✓		14 LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50		
led[1]	OUT		E19	✓		14 LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50		
led[0]	OUT		U16	✓		14 LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50		
swt (8)													
swt[7]	IN		W13	✓		14 LVCMOS33*	3.300			NONE	NONE		
swt[6]	IN		W14	✓		14 LVCMOS33*	3.300			NONE	NONE		
swt[5]	IN		V15	✓		14 LVCMOS33*	3.300			NONE	NONE		
swt[4]	IN		W15	✓		14 LVCMOS33*	3.300			NONE	NONE		
swt[3]	IN		W17	✓		14 LVCMOS33*	3.300			NONE	NONE		
swt[2]	IN		W16	✓		14 LVCMOS33*	3.300			NONE	NONE		
swt[1]	IN		V16	✓		14 LVCMOS33*	3.300			NONE	NONE		
swt[0]	IN		V17	✓		14 LVCMOS33*	3.300			NONE	NONE		
Scalar ports (0)													

图 12 Basys3 IO 端口映射表

Name	Direction	Neg Diff Pair	Site	Fixed	Bank	I/O Std
All ports (16)						
led (8)	OUT			✓		(Multiple) LVCMOS33*
led[7]	OUT		U16	✓		14 LVCMOS33*
led[6]	OUT		U17	✓		14 LVCMOS33*
led[5]	OUT		V17	✓		14 LVCMOS33*
led[4]	OUT		R18	✓		14 LVCMOS33*
led[3]	OUT		N14	✓		14 LVCMOS33*
led[2]	OUT		J13	✓		15 LVCMOS33*
led[1]	OUT		K15	✓		15 LVCMOS33*
led[0]	OUT		H17	✓		15 LVCMOS33*
swt (8)	IN			✓		(Multiple) LVCMOS33*
swt[7]	IN		R13	✓		14 LVCMOS33*
swt[6]	IN		U18	✓		14 LVCMOS33*
swt[5]	IN		T18	✓		14 LVCMOS33*
swt[4]	IN		R17	✓		14 LVCMOS33*
swt[3]	IN		R15	✓		14 LVCMOS33*
swt[2]	IN		M13	✓		14 LVCMOS33*
swt[1]	IN		L16	✓		14 LVCMOS33*
swt[0]	IN		J15	✓		15 LVCMOS33*
Scalar ports (0)						

图 12 Nexys4 DDR IO 端口映射表

如 Figure13，这些端口已分配引脚。如果要在此视图中分配引脚，则可以在所需端

口行的 *Site* 列下单击以显示下拉框。输入相应的 **Port Variable** 以跳转到具有该变量

的引脚。向下滚动，直到看到正确的端口名称，然后选择它并按 **Enter** 键分配引脚。

Name	Direction	Neg Diff...	Site	Fixed	Bank	I/O Std
All ports (16)						
led (8)	OUT			✓		(Mu... LVCMOS33*
led[7]	OUT		U1	✓		14 LVCMOS33*
led[6]	OUT		U1	✓		14 LVCMOS33*
led[5]	OUT		U11	✓		14 LVCMOS33*
led[4]	OUT		U12	✓		14 LVCMOS33*
led[3]	OUT		U13	✓		14 LVCMOS33*
led[2]	OUT		U13	✓		15 LVCMOS33*
led[1]	OUT		U14	✓		15 LVCMOS33*
led[0]	OUT		U17	✓		15 LVCMOS33*
swt (8)	IN		U18	✓		(Mu... LVCMOS33*

图 13 分配引脚

你还可以通过在 I / O ports 选项卡中选择其条目并将其拖动到 Package 视图并将其

放置在所需位置来分配引脚

如 Figure14，你还可以通过在 I/O Ports 选项卡中选择其条目来分配 I/O 标准，选择 I/O Port 属性窗口的配置选项卡，然后单击 I/O 标准字段的下拉按钮，选择 LVCMOS33。

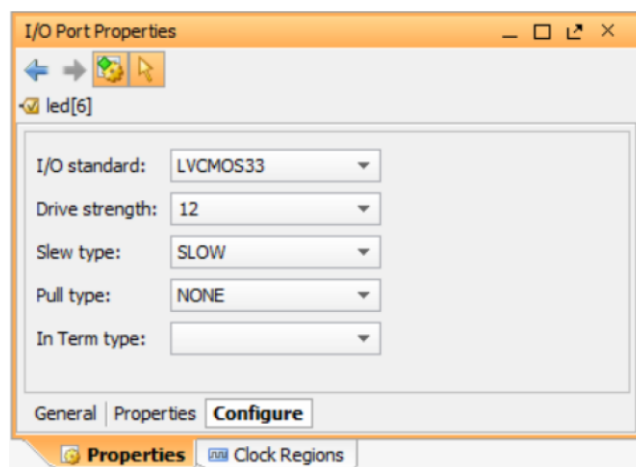


图 14 通过 IO 属性表设定 IO 标准

你还可以使用 tcl 命令分配引脚约束和 I/O 标准，方法是在 Tcl 控制台选项卡中键入命令以分配 R13 引脚位置和 LVCMOS33 I/O (如下所示) 并在命令后按 Enter 键。

```
set_property -dict { PACKAGE_PIN R13    IOSTANDARD LVCMOS33 } [get_ports { swt[7] }];
```

使用 XSim 仿真器仿真 Vivado 项目

Step 2

2-1. 添加 tutorial_tb.v testbench 文件。

2-1-1. 如 Figure15，单击 *Flow Navigator* 窗格的 *Project Manager* 任务下的 **Add Sources**。

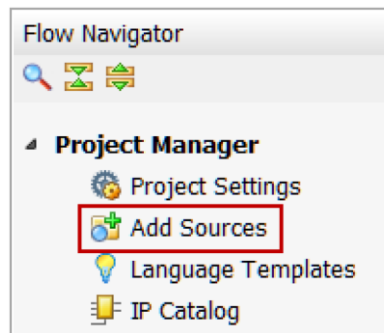


图 15 添加源文件

2-1-2. 如 Figure 16，选择 *Add or Create Simulation* 选项，然后单击 **Next**

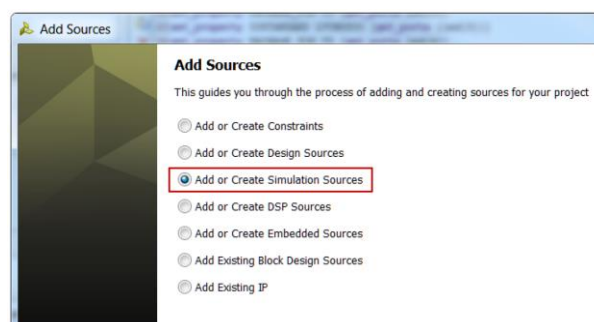


图 16 选择添加或创建仿真源文件选项

2-1-3. 在 *Add Sources Files* 表单中，单击 **Green Plus** 按钮，然后单击 **Add Files...**按钮

2-1-4. 浏览到 `c : \xup \digital \sources \tutorial` 文件夹并选择 `tutorial_tb.v` 并单击 **OK**

2-1-5. 点击 **Finish**

2-1-6. 如 Figure17，选择 *Sources* 选项卡，然后展开 *Simulation Sources* 组。

tutorial_tb.v 文件添加在 *Simulation Sources* 组下，`tutorial.v` 自动放在其层次结构中作为 tut1 实例。

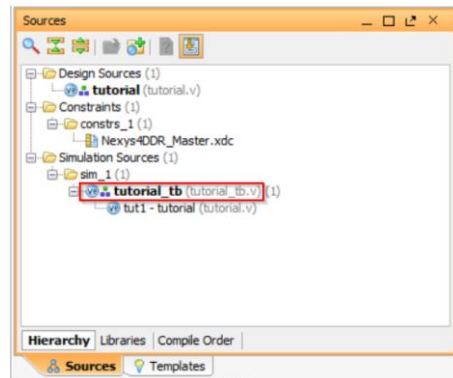


图 17 仿真文件层次结构

2-1-7. 使用 Windows 资源管理器，验证是否在 tutorial.srcs 目录下的 constrs_1 和 sources_1

目录创建了 sim_1 目录，并且在 tutorial.srcs> sim_1> imports> sources 下放置了 tutorial_tb.v 的副本。

2-1-8. 如 Figure18 双击 Sources 窗格中的 tutorial_tb 以查看其内容

```
1 `timescale 1ns / 1ps
2 //////////////////////////////////////////////////
3 // Module Name: tutorial_tb
4 //////////////////////////////////////////////////
5 module tutorial_tb(
6
7 );
8
9     reg [7:0] switches;
10    wire [7:0] leds;
11    reg [7:0] e_led;
12
13    integer i;
14
15    tutorial tut1(.led(leds),.swt(switches));
16
17    function [7:0] expected_led;
18        input [7:0] swt;
19    begin
20        expected_led[0] = ~swt[0];
21        expected_led[1] = swt[1] & ~swt[2];
22        expected_led[3] = swt[2] & swt[3];
23        expected_led[2] = expected_led[1] | expected_led[3];
24        expected_led[7:4] = swt[7:4];
25    end
26    endfunction
27
28    initial
29    begin
30        for (i=0; i < 255; i=i+2)
31        begin
32            #50 switches=i;
33            #10 e_led = expected_led(switches);
34            if(leds == e_led)
35                $display("LED output matched at", $time);
36            else
37                $display("LED output mis-matched at ", $time, ": expected: %b, actual: %b", e_led, leds);
38            end
39        end
40    end
41 endmodule
```

图 18 带自检工程的测试平台

测试平台在第一行定义了仿真步长和分辨率。测试平台模块定义从第 5 行开始。第 15 行实例化 DUT (被测设备/模块)。第 17 行到第 26 行，为期望值计算定义了相同的模块功能。第 28 到 39 行定义了刺激生成，并将预期输出与 DUP 提供的内容进行比较。第 41 行结束测试平台。当仿真正拉起进行的时候，\$ display 任务将在仿真运行时在仿真器控制台窗口中打印消息。

2-2. 使用 XSim 仿真器仿真设计 (200 ns)

2-2-1. 在 *Flow Navigator* 窗格的 *Project Manager* 任务下选择 *Simulation Settings*。出现 **A Project Settings** 表单，其中显示 **Simulation** 表单

2-2-2. 选择 **Simulation** 选项卡，将 **Simulation Run Time** 值设置为 200 ns，然后单击 **OK**。

2-2-3. 单击 *Flow Navigator* 窗格下的 **Run Simulation> Run Behavioral Simulation**。

将编译测试平台和源文件，并运行 XSim 仿真器 (假设没有错误)。您将看到类似于下图 (Figure 19) 所示的仿真器输出。

您将看到四个主要视图：(i) *Scopes*，其中显示测试平台层次结构以及 *glbl* 实例，(ii) *Objects*，显示顶级信号的对象，(iii) 波形窗口 (iv) *Tcl Console* 显示仿真活动的位置。

请注意，由于使用的测试平台是自检，因此在运行仿真时会显示结果。

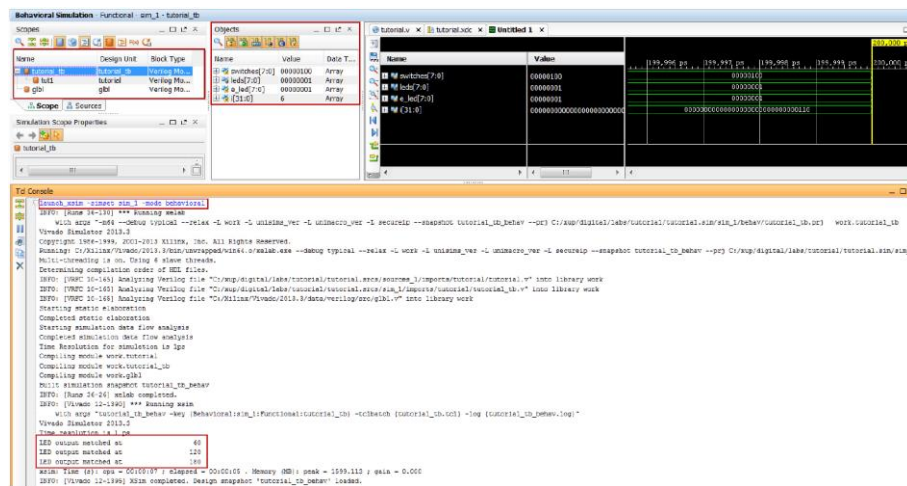


图 19 仿真输出

请注意，**tutorial.sim** 目录是在 **tutorial** 目录下创建的，还有几个较低级别的目录。如

Figure 20

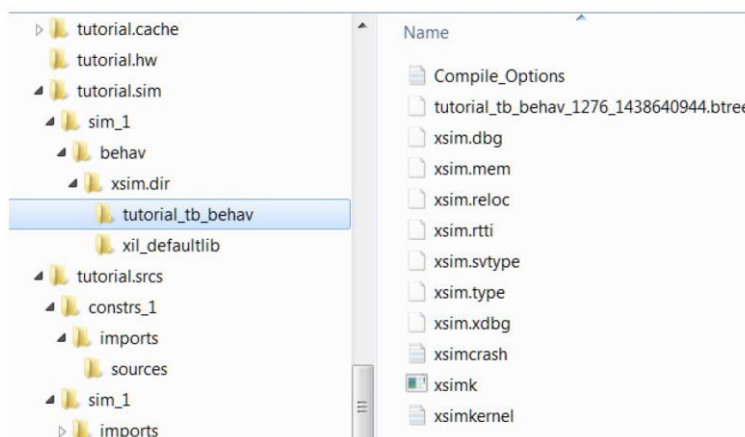


图 20 行为仿真后的目录结构

http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_1/ug900-viva

do-logicsimulation.pdf 学习更多关于 Vivado 仿真器的资料

2-2-4. 单击位于波形窗口左侧 Zoom Fit 按钮 () 以查看整个波形。

请注意，输入更改时输出会发生变化。

你还可以通过单击视图右上角的浮动按钮来使仿真波形窗口成为浮动窗口。这将允许您有

一个更宽的窗口来查看仿真波形。要将浮动窗口重新集成回 GUI ,只需单击 Dock Window 按钮。

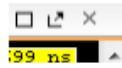


Figure 21. Float Button



Figure 22. Dock Window Button

2-3. 在需要的情况下更改显示格式

2-3-1. 在波形窗口中选择 `i [31 : 0]` , 右键单击 , 选择 *Radix* , 然后在 *Integer* 表单中选择 *Unsigned*

Decimal 来查看 for 循环的索引。同样 , 将 `switches[7 : 0]` 的基数更改为 *Hexadecimal*。

将 `leds [7 : 0]` 和 `e_led [7 : 0]` 基数保留为 *binary* , 因为我们希望看到每个输出位。

2-4. 添加更多信号以监控低电平信号 , 并继续运行仿真 (500 ns)

2-4-1. 如有必要 , 在 *Scopes* 窗口中展开 `tutorial_tb` 实例 , 然后选择 `tut1` 实例。

`swt [7 : 0]` 和 `led [7 : 0]` 信号将显示在 *Objects* 窗口中

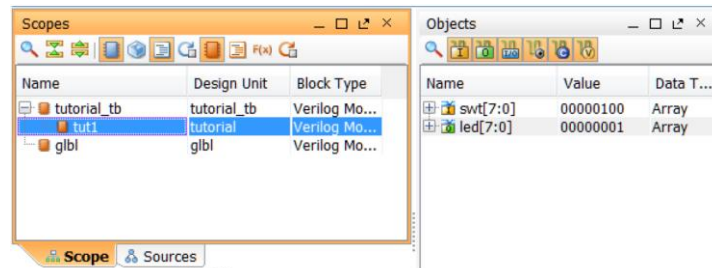


图 23 选择内部信号

2-4-2. 选择 `swt [7 : 0]` 和 `led [7 : 0]` 并将它们拖动到波形窗口中以监视这些低电平信号。您也可以

以通过右键单击并选择 **Add to Wave Window** 来添加信号。

2-4-3. 在仿真器工具按钮功能区栏上 , 在时间窗口中键入 500 , 单击单位字段的下拉按钮并选

择 ns , 然后单击按钮。

仿真将再运行 500 ns。

2-4-4. 单击 *Zoom Fit* 按钮并观察输出

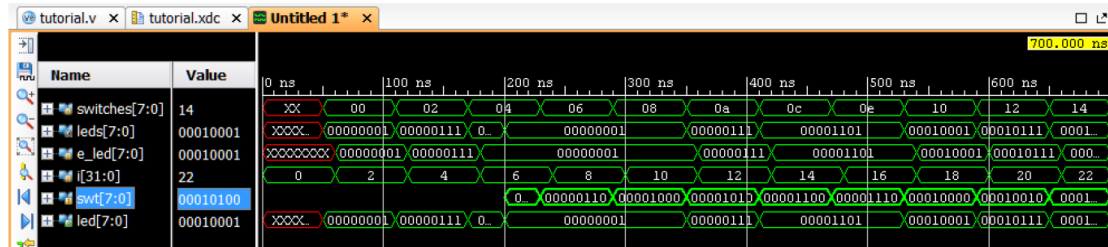


图 24 继续仿真 500ns

2-4-5. 选择 **File> Close Simulation** 来关闭仿真器。

2-4-6. 单击 **OK**，然后单击 **Discard** 以关闭它而不保存波形。

综合你的设计

Step 3

3-1. 使用 Vivado 综合工具合成设计并分析项目输出。


3-1-1. 单击 *Flow Navigator* 窗格的 *Synthesis* 任务下的 **Run Synthesis**。

合成过程将在 tutorial.v 文件 (及其所有分层文件，如果存在) 上运行。完成此过程后，

将显示包含三个选项的 *Synthesis Completed* 对话框。

3-1-2. 选择 *Open Synthesized Design* 选项并单击 **OK**，因为我们想要在进入实现阶段之前查看综合输出。

如果对话框已经显示，请单击 **Yes** 以关闭详细设计。

3-1-3. 选择 **Project Summary** 选项卡  (如果选项卡不可见，请选择默认布局) 并了解各种窗口。

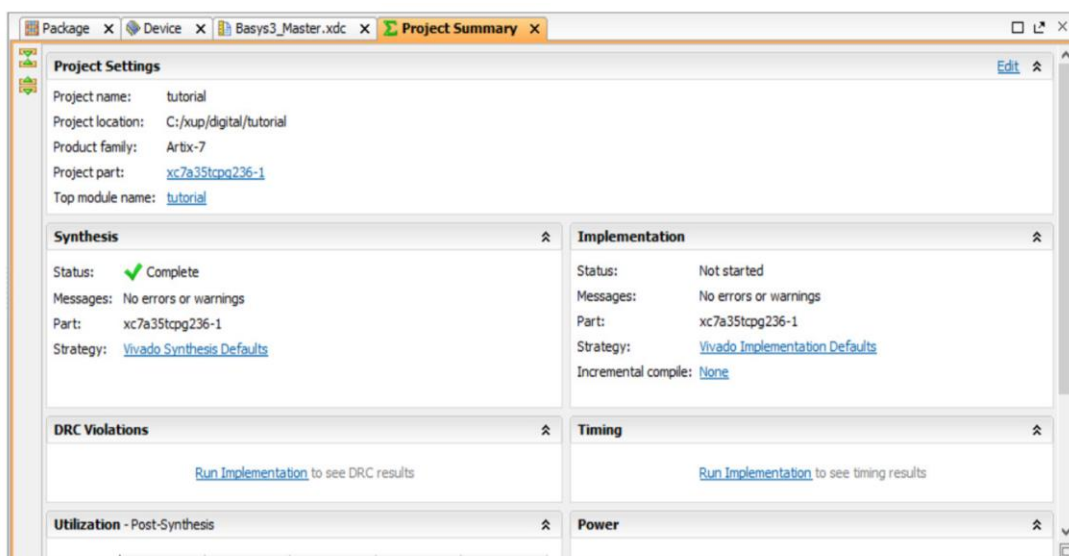


Figure 25. Project Summary view for Basys3

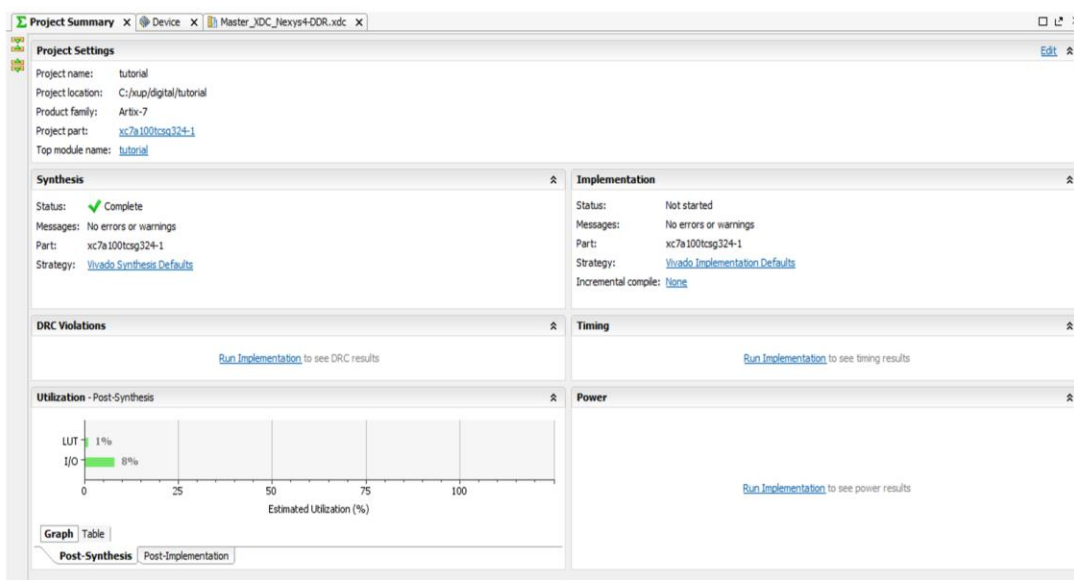


Figure 25. Project Summary view for Nexys4 DDR

单击各种链接以查看它们提供的信息，以及允许您更改综合设置的信息。

3-1-4. 单击 Project Summary 选项卡中的 Table 选项卡。

请注意，使用了三个 LUT 和 16 个 IO (8 个输入和 8 个输出)。

Resource	Estimation	Available	Utilization %
LUT	3	63400	1
I/O	16	210	8

Figure 26. Resource utilization estimation summary

3-1-5. 单击 *Flow Navigator* 窗格的 *Synthesis* 任务的 *Open Synthesized Design* 任务下的 *Schematic*，在原理图视图中查看合成设计。

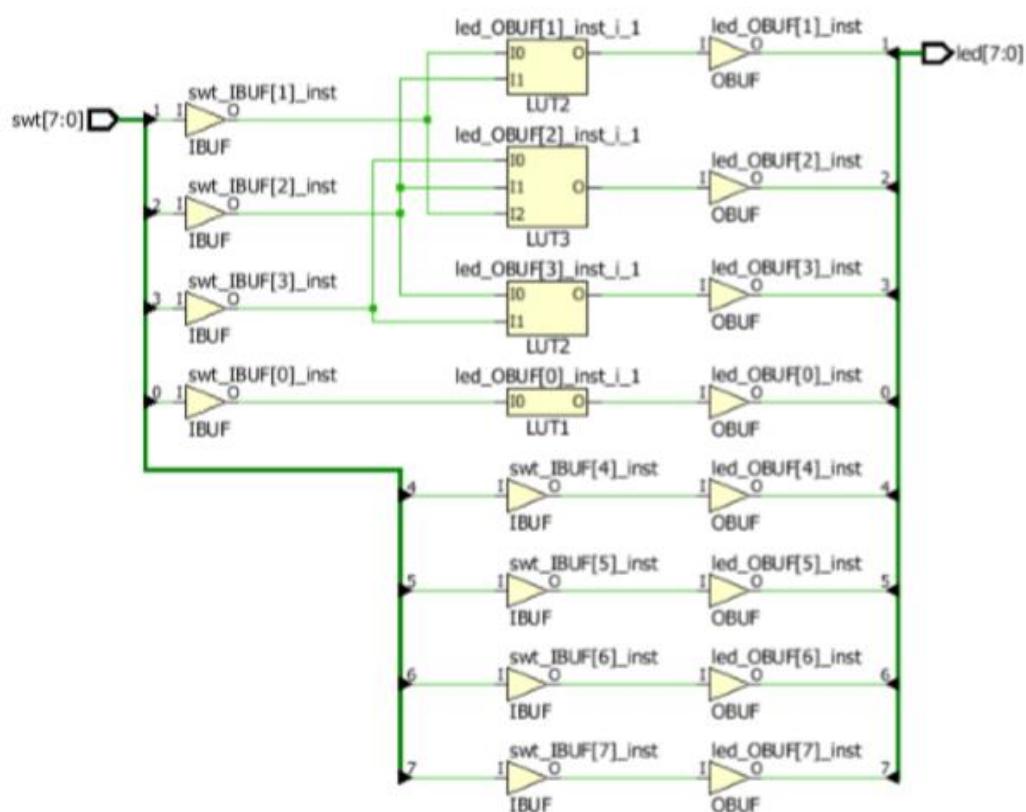


Figure 27. Synthesized design's schematic view

请注意，当输入和输出被缓冲时，IBUF 和 OBUF 会自动实例化（添加）到设计中。逻辑

门在 LUT 中实现（1 个输入列为 LUT1, 2 个输入列为 LUT2, 3 个输入列为 LUT3）。RTL 分析输出中的四个门映射到合成输出中的四个 LUT。

使用 Windows 资源管理器，确认是否在 **tutorial** 下创建了 **tutorial.runs** 目录。在 **runs** 目录下，创建了 **synth_1** 目录，其中包含多个临时子目录。

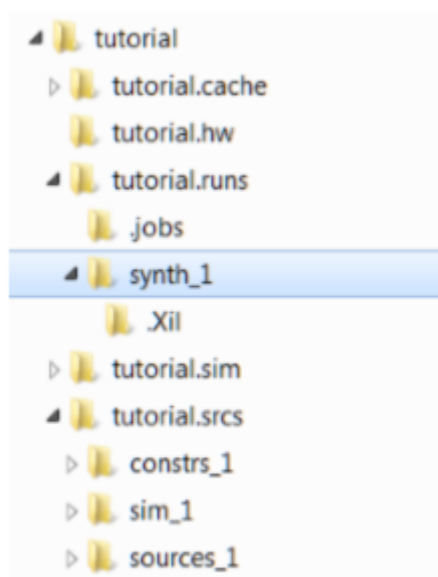


Figure 28. Directory structure after synthesizing the design

实现你的设计

Step4

4-1. 使用 Vivado Defaults Implementation 设置实现设计并分析

Project Summary 输出

4-1-1. 单击 *Flow Navigator* 窗格的 *Implementation* 任务下的 **Run Implementation**。

实现过程将在综合输出文件上运行。完成此过程后，将显示包含三个选项的 *Implementation Completed* 对话框。

4-1-2. 当我们要在 Device 视图选项卡中查看已实现的设计。选择 **Open implemented design** 并单击 **OK**

4-1-3. 单击 **Yes** 以关闭合成设计。

已实现的设计将被打开

4-1-4. 在 *Netlist* 窗格中，选择其中一个网络（例如 `led_OBUF [1]`）并注意在 Device 视图选项

卡中为 Basys3 的 X0Y0 时钟区域和 Nexys4 DDR 的 X0Y1 和 X0Y2 时钟区域显示的网络

（您可能必须放大看看）。

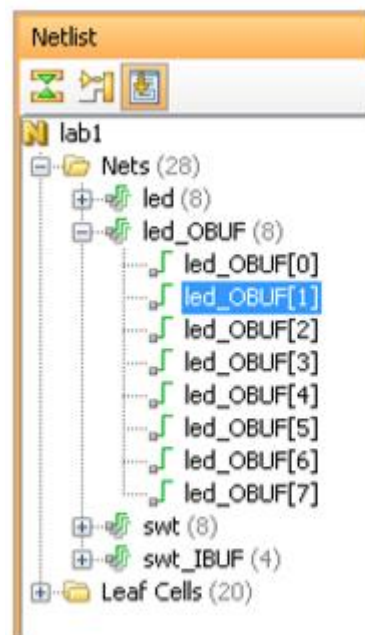


Figure 29. Selecting a net

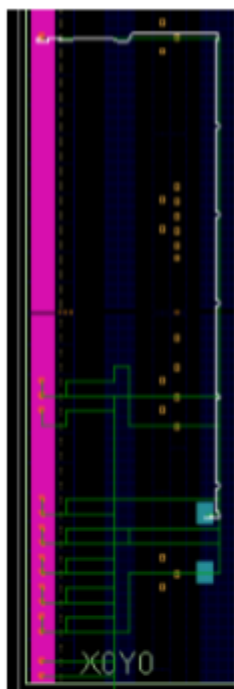


Figure 29. Viewing implemented design for Basys3

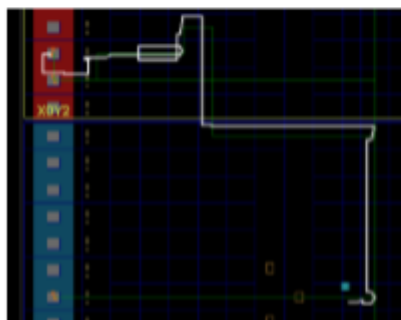


Figure 29. Viewing implemented design for Nexys4 DDR

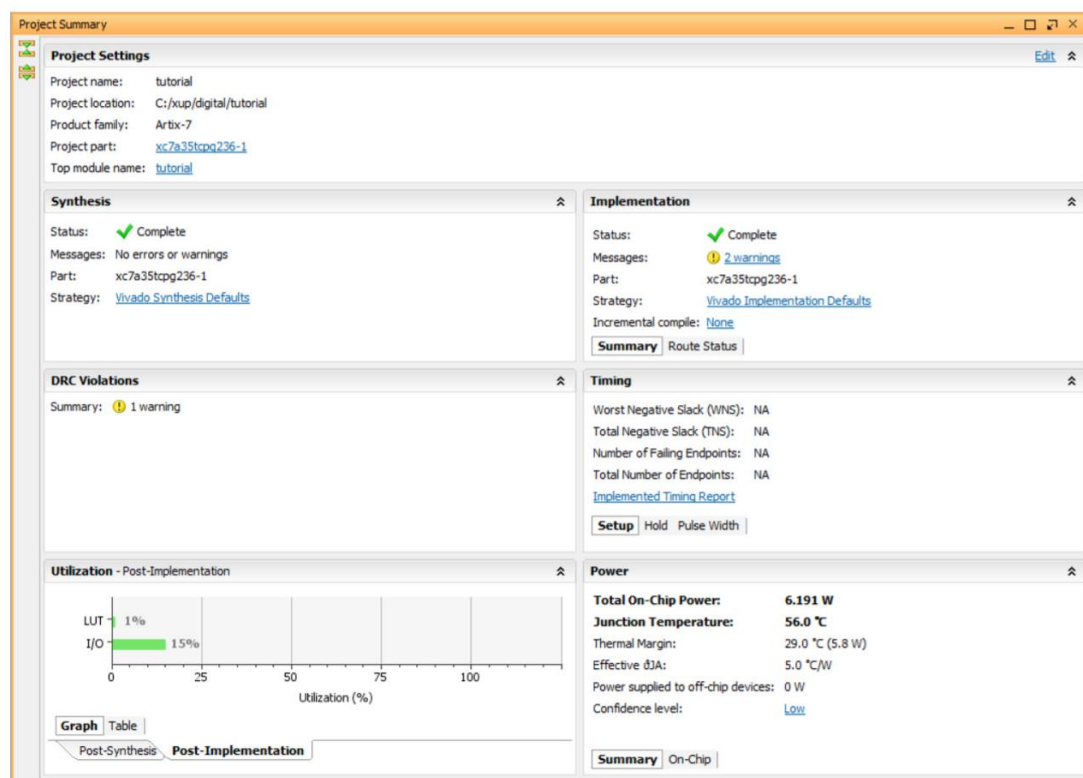
4-1-5. 选择 **File> Close Implemented Design** 关闭已实现的设计视图，然后选择 **Project**

Summary 选项卡（您可能必须更改为 Default Layout 视图）并观察结果。

请注意，实际资源利用率是 3 个 LUT 和 16 个 IO。此外，它表明没有为此设计定义时

序约束（因为该设计是组合逻辑）。在 *Timing and Utilization* 窗口下选择

Post-implementation 选项卡。



使用 Windows 资源管理器，确认 `impl_1` 目录是否被创建到与 `tutorial_runs` 目录下的

`synth_1` 的相同级别。`impl_1` 目录包含多个文件，包括报告文件。

4-1-6. 在 Flow Navigator 下的 *Synthesized Design* 下选择 *Report Utilization* 条目。该报告将在

辅助视图窗格中显示资源利用率。请注意，由于设计是组合的，因此不需要任何寄存器。

执行定时仿真

Step5

5-1. 执行一次定时仿真

5-1-1. 在 *Flow Navigator* 窗格的 *Simulation* 任务下选择 **Run Simulation> Run**

Post-Implementation Timing Simulation 过程。

XSim 仿真器将使用实现的设计和 `tutorial_tb` 作为顶层模块启动。


使用 Windows 资源管理器 验证是否在 `tutorial.sim>sim_1>impl` 目录下创建了 `timing`

目录。`timing` 目录包含生成的文件以运行时序仿真

5-1-2. 单击 **Zoom Fit** 按钮可查看 0 到 200 ns 的波形窗口。

5-1-3. 右键单击 50 ns (开关输入设置为 0000000b) 并选择 **Markers>Add Marker**

5-1-4. 同样，右键单击并在 `leds` 变化的 55.000 ns 左右添加一个标记。

5-1-5. 您还可以通过单击 Add Marker 按钮()添加标记。单击 **AddMarker** 按钮 ,然后在 `e_led` 更改的大约 60 ns 处单击鼠标左键。

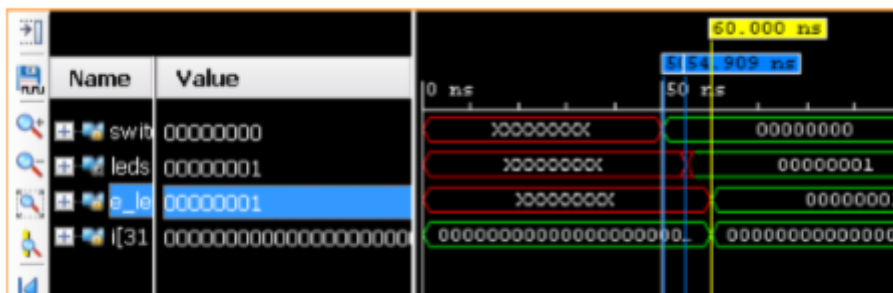


Figure 31. Timing simulation output

请注意，我们在输入更改后的 10 ns 处监视预期的 LED 输出 (请参阅测试平台)，而实际延迟大约为 5.000 ns

5-1-6. 选择 **File > Close Simulation** 关闭仿真器，不保存任何更改。

生成 Bistream 并验证功能

Step6

6-1. 确保电源跳线设置为 USB，连接电路板，打开电源。生成 Bitstream，打开硬件会话，对 FPGA 进行编程。

6-1-1. 确保电源设备跳线设置为 USB，并且提供的 Micro-USB 电缆连接在主板和 PC 之间。请

注意，您无需连接电源插孔，只需通过 USB 即可为电路板供电和配置



Figure 32. Board settings for Nexys4 DDR



Figure 32. Boards settings for Basys3

6-1-2. 打开电路板上的开关

6-1-3. 单击 *Flow Navigator* 窗格的 *Program and Debug* 任务下的 **Generate Bitstream** 条目。

比特流生成过程将在实现的设计上运行。完成此过程后，将显示包含三个选项的 *Bitstream Generation Completed* 对话框。

此过程将在 `impl_1` 目录下生成 `tutorial.bit` 文件，该文件是在 `tutorial.runs` 目录下生成的。

6-1-4. 选择 *Open Hardware Manager* 选项，然后单击 **OK**。

将打开“Hardware Manager”窗口，指示“unconnected”状态。

6-1-5. 单击 **Open Target** 链接

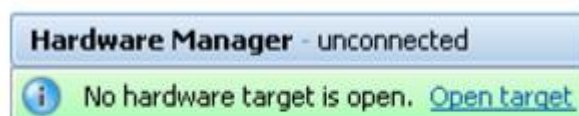


Figure 33. Opening new hardware target

6-1-6. 从下拉菜单中，单击 **Auto Connect**

硬件会话状态从 `unconnected` 更改为服务器名称，并突出显示设备。另请注意，Status 显示为未编程

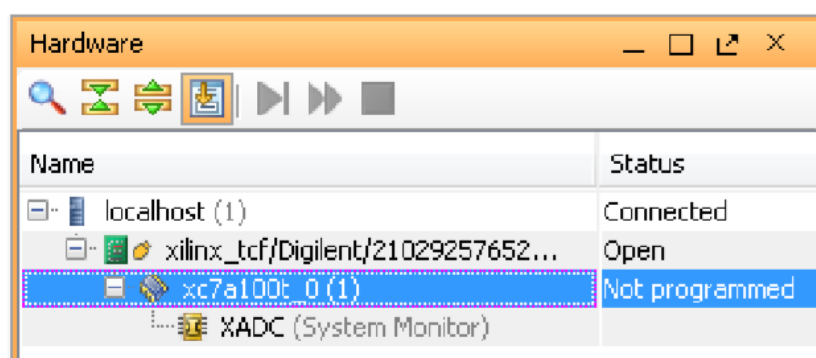


Figure 34. Opened hardware session for the Nexys4 DDR

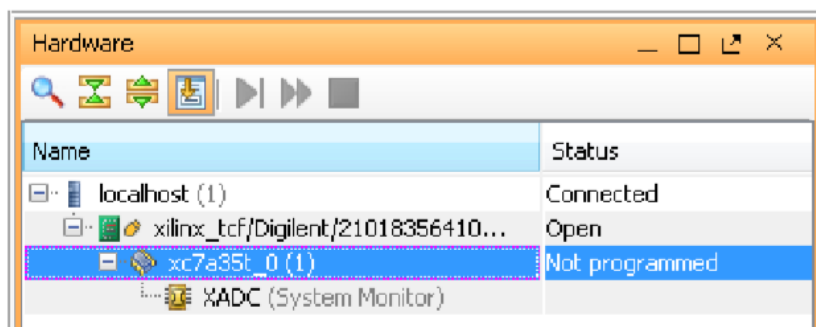


Figure 34. Opened hardware session for the Basys3

6-1-7. 选择设备并验证是否在 General 选项卡中选择了 lab1.bit 作为编程文件

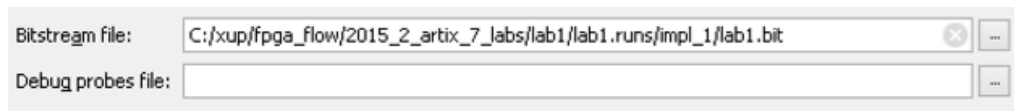


Figure 35. Programming file

6-1-8. 单击绿色信息栏中的 *Program device* > *XC7A100T_0* 或 *XC7A35T_0* 链接，对目标 FPGA 器件进行编程。

另一种方法是右键单击设备并选择 *Program Device...*

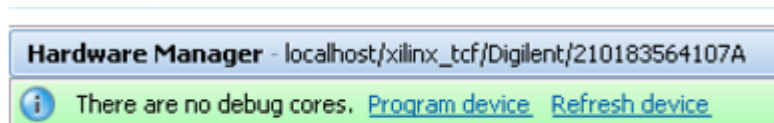


Figure 36. Selecting to program the FPGA

6-1-9. 单击 **Program** 以对 FPGA 进行编程。

设备编程后，DONE 灯将亮起。根据开关位置，您可能会看到 其他一些 LED 点亮。

6-1-10. 通过拨动开关并观察 LED 上的输出来验证功能。

6-1-11. 选择 **File > Close Hardware Manager**。 关闭硬件会话。

6-1-12. 单击 **OK** 关闭会话。

6-1-13. 关闭硬件电源

6-1-14. 选择 **File > Exit** 关闭 Vivado 程序，然后单击 **OK**。

总结：

Vivado 软件工具可用于执行完整的设计流程。该项目是使用提供的源文件（HDL 模型和用户约束文件）创建的。先进行行为仿真以验证模型功能。然后对设计进行合成，并实现该模型，生成比特流，使用相同的测试平台在实现的设计上运行时序仿真。最后使用生成的比特流在硬件中验证功能。