**数字逻辑实验手册**

**（基于DIGILENT BASYS3口袋实验室）**

**吴志红 潘薇 卢晓春 主编**

**四川大学计算机学院**

**2020年**

目录

[第一部分 实验平台介绍 1](#_Toc530480415)

[1.熟悉Basys3实验板 1](#_Toc530480416)

[1.1 注意事项 1](#_Toc530480417)

[1.2 使用介绍 2](#_Toc530480418)

[2.熟悉Vivado平台 7](#_Toc530480419)

[2.1 开发环境介绍 7](#_Toc530480420)

[2.2 基于模块化的设计流程 7](#_Toc530480421)

[第二部分数字逻辑实验 19](#_Toc530480422)

[3.实现布尔表达式 19](#_Toc530480423)

[3.1 实验要求 19](#_Toc530480424)

[3.2 实验步骤 19](#_Toc530480425)

[4.加法器 22](#_Toc530480426)

[4.1实验要求 22](#_Toc530480427)

[4.2 实验步骤 22](#_Toc530480428)

[5.译码器和编码器 25](#_Toc530480429)

[5.1 实验要求 25](#_Toc530480430)

[5.2 实验步骤 25](#_Toc530480431)

[6.多路选择器 28](#_Toc530480432)

[6.1 实验要求 28](#_Toc530480433)

[6.2 实验步骤 28](#_Toc530480434)

[7.寄存器 30](#_Toc530480435)

[7.1实验要求 30](#_Toc530480436)

[7.2实验步骤 30](#_Toc530480437)

[8.计数器 36](#_Toc530480438)

[8.1 实验要求 36](#_Toc530480439)

[8.2 实验步骤 36](#_Toc530480440)

[9.流水灯 38](#_Toc530480441)

[9.1实验要求 38](#_Toc530480442)

[9.2实验步骤 38](#_Toc530480443)

[附录1：关于Vivado中的引脚约束 40](#_Toc530480444)

[附录2：了解Verilog语言 41](#_Toc530480445)

# 第一部分 实验平台介绍

## 1.熟悉Basys3实验板

Basys3是围绕着一个Xilinx Artix®-7 FPGA芯片XC7A35T-1CPG236C搭建的，它为学习FPGA和数字电路设计的用户提供了一个理想、随时可以使用的电子设计硬件平台。

Basys3的关键特性：

* 33280个逻辑单元，六输入LUT结构
* 1800Kbits快速RAM块
* 5个时钟管理单元，均各含一个锁相环（PLL）
* 90个DSP slices
* 内部时钟最高可达450MHz
* 1个片上模数转换器（XADC）

Basys3板提供完整的硬件存取电路，可以完成从基本逻辑到复杂控制器的设计。四个标准扩展连接器配合用户设计的电路板，或Pmods（Digilent设计的A/D和D/A转换，电机驱动器，传感器输入等）其他功能，扩展信号的8针接口均采用ESD保护。附带的USB电缆，提供电源和编程接口，不需要额外的配置电源或其他编程电缆，是一个入门或复杂数字电路系统设计的完美低成本平台。

### 1.1 注意事项

1）Basys 3只接受5V直流输入，通过USB供电（可以通过EXT供电，但实验时不用次方式供电。EXT供电方式后面会介绍）；

2）Basys 3上电后，不可带电插拔USB线，不得动跳线开关；

3）Basys 3须远离水源；

4）取用实验板时，请卡住实验板边缘，尽量不要接触芯片管脚及导线，尤其是在上电时；

5）归还实验板时，请检查USB电缆是否放入盒子内。

### 1.2 使用介绍

#### 1.2.1概述

Basys3板上集成了大量的 I/O设备和FPGA所需的支持电路，让您能够构建无数的设计而不需要其他器件。

实验板正面如图1-1所示。

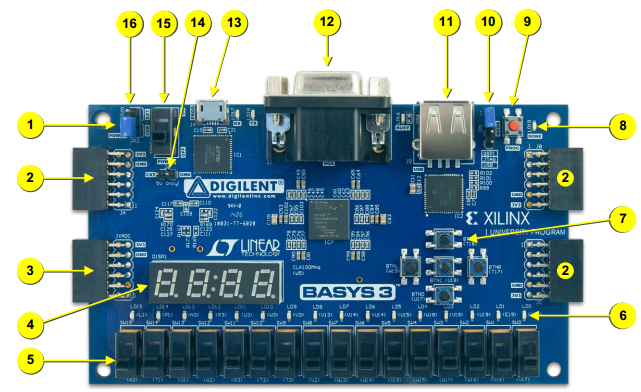
图1-1 实验板顶视图

图1-1的各标号对应的I/O设备如表1-1所示。

表1-1 实验板外设标号

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **描述** | **序号** | **描述** |
| 1 | 电源指示灯 | 9 | FPGA配置复位按键 |
| 2 | Pmod数字信号接口 | 10 | 编程模式跳线柱 |
| 3 | Pmod模拟信号接口 | 11 | USB接口 |
| 4 | 4位7段数码管 | 12 | VGA显示接口 |
| 5 | 16个拨码开关 | 13 | UART/JTAG共用USB接口 |
| 6 | 16个LED灯 | 14 | 外部电源接口 |
| 7 | 5个按键开关 | 15 | 电源开关 |
| 8 | FPGA编程指示灯 | 16 | 电源选择跳线柱 |

#### 1.2.2 供电方式

Basys3板可以通过2种方式进行供电，一种是通过J4的USB端口供电，另一种是通过J6的接线柱进行外部供电（5V）。通过JP2跳线帽（图1-1的16，表1-1进行了说明）的不同选择进行供电方式的选择。电源开关通过SW16进行控制，LD20为电源开关的指示灯。如图1-2所示。

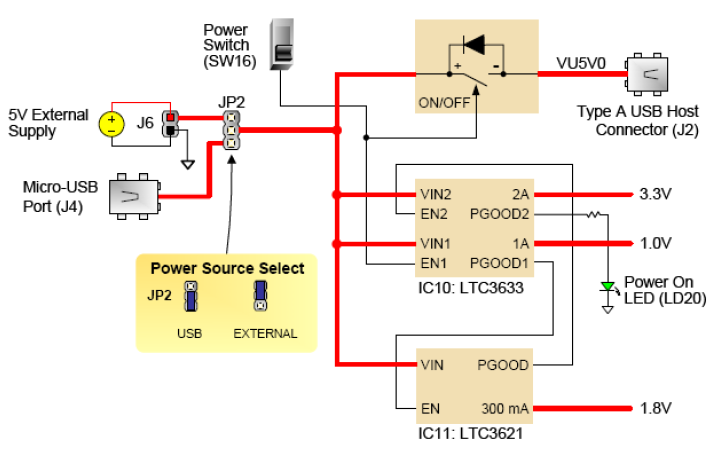


图1-2 USB接口电路

在本实验中，实验板采用USB供电。

#### 1.2.3 LED灯电路

LED灯部分的电路如图1-3所示。当FPGA的输出为高电平时，相连接的LED灯点亮；否则，LED灯熄灭。板上配有16个LED灯，可用作标志显示或结果显示。

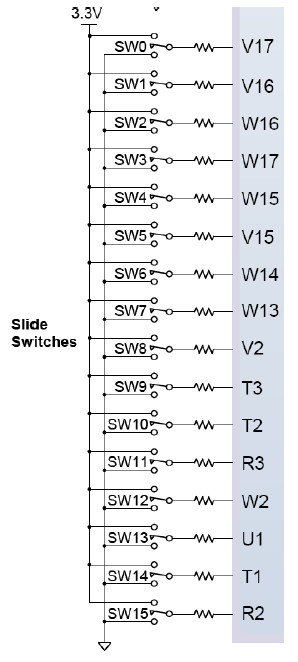
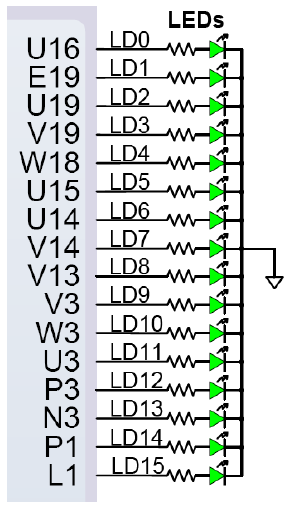


图1-3 LED灯电路图 1-4 拨码开关电路

#### 1.2.4 拨码开关电路

拨码开关电路如图1-4所示。开关打到下档时，表示所连接的FPGA输入为低电平。

#### 1.2.5按键电路

按键部分的电路如图1-5所示。当按键按下时，对应的FPGA输入为高电平。

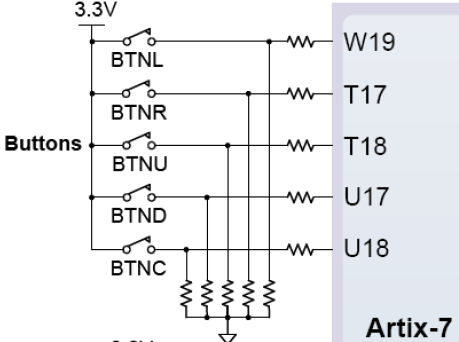


图1-5 按键电路

#### 1.2.6 数码管电路

数码管电路如图1-6所示。4个带小数点的七段共阳极数码管，当连接的FPGA输出为低电平时，该段位的LED点亮。位选通为低电平选通。

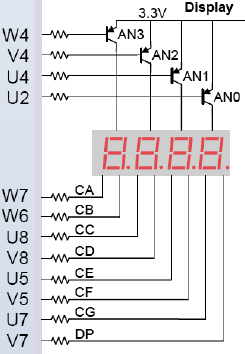


图1-6 数码管电路

#### 1.2.7 上电加载方式

上电后，Basys3板上必须配置FPGA芯片，然后才能执行相关功能。在配置过程中，一个“bit”文件（程序经过编译后的二进制代码）会通过电缆下载到FPGA内存单元中，实现逻辑功能和电路互联。借助赛灵思免费的Vivado软件，可以通过VHDL或Verilog语言源文件，或基于原理图输入，创建.bit文件。

编程加载模式有三种方式，通过JP1的跳线帽进行选择。

* 用Vivado通过QSPI方式下载.bit文件到Flash芯片，实现掉电不易失。板子上电后，FPGA首先从Flash芯片里读.bit文件。
* 用Vivado通过JTAG方式从⑬下载.bit文件到Flash芯片。
* 用U盘或移动硬盘通过J2的USB端口⑪下载.bit文件到FPGA芯片（建议将.bit文件放在U盘根目录下，且只放1个），该U盘应该是FAT32文件系统。

在实验中，始终按图1-7选一配置，如下所示。

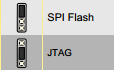


图1-7 上电加载方式选择

#### 1.2.8其他

表1-2 部分引脚对照

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| LED灯 | PIN | 开关 | PIN | 7段码管 | PIN | 按键 | PIN |
| LD0 | U16 | SW0 | V17 | AN0 | U2 | BTNU | T18 |
| LD1 | E19 | SW1 | V16 | AN0 | U4 | BTNR | T18 |
| LD2 | U19 | SW2 | W16 | AN0 | V4 | BTND | U17 |
| LD3 | V19 | SW3 | W17 | AN0 | W4 | BTNL | W19 |
| LD4 | W18 | SW4 | W15 | CA | W7 | BTNC | U18 |
| LD5 | U15 | SW5 | V15 | CB | W6 |  |  |
| LD6 | U14 | SW6 | W14 | CC | U8 | 时钟 | PIN |
| LD7 | V14 | SW7 | W13 | CD | V8 | MRCC | W5 |
| LD8 | V13 | SW8 | V2 | CE | U5 |  |  |
| LD9 | V3 | SW9 | T3 | CF | V5 | USB(J2) | PIN |
| LD10 | W3 | SW10 | T2 | CG | U7 | PS2\_CLK | C17 |
| LD11 | U3 | SW11 | R3 | DP | V7 | PS2\_DAT | B17 |
| LD12 | P3 | SW12 | W2 |  |  |  |  |
| LD13 | N3 | SW13 | U1 |  |  |  |  |
| LD14 | P1 | SW14 | T1 |  |  |  |  |
| LD15 | L1 | SW15 | R2 |  |  |  |  |

VGA显示电路、I/O扩展电路等，需要时查询Basys3的用户手册。

## 2.熟悉Vivado平台

### 2.1 开发环境介绍

Basys3板的FPGA芯片属于Xilinx公司的ATRIX 7系列，其对应的开发环境是Xilinx推出的Vivado设计套件。

### 2.2 基于模块化的设计流程

采用模块化的设计流程，通过调用已经写好的模块设计数字电路，流程如下：

**1、创建新工程**

1）打开Vivado设计开发软件，如图2-1，选择Create New Project.

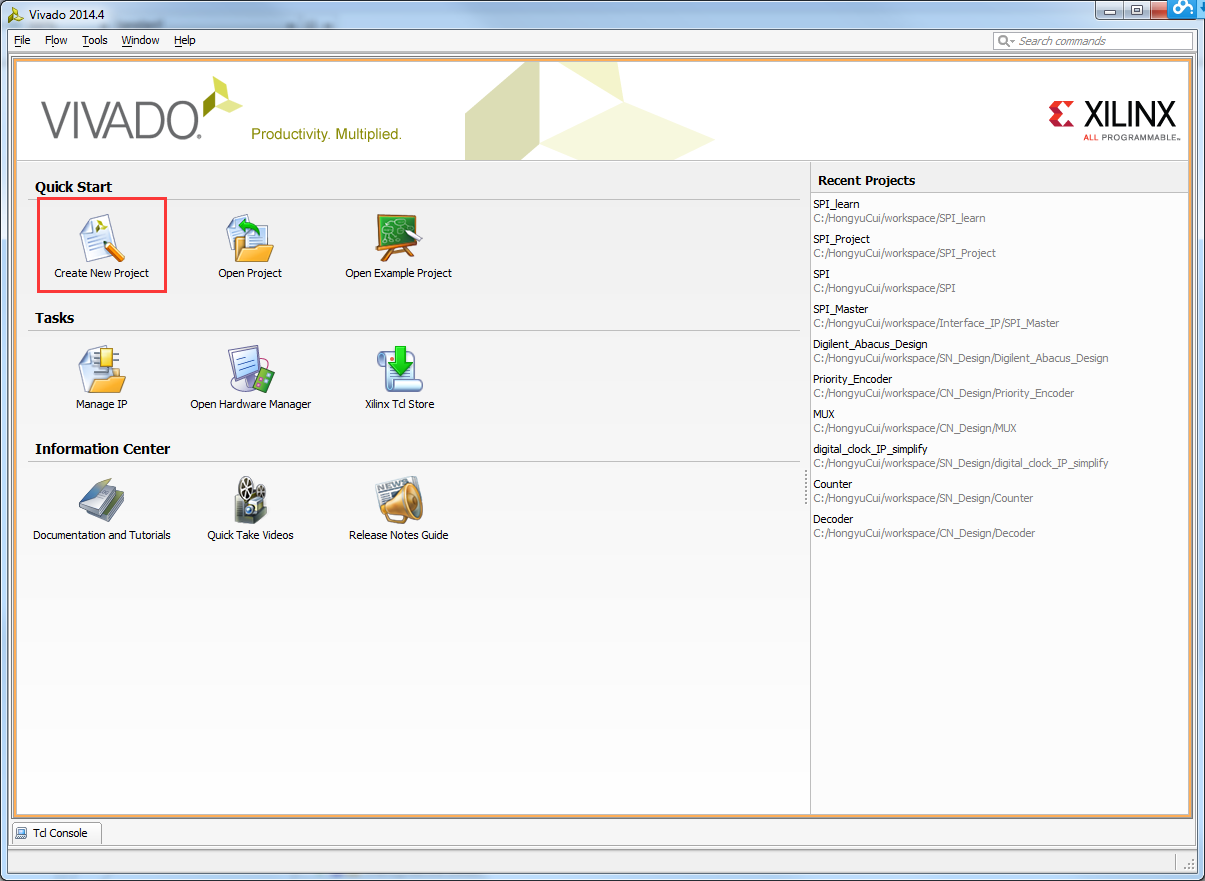


图2-1 Vivado初始界面

2）在弹出的创建新工程的界面中，如图2-2，点击Next，开始创建新工程。

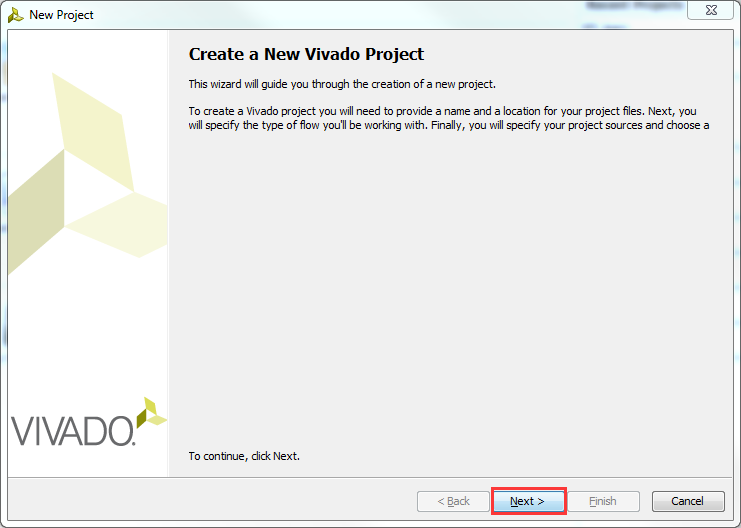
****

图2-2 创建新工程

3）在Project Name界面中，工程名称预置为project\_1（可根据个人需要进行修改），设置好工程存放路径为E:/demo（或其它拟存放工程的路径），勾选创建工程子目录的选项（默认项），如图2-3。这样，整个工程文件都将存放在创建的demo子目录中。点击Next。(注意：路径以及工程名必须是英文！)

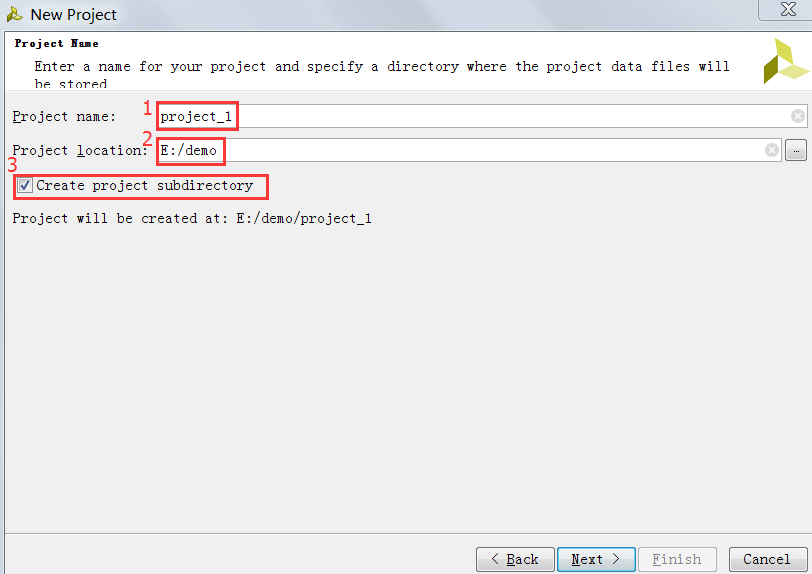


图2-3 命名及设置路径

4）在选择工程类型的界面中，如图2-4，选择RTL工程。由于本工程无需创建源文件，故将Do not specify sources at this time（不指定添加源文件）勾选上（默认项）。点击Next。

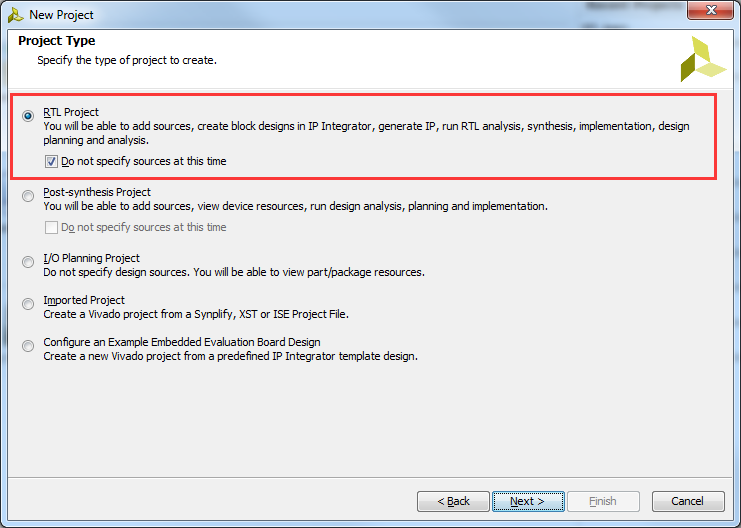
****

图2-4 选择RTL工程

5）在器件板卡选型界面中，如图2-5，在Search栏中输入xc7a35tcpg236搜索本次实验所使用的Basys3板卡上的FPGA芯片。并选择xc7a35tcpg236-1器件。（器件命名规则详见xilinx官方文档）点击Next。

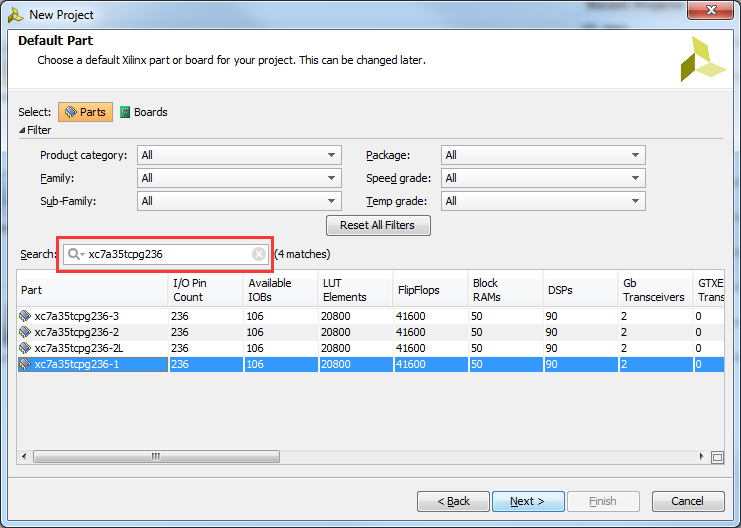


图2-5 板卡选型

6）最后在新工程总结中，检查工程创建是否有误。没有问题，则点击Finish，完成新工程的创建。

**2、添加已设计好的IP核。**

工程建立完毕，我们需要将project\_1这个工程所需的IP目录文件夹复制到本工程文件夹下。本书实验需要的IP目录为Lab\_IP及74LSXX\_LIB(注：其中Lab\_IP是封装好的单个门电路IP核，74LSXX\_LIB是按74系列集成电路封装的IP核，比较复杂的电路选后者能更简洁一些，可以根据工程需求自行选择)。

1）在Vivado设计界面的左侧设计向导栏中，如图2-6，点击Project Manager目录下的Project Setting。

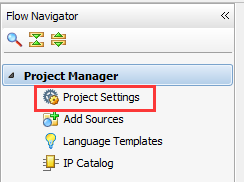
****

图2-6 设计向导栏选项

2）在Project Setting界面中，如图2-7，选择IP选项，进入IP设置界面。点击Add Respository...，添加本工程文件夹下的IP\_Catalog目录：

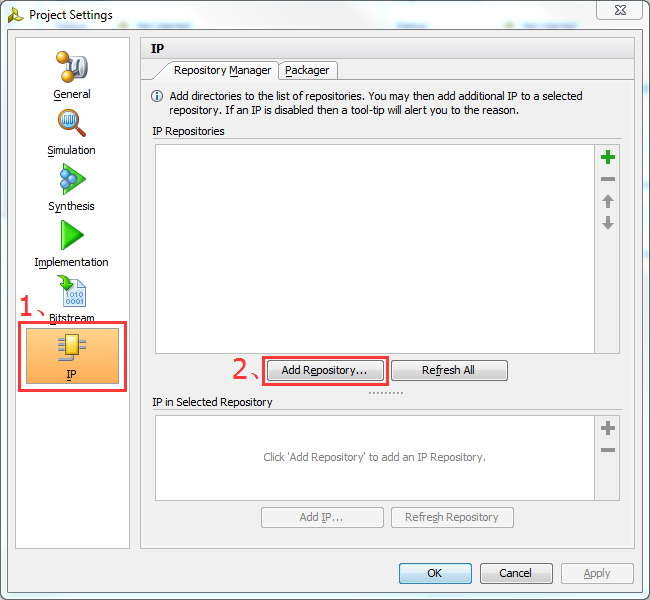
****

图2-7 IP设置

3）完成目录添加后，可以看到所需IP核已经自动添加。点击OK完成IP添加。如图2-8。

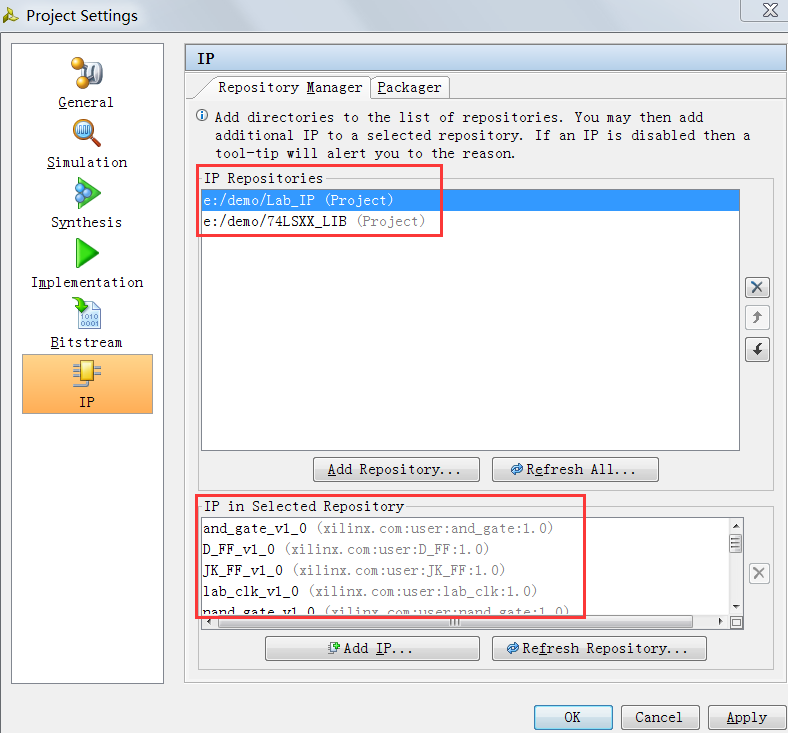


图2-8 完成IP添加

**3、创建原理图，添加IP，进行原理图设计。**

1）在Project Navigator下的IP Integrator目录下，点击Create Block Design，创建原理图，如图2-9。

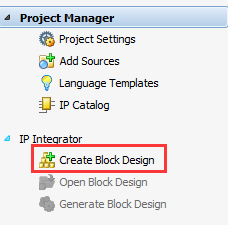
****

图2-9 创建原理图

2）在弹出的创建原理图界面中，如图2-10，保持默认。点击OK完成创建。

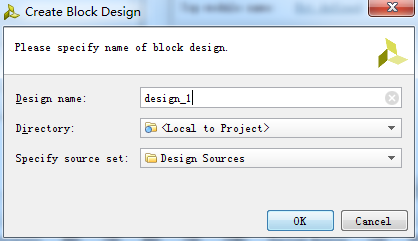
****

图2-10 原理图文件名路径界面

3）在原理图设计界面中，如图2-11，添加IP的方式有3种：

* 在设计刚开始时，原理图界面的最上方有相关提示，可以点击Add IP，进行添加IP。
* 在原理图设计界面的左侧，有相应快捷键。
* 在原理图界面中，鼠标右击选择Add IP。

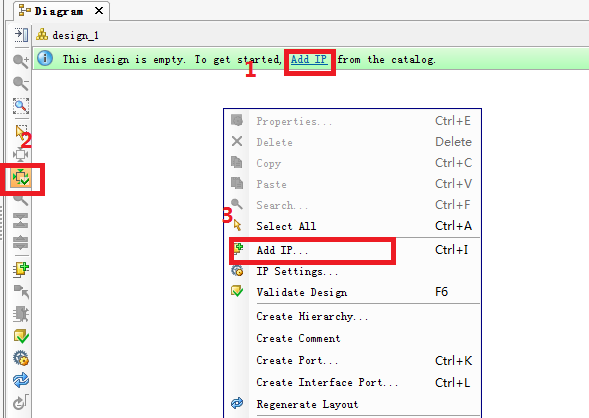
****

图2-11 原理图添加IP

4）在IP选择框中，输入gate，搜索本实验所需要的IP。

5)按Enter键，或者鼠标双击该IP，可以完成添加。如添加1个nand\_gate如图2-12。

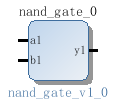


图2-12 模块图及引脚命名

**注意：Vivado要求模块的输入引脚（左侧）必须都有一个端口，而输出引脚则不必。**

6）添加完IP后，进行端口设置和连线操作。连线时，将鼠标移至IP引脚附近，鼠标图案变成铅笔状。此时，点击鼠标左键进行拖拽。Vivado会提醒用户可以与该引脚相连的引脚或端口。

7）创建与外界相连的端口有两种方式：

* 点击选中IP的某一引脚，点击右键在菜单中选择Make External…自动创建与IP引脚同名、同方向的端口；
* 点击选中IP的某一引脚，在右键菜单中选择Create Port，然后在弹出的窗口中设置端口名称，方向以及类型；

8）通过点击端口，可以在原理图窗口的左侧external port properities窗口中修改端口名称。如将上一步中自动创建的端口名a1改为a1\_in，b1改为b1\_in。

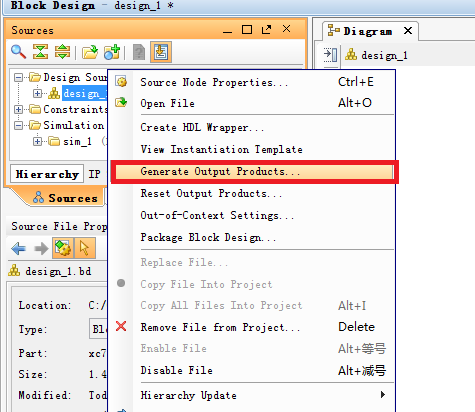
9）完成原理图设计后，生成顶层文件。在Source界面中右击design\_1，选择Generate Output Products，如图2-13。

图2-13 生成顶层文件

10）在生成输出文件的界面中点击Generate，如图2-14。生成完输出文件后，再次右击design\_1，选择Create HDL Wrapper，创建HDL代码文件。对原理图文件进行实例化。在创建HDL文件的界面中，保持默认选项，点击OK，完成HDL文件的创建。如图2-15。至此，原理图设计已经完成。

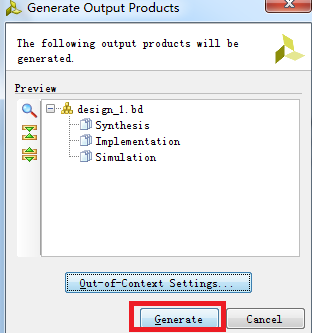
****

图2-14 生成顶层文件界面

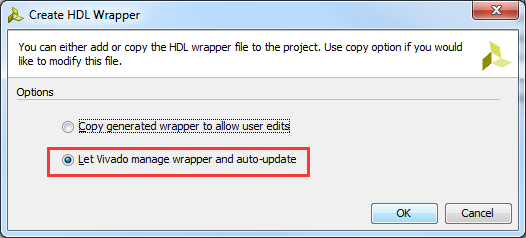
****

图2-15 创建HDL文件界面

**4.添加管脚约束、综合、实现、生成bit流并下载到实验板**

1）在导航栏的RTL Analysis里选择打开Elaborated Design。

2）在Vivado标签栏中，将界面调整到I/O PLANNING，如图2-16。



图2-16 Vivado标签栏

3）在屏幕下方I/O ports窗口中将设计端口与FPGA引脚进行关联，如图2-17。

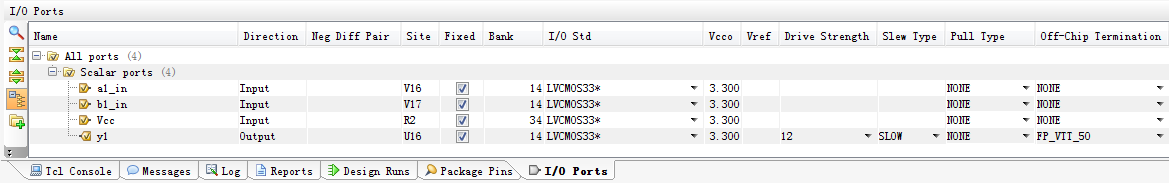


图2-17 引脚约束

根据1.2节的介绍，实验板上FPGA的V16、V17引脚分别与拨码开关SW1和SW2相连，拨码开关朝上拨为输入高电平。U16引脚与LED灯LD0连接，当有高电平输出时，灯被点亮（引脚与拨码开关和LED灯的对应关系见表1-2）。

因此，在Site栏，三个引脚a1\_in、b1\_in、y1分别设置为V16、V17、U16；Fixd栏会自动勾选；I/O std栏均设为LVCMOS33即可。

完成后保存，将提示为存储的文件取名，在此用同样英文命名，如图2-18。

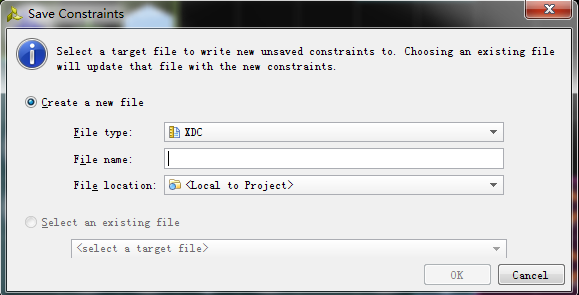


图2-18 保存约束文件

4）综合验证。

在导航栏的Synthesis里选择打开Run synthesis。

5）完成综合验证后，选择Run Implementation，进行工程实现，如图2-19。

6）工程实现完成后，选择Generate Bitstream，生成编译文件，如图2-20。

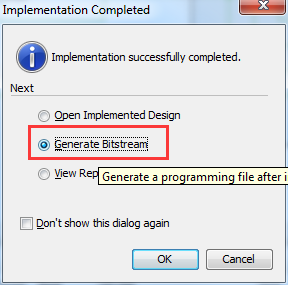
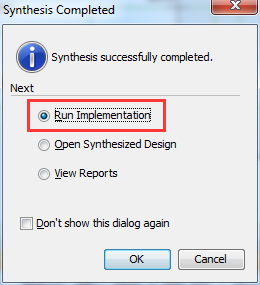


图2-19 进行工程实现图 2-20 生成编译文件

7）生成编译文件后，选择Open Hardware Manager，打开硬件管理器，进行板级验证，如图2-21。

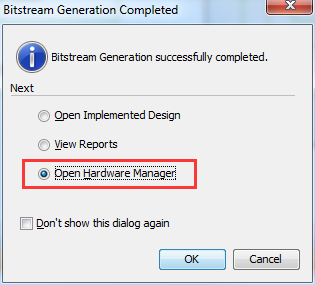


图2-21 打开Hardware Manager

8）打开目标器件，点击Open target。如果初次连接板卡，选择Open a New Hardware。如果之前连接过板卡，可以选择Open Recent Target，在其列表中选择相应板卡。



图2-22 打开目标器件

在打开新硬件目标界面中，点击Next进行创建。选择Local server，点击Next。



图2-23 打开新硬件目标界面

点击Next，再点击Finish，完成创建。

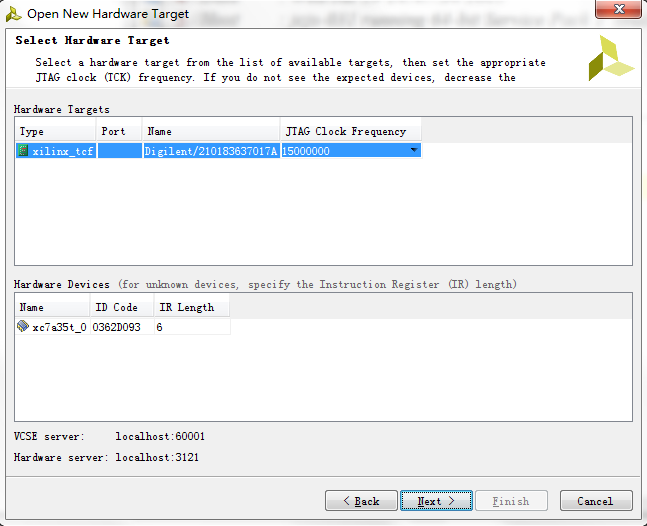


图2-24 完成创建

9）下载bit文件。

点击Hardware Manager上方提示语句中的Program device。选择目标器件。检查弹出框中所选中的bit文件，然后点击Program进行下载。进行板级验证。

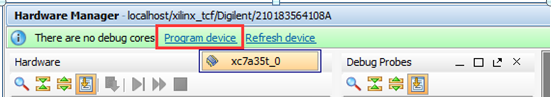


图2-25 下载bit文件

拨动最右端的两个拨码开关SW1和SW0，观察LED灯LD0的明亮情况，查看开关与灯组成的逻辑是否实现了一个与非门的功能。

# 第二部分数字逻辑实验

## 3.实现布尔表达式

### 3.1 实验要求

**目的：**通过实验，使学生学会根据布尔表达式实现与其对应的数字电路。

**内容：**使用第一部分介绍的基本门电路IP核，利用Vivado设计一个组合逻辑电路，实现布尔表达式的逻辑功能。

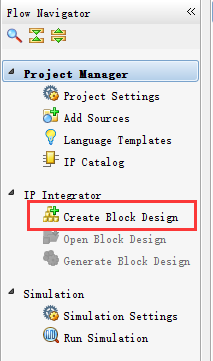
**报告：**按要求填写实验报告。

**扩展：**使用附录2中Verilog语言的结构化描述进行设计及实现。

### 3.2 实验步骤

#### 3.2.1 设计示例

1. 分析表达式，*A、B、C*作为电路的输入信号。*Y*是电路的输出信号。根据表达式画卡诺图，观察是否可以化简，得到化简后的方程进行电路设计。
2. 化简后。
3. 根据方程可知，仅需要IP核中的or\_gate（或门）和not\_gate（非门）。
4. 创建新的工程(**注：不要使用中文路径，中文名字！**)，加载IP核。
5. 创建原理图，进行原理设计。
   1. 在Project Navigator下的IP Integrator目录下，点击Create Block Design，创建原理图。
   2. 保持默认，点击OK完成创建。
   3. 在Diagram中通过添加IP来添加逻辑门电路。

图3-1 创建原理图

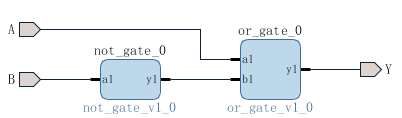


图3-2 表达式的BD设计图示例

1. 右击Block\_Designs中的design\_1，先点击“Generate Output Products”，然后点击“Create HDL Wrapper”。

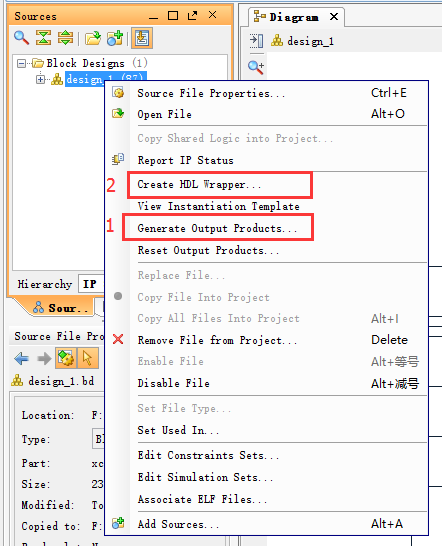


图3-3 生成顶层文件

1. 打开Elaborated Design，配置管脚约束（I/O PLANNING），为输入指定相应的拨码开关，管脚可以查看1.2.6。
2. 综合、实现、生成bitstream。
3. 完成板级验证。

#### 3.2.2 设计电路实现布尔表达式

根据表达式进行化简。根据化简的结果，仅使用**与非门**，设计一个组合逻辑电路实现4变量的布尔表达式。

**要求**：在实验报告中，参考表4-1列出4位二进制数不同取值，填写对应的Y值、表达式的化简结果，以及表达式对应的电路图。

表3-1 记录表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A（SW） | B（SW） | C（SW） | D（SW） | Y（LED） |
| 0 | 0 | 0 | 0 |  |
| 0 | 0 | 0 | 1 |  |
| 0 | 0 | 1 | 0 |  |
| 0 | 0 | 1 | 1 |  |
| ... | … | … | … |  |
| 1 | 1 | 1 | 1 |  |
| 化简结果 |  | | | |

## 4.加法器

### 4.1实验要求

**目的：**通过实验，使学生学会设计加法器。

**内容：**利用Vivado设计一个2位并行加法器，并利用半加器和全加器实现一个2位并行加法器。

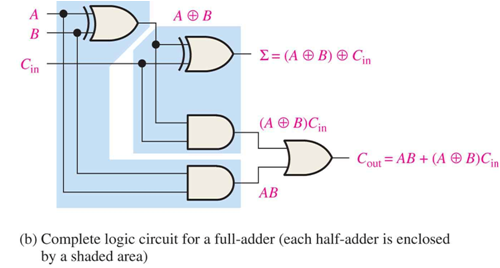
**报告：**按要求填写实验报告。

**扩展：**使用附录2中Verilog语言的结构化描述进行设计及实现。

### 4.2 实验步骤

#### 4.2.1全加器的设计

根据图4-1（教材P198）介绍的原理用Block Design设计一个全加器：

图4-1 全加器原理图

1. 分析输入（加数A、被加数B及低位进位Cin）、输出（和S及进位位Cout）；根据方程和IP核库判断需要使用的门电路以及个数（异或门、与门、或门）。
2. 创建新的工程(**注：不要使用中文路径，中文名字！**)，加载IP核。
3. 在Project Navigator下的IP Integrator下，点击Create Block Design，创建新的Block Design，根据上面的分析添加基本门电路、输入输出端口并连线（**注意未使用的输入管脚不能为空**），将该设计保存时命名为fulladder(**注意名字！**)。

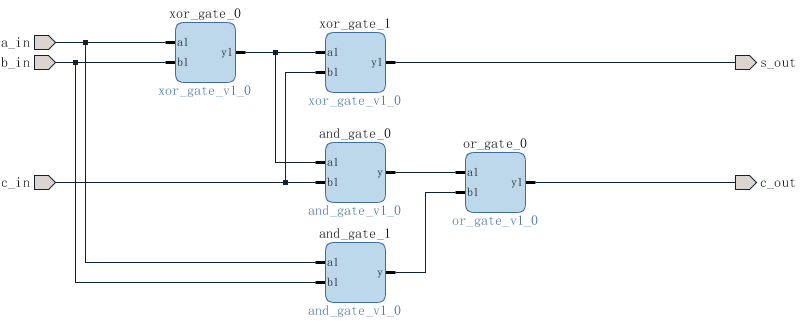


图4-2 全加器BD设计图

1. 完成原理图设计后，生成顶层文件（Generate Output Products）和HDL代码文件（Create HDL Wrapper）。
2. 配置管脚约束（I/O PLANNING），为3个输入指定相应的拨码开关，为2个输出指定相应的led灯显示。

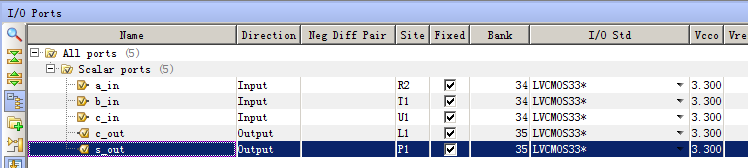


图4-3 管脚约束示例

1. 综合、实现、生成bitstream。
2. 板级验证。

#### 4.2.2利用逻辑门构成半加器和全加器，并设计一个2位并行加法器

参照下图（教材P202图6.9）用Block Design设计一个2位并行加法器：使用逻辑门实现一个半加器级联一个全加器构成两位并行加法器，完成设计，将设计下载到实验板进行验证。

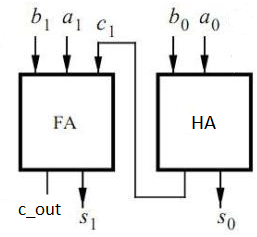


图4-4 2位并行加法器原理图

1. 创建新的工程，加入lab\_IP核，按上图原理设计一个2位二进制加法器，添加管脚约束、综合、实现、生成bitstream进行板级验证。

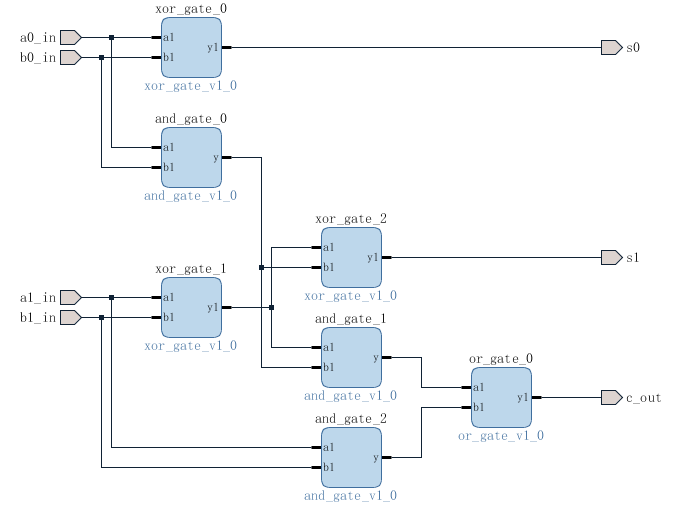


图4-5 2位并行加法器的BD设计图

**要求**：在实验报告中参考表3-1列出两个2位二进制数3种不同取值对应的LED灯显示。

表4-1 记录表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A1 | A0 | B1 | B0 | LED灯显示 |
| 0 | 1 | 1 | 0 |  |
| 1 | 0 | 0 | 1 |  |
| 0 | 1 | 1 | 1 |  |

## 5.译码器和编码器

### 5.1 实验要求

**目的：**通过实验，使学生学会设计译码器和优先编码器。

**内容：**使用基本门电路IP核设计一个3-8译码器、一个4-2优先编码器。

**报告：**按要求填写实验报告。

**扩展：**1**.** 封装IP核

2.使用附录2中Verilog语言的结构化描述进行设计及实现。

### 5.2 实验步骤

#### 5.2.1设计译码器

图5-1是3-8译码器（74LS138）的电路图。我们将依据此图来进行设计。

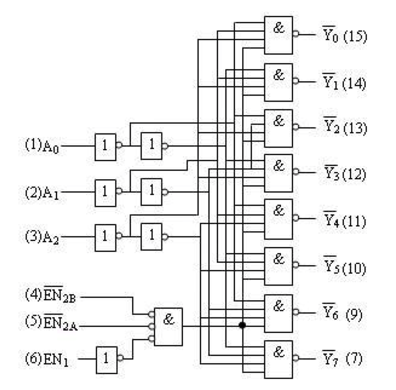
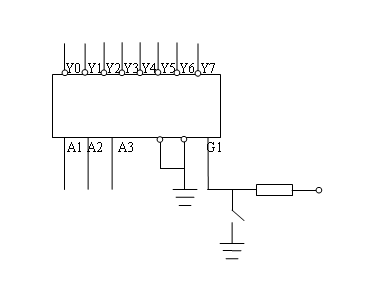
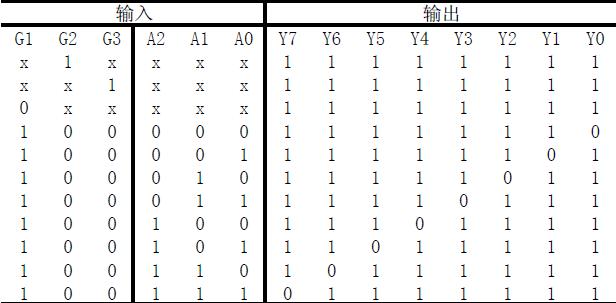


图5-1 电路及原理图

1. 分析输入、输出；根据方程和IP核库判断需要使用的门电路以及个数。
2. 创建新的工程，加载需要使用的IP核。
3. 创建BD设计文件，添加你所需要的IP核，根据图5-1进行端口设置和连线操作。
4. 完成原理图设计后，生成顶层文件（Generate Output Products）和HDL代码文件（Create HDL Wrapper）。
5. 配置管脚约束（I/O PLANNING），为输入指定相应的拨码开关，为输出指定相应的led灯显示。
6. 综合、实现、生成bitstream。
7. 仿真验证，依据3-8译码器输入输出真值表（表5-1），在实验板验证试验结果。

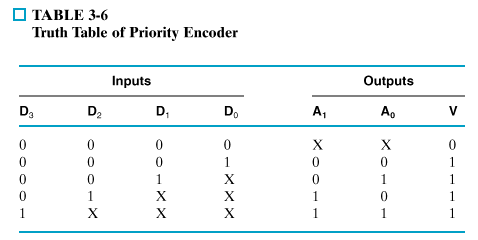
表5-1 译码器真值表



#### 5.2.2设计4-2优先编码器

表5-2是一个4-2优先编码器的真值表。我们将依据此图来进行设计和验证。

表5-2 优先编码器真值表



1. 分析输入、输出，列出方程。根据方程和IP核库判断需要使用的门电路以及个数。
2. 创建新的工程，加载需要使用的IP核。
3. 创建BD设计文件，添加你所需要的IP核，进行端口设置和连线操作。
4. 完成原理图设计后，生成顶层文件（Generate Output Products）和HDL代码文件（Create HDL Wrapper）。
5. 配置管脚约束（I/O PLANNING），为输入指定相应的拨码开关，为输出指定相应的led灯显示。
6. 综合、实现、生成bitstream。
7. 仿真验证，依据真值表，在实验板验证试验结果。

## 6.多路选择器

### 6.1 实验要求

**目的：**通过实验，使学生学会设计多路选择器。

**内容：**利用Vivado设计一个2选1多路选择器和一个4选1多路选择器。

**报告：**按要求填写实验报告。

**扩展：**使用附录2中Verilog语言的结构化描述进行设计及实现。

### 6.2 实验步骤

#### 6.2.1 设计4选1多路选择器并封装为IP核

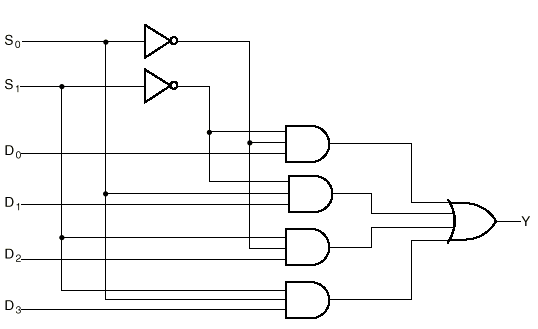
图6-1是一位4选1多路选择器的电路图，我们将依据此图来进行设计。

图6-1 4选1多路选择器电路图

1. 分析输入、输出；根据方程和IP核库判断需要使用的门电路以及个数。
2. 创建新的工程，加载需要使用的IP核。
3. 创建BD设计文件，添加你所需要的IP核，进行端口设置和连线操作。
4. 完成原理图设计后，生成顶层文件（Generate Output Products）和HDL代码文件（Create HDL Wrapper）。
5. 配置管脚约束（I/O PLANNING），为输入指定相应的拨码开关，为输出指定相应的led灯显示。
6. 综合、实现、生成bitstream。
7. 仿真验证

4-1多路器的逻辑表达式为Y = S1’S0’D0 + S1’S0D1 + S1S0’D2 + S1S0D3，对应功能如下表，请仿真验证所设计的多路器电路是否正确。

表6-1 4-1多路器功能表

|  |  |  |
| --- | --- | --- |
| S1 | S0 | Y |
| 0 | 0 | D0 |
| 0 | 1 | D1 |
| 1 | 0 | D2 |
| 1 | 1 | D3 |

#### 6.2.2利用自己设计的4选1多路选择器实现逻辑函数

请利用自己设计的多路器实现以下函数并上板验证：

1) y = ab + a’b + ab’

2) y = abc + ab’ + a’bc

要求：实验报告给出详细设计过程。

## 7.寄存器

### 7.1实验要求

**目的：**通过实验，使学生学会设计寄存器。

**内容：**设计一个4位并行输入并行输出寄存器和一个4位循环移位寄存器（IP核D\_FF（带低有效置位复位的下降沿D触发器）和lab\_clk（时钟分频））。

**报告：**按要求填写实验报告。

**扩展：**使用附录2中Verilog语言的结构化描述进行设计及实现。

### 7.2实验步骤

#### 7.2.1 4位并行输入并行输出寄存器

图7-1是4位并行输入并行输出寄存器的原理图。我们将依据此图来进行设计。

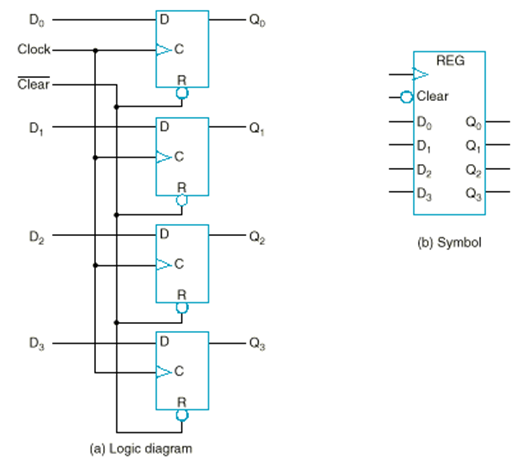


图7-1 4位并行输入并行输出寄存器原理图

实验步骤：

1. 分析输入、输出；根据方程和IP核库判断需要使用的门电路以及个数。
2. 按照第二章模块化设计流程新建工程，并且导入正确的IP核，准备新建BlockDesign文件。
3. 参考图7-2，在导入的IP核中选择D触发器进行BlockDesign设计，并且加入清零和置位功能接口，清零端和置位端都是低有效。

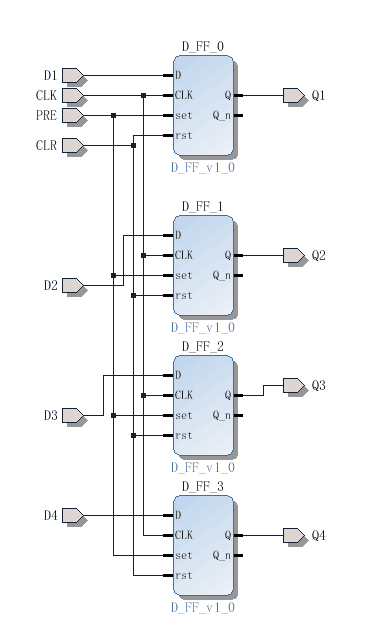


图7-2 4位并行输入并行输出的寄存器BD设计图

1. BD文件设计好之后，点击GenerateOutputProducts和CreateHDLWrapper，之后进入RTL分析，进行引脚分配如图7-3。

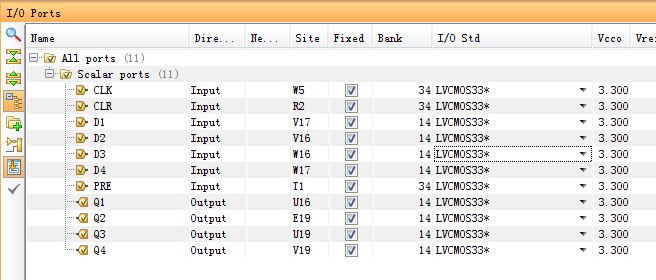


图7-3 引脚分配图

1. 把寄存器输入，清零端和置位端接到开关引脚上面，把输出接到LED灯引脚上面，把CLK接到时钟W5引脚上。
2. 引脚分配完成之后进行综合，分析，生成比特流。
3. 最后下载到实验板上面进行验证实验结果：当清零端为低电平时，四个LED灯都不亮代表寄存器输出为0，当置位端为低电平时四个LED灯全部亮，代表寄存器输出为1，当清零端和置位端为高电平时，LED灯亮暗根据对应的四个开关电平的高低。

#### 7.2.2 4位循环移位寄存器

图7-4是4位循环移位寄存器的原理图。我们将依据此图来进行设计。

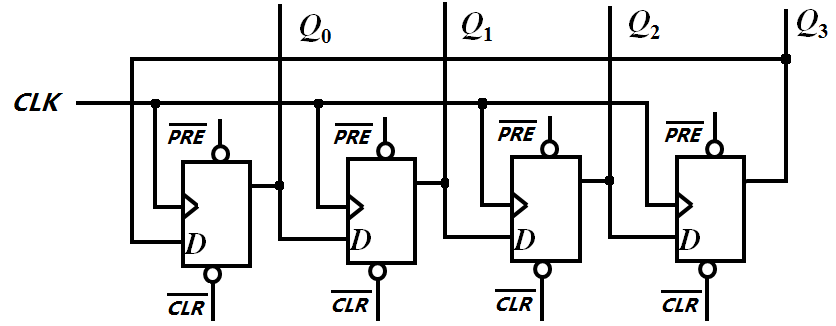


图7-4 4位循环移位寄存器原理图

实验步骤：

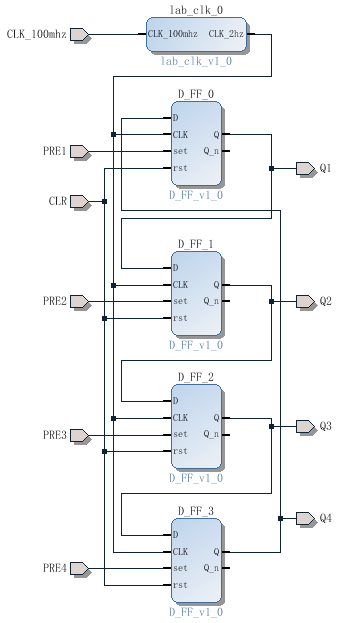
1. 分析输入、输出；根据原理图和IP核库判断需要使用的门电路以及个数。
2. 按照第二章模块化设计流程，新建工程，并且导入正确的IP核，准备新建BlockDesign文件。
3. 参考图7-5，在导入的IP核中选择D触发器进行BlockDesign设计，并且加入清零和置位功能接口，清零端和置位端都是低有效，将上一个寄存器的输出接到下一个寄存器的输出端，最后一个寄存器的输出接到第一个寄存器的输入，实现循环移位。

图7-5 4位循环移位寄存器BD设计图

1. BD文件设计好之后，点击Generate Output Products和Create HDL Wrapper，之后进入RTL分析，进行引脚分配如图7-6。

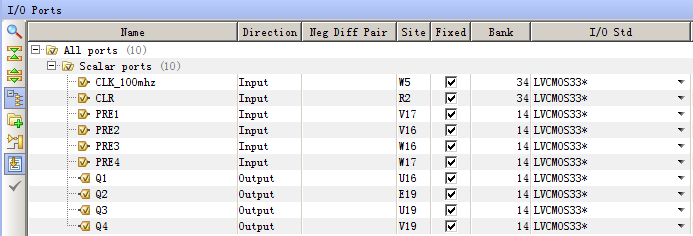


图7-6 4位循环移位寄存器引脚分配图

1. 把寄存器输入，清零端和置位端接到开关引脚上面，把输出接到LED灯引脚上面，把CLK接到时钟W5引脚上。
2. 引脚分配完成之后进行综合，分析，生成比特流，最后下载到实验板上面进行验证实验结果：当清零端为低电平时，四个LED灯都不亮代表寄存器输出为0，当置位端为低电平时四个LED灯全部亮，代表寄存器输出为1，当清零和置位端为高电平时，LED会根据存储在寄存器中的内容进行循环位移。实际操作时应注意先使用CLR开关清零，然后使用PRE开关置位（如0100），然后将5个开关都同时置为高电平，这样才能观察到开始循环移位。

注意事项：

1. 实验板W5引脚的时钟频率为100MHz。
2. IP核中lab\_clk为分频器输入频率为100MHz，输出频率为2Hz。
3. 正沿触发D触发器的清零端clr和置位端pre低电平有效，不能留有空脚，否则无法实现清零和置位功能。
4. 本实验引脚分配较多，在引脚分配时要注意，最好按照开关，LED灯对应分配，方便观察实验现象。
5. 在设计4位并行输入并行输出寄存器时CLK直接接到W5引脚，因为时钟频率很快，在输入和输出之间几乎没有时间差；若输入后输出无变化，检查清零端是否有效。
6. 在设计4位循环移位寄存器时，CLK经过lab\_clk分频器频率降为2Hz，在实验板上验证实验时，首先将置位端和清零端都设置为高电平，既开关处于闭合状态，然后置位端开始进行置数，但是要注意置位时间，时钟频率为2Hz，每隔0.5秒就进行一次位移，如果置位时间过长LED灯被全部点亮就无法观察到循环过程。

实验问题分析：

1. 在并行输入输出寄存器中LED灯一直亮，检查输入端口引脚是否为空，若输入端口引脚为空，则输入和输出都为高电位LED灯会一直亮，应将输入引脚接到开关引脚上。
2. 若置位端和清零端失效，检查对应端口引脚是否为空，置位端和清零端都是低电平有效，空脚输入为高电平，置位和清零功能失效。
3. 若循环右移寄存器观察不到循环位移，检测时钟是否进行分频，若未进行分频，时钟频率过高循环过快，无法观察到LED灯闪烁，对时钟频率分频降低频率，方便观查现象。

## 8.计数器

### 8.1 实验要求

**目的：**通过实验，使学生深入理解计数器的工作原理，学会设计和使用计数器。

**内容：**设计一个模8异步计数器和一个模5计数器。可使用的IP核包括：JK\_FF（带低有效置位复位的下降沿触发JK触发器）和lab\_clk（时钟分频模块）。

**报告：**按要求填写实验报告。

**扩展：**使用附录2中Verilog语言的结构化描述进行设计及实现。

### 8.2 实验步骤

#### 8.2.1 模8异步计数器

图8-1是模8异步计数器的原理图。我们将依据此图来进行设计。

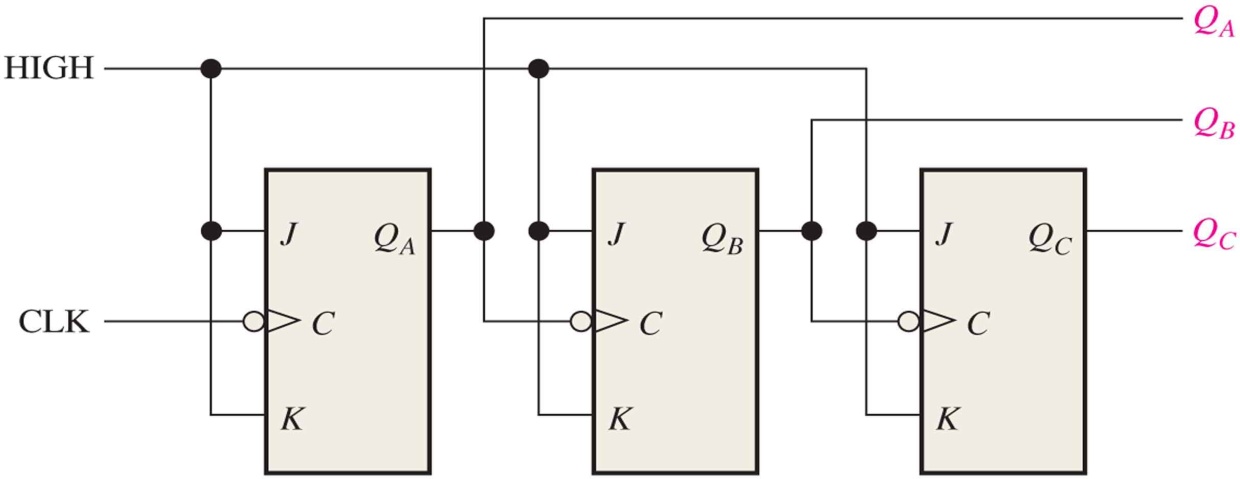


图8-1 模8异步计数器原理图

实验步骤：

1. 分析输入、输出；根据方程和IP核库判断需要使用的门电路以及个数。
2. 按照第二章模块化设计流程新建工程，并且导入正确的IP核，准备新建BlockDesign文件。为了方便观察结果，使用时钟分频模块对时钟进行分频。
3. 参考图7-2，在导入的IP核中选择JK\_FF和lab\_clk进行BlockDesign设计。

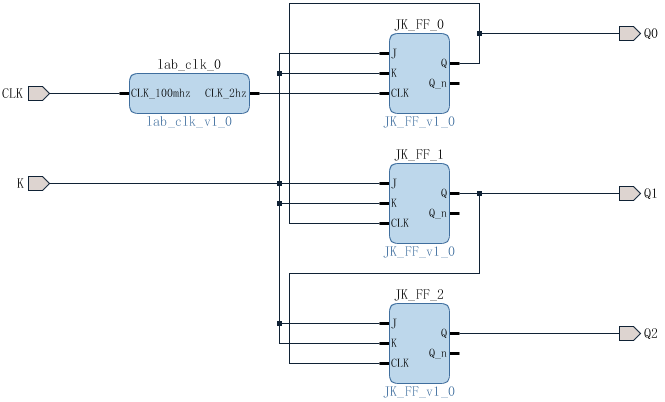


图8-2 模8异步计数器BD设计图

1. BD文件设计好之后，点击GenerateOutputProducts和CreateHDLWrapper，之后进入RTL分析，进行引脚分配如下图：

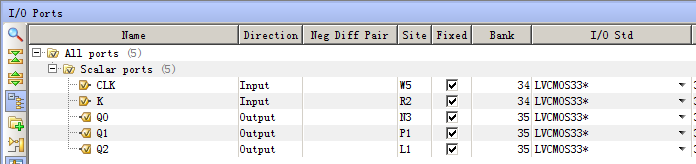


图8-3 引脚分配图

1. 把寄存器输入，清零端和置位端接到开关引脚上面，把输出接到LED灯引脚上面，把CLK接到时钟W5引脚上。
2. 引脚分配完成之后进行综合，分析，生成比特流。
3. 最后下载到实验板上面进行验证实验结果：将K端接入高电平后应观察到Q2Q1Q0呈现000-001-010-011-100-101-110-111-000的循环。

#### 8.2.2 模5计数器

请在上一节完成的模8计数器基础上构建一个模5的计数器，使得LED灯呈现000-001-010-011-100-000的循环。

## 9.流水灯

### 9.1实验要求

**目的：**通过实验，使学生学会简单的时序逻辑设计。

**内容：**使用尽可能少的D触发器设计一个4位流水灯。要求：4盏灯一字排列，从右到左，灯依次点亮熄灭，前一盏灯熄灭后，后一盏灯点亮。用1表示点亮，0表示熄灭的话，那么LED灯呈现0001->0010->0100->1000->0001的循环。该电路还应该有一个使能开关，当开关闭合时，电路开始工作，否则电路保持在0000。

**报告：**按要求填写实验报告。

### 9.2实验步骤

1. 分析输入、输出；列出状态图、状态表、状态方程。以下示例供参考，也可以自己重新设计。设状态Q0Q1Q2，状态转换图如下：

Reset

1000

0100

0010

Reset

Reset

Reset

Reset

0001

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Q0 | Q1 | Q2 | Reset | Q0\* | Q1\* | Q2\* | Y0 | Y1 | Y2 | Y3 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

图9-1 状态转换图和转换表示例

1. 输出方程：Y0=Q0Q1’Q2’

Y1=Q0’Q1Q2

Y2=Q0’Q1Q2’

Y3=Q0’Q1’Q2

状态方程：Q0\*=Q0’Q1Q2Reset’

Q1\*=(Q0’Q1’Q2+Q0’Q1Q2’)Reset’

Q2\*=(Q0’Q1’Q2’+Q0’Q1Q2’+Q0Q1’Q2’)Reset’

1. 根据方程和已有IP核判断需要使用的门电路以及个数。
2. 新建工程，并且导入正确的IP核，准备新建BlockDesign文件，根据原理图连线。
3. 完成原理图设计后，生成顶层文件（Generate Output Products）和HDL代码文件（Create HDL Wrapper）。
4. 配置管脚约束（I/O PLANNING）。
5. 综合、实现、生成bitstream。
6. 下载程序到实验板，进行板级功能验证。

# 附录1：关于Vivado中的引脚约束

有些同学会对引脚约束感到困惑：在实验板上，拨码开关被命名为SW，然而每个拨码开关又具有一个类似于U17的编号。

在此，列出以下几点以方便对引脚约束的理解。

1.电路的实现场所在FPGA芯片内部，任何一个数字电路都会有输入输出端口；

2.将FPGA的引脚的一端与设计的端口相对应，相连接，这一过程为引脚约束，这一过程的是可以通过Vivado来进行重配置的；

3.在芯片外部，FPGA的引脚的另一端与实验板上的外设相连，这种连接是通过焊接和实际的走线实现的，是物理的，不可更改的；

**输入设备**

**输出外设**

**输入引脚**

**输入端口**

**输出端口**

**输出引脚**

**FPGA内部**

**用户的电路设计**

# 附录2：了解Verilog语言

Verilog是由Gateway设计自动化公司的工程师于1983年末创立的。当时Gateway设计自动化公司还叫做自动集成设计系统（Automated Integrated Design Systems），1985年公司将名字改成了前者。该公司的菲尔·莫比（Phil Moorby）完成了Verilog的主要设计工作。1990年，Gateway设计自动化被Cadence公司收购。

1990年代初，开放Verilog国际（Open Verilog International, OVI）组织（即现在的Accellera）成立，Verilog面向公有领域开放。1992年，该组织寻求将Verilog纳入电气电子工程师学会标准。最终，Verilog成为了电气电子工程师学会1364-1995标准，即通常所说的Verilog-95。

设计人员在使用这个版本的Verilog的过程中发现了一些可改进之处。为了解决用户在使用此版本Verilog过程中反映的问题，Verilog进行了修正和扩展，这部分内容后来再次被提交给电气电子工程师学会。这个扩展后的版本后来成为了电气电子工程师学会1364-2001标准，即通常所说的Verilog-2001。Verilog-2001是对Verilog-95的一个重大改进版本，它具备一些新的实用功能，例如敏感列表、多维数组、生成语句块、命名端口连接等。目前，Verilog-2001是Verilog的最主流版本，被大多数商业电子设计自动化软件包支持。

2005年，Verilog再次进行了更新，即电气电子工程师学会1364-2005标准。该版本只是对上一版本的细微修正。这个版本还包括了一个相对独立的新部分，即Verilog-AMS。这个扩展使得传统的Verilog可以对集成的模拟和混合信号系统进行建模。容易与电气电子工程师学会1364-2005标准混淆的是加强硬件验证语言特性的SystemVerilog（电气电子工程师学会1800-2005标准），它是Verilog-2005的一个超集，它是硬件描述语言、硬件验证语言（针对验证的需求，特别加强了面向对象特性）的一个集成。

2009年，IEEE 1364-2005和IEEE 1800-2005两个部分合并为IEEE 1800-2009，成为了一个新的、统一的SystemVerilog硬件描述验证语言（hardware description and verification language, HDVL）。

对于硬件设计，它具有以下优点：

* Verilog HDL是一种通用的硬件描述语言，易学易用。由于它的语法与C语言类似，因此对于具有C语言编程经验的设计者来说，很容易学习和掌握。
* Verilog HDL允许在同一个电路模型内进行不同抽象层次的描述。设计者可以从开关、门、RTL或者行为等各个层次对电路模型进行定义。同时，设计者只需学习一种语言就能够使用它来描述电路的激励，进行层次化设计。
* 绝大多数流行的综合工具都支持Verilog HDL，这是Verilog HDL成为设计者的首选语言的重要原因之一。
* 所有的制造厂商都提供用于Verilog HDL综合之后的逻辑仿真的元件库，因此使用Verilog HDL进行设计，即可在更广泛的范围内选择委托制造的厂商。
* 编程语言接口（PLI）是Verilog语言最重要的特性之一，它使得设计者可以通过自己编写C代码来访问Verilog内部的数据结构。设计者可以使用PLI按照自己的需要来配置Verilog HDL仿真器。

1. **Verilog HDL中的数据类型**
2. **线网变量**

线网表示硬件单元之间的连接。就像在真实的电路中一样，线网由其连接器件的输出端连续驱动。线网一般使用关键字wire进行声明。如果没有显示地说明为向量，则默认线网的位宽为1。线网的默认值为z（高阻态）。线网的值由其驱动源决定，如果没有驱动源，则线网的值为z。举例如下：

wire a; //声明a是wire（连线）类型

wire b, c; //声明b和c也是wire（连线）类型

assign a = b & c; //a由b & c来驱动

wire d = 1’b0; //连线d在声明时，d被赋值为逻辑值0，

// 1’b0中1是数据位数为1，b代表二进制数，0是该数的值，总的意思就是

// 1位二进制数0

1. **寄存器变量**

寄存器用来表示存储元件，它保持原有的数值，直到被改写。注意不要将这里的寄存器与实际电路中由边沿触发的触发器构成的硬件寄存器混淆。在Verilog中，术语register仅仅意味着一个保持数值的变量。与线网不同，寄存器不需要驱动源，而且也不像硬件寄存器那样需要始终信号。在仿真过程中的任意时刻，寄存器的值都可以通过赋值来改变。寄存器数据类型一般通过使用关键字reg来声明，默认值为x（不确定状态）。下面给出了如何使用寄存器的例子。

reg reset; //声明能保持数值的变量reset

initial begin //initial是初始化模块，该模块中的语句只执行一次

reset = 1’b1; //把reset初始化为1，使数字电路复位

#100 reset = 1’b0; //经过100个时间单位后，reset置逻辑0

end

1. **向量**

线网和寄存器类型的数据均可以声明为向量（位宽大于1）。如果在声明中没有指定位宽，则默认为标量（1位）。举例如下：

wire a; // 标量线网变量，默认

wire [7:0] bus; // 8位的总线

wire [31:0] busA, busB, busC; // 3条32位宽的总线

reg clock; // 标量寄存器，默认

reg [0:40] virtual\_addr; // 向量寄存器，41位宽的虚拟地址

向量通过[high#:low#]或[low#:high#]进行说明，方括号中左边的数总是代表向量的最高有效位。在上面的例子中，向量virtual\_addr的最高有效位是它的第0位。

***向量域选择***

对于上面的例子中的向量，我们可以指定它的某一位或若干个相邻位。举例如下：

busA[7] // 向量busA的第7位

bus[2:0] // 向量bus的最低3位

// 如果写成bus[0:2]是非法的，因为最高位应该写在范围说明的左侧

Virtual\_addr[0:1] // 向量virtual\_addr的两个最高位

***可变的向量域选择***

除了用常量指定向量域以外，Verilog HDL还允许指定可变的向量域选择。这样就使得设计者可以通过for循环来动态地选取向量的各个域。下面是动态选择的两个专用操作符：

[<starting\_bit>+ : width] ：从起始位开始递增，位宽为width。

[<starting\_bit>- : width] ：从起始位开始递减，位宽为width。

起始位可以是一个变量，但是位宽必须是一个常量。下面的例子说明了可变的向量域选择的使用方法：

reg [255:0] data1; // data1[255]是最高位

reg [0:255] data2; // data2[0]是最高位

reg [7:0] byte;

// 用变量选择向量的一部分

byte = data1 [31- : 8]; //从31位算起，宽度为8位，相当于data1[31:24]

byte = data1 [24+ : 8]; //从24位算起，宽度为8位，相当于data1[31:24]

byte = data2 [31- : 8]; //从31位算起，宽度为8位，相当于data1[24:31]

byte = data2 [24+ : 8]; //从24位算起，宽度为8位，相当于data1[24:31]

// 起始位可以是变量，但位宽必须是常数。因此可以通过可变域选择，

// 用循环语句选取一个很长的向量的所有位

for (j=0; j<=31; j=j+1)

byte = data1[(j\*8)+ : 8]; // 次序是[7:0], [15:8] . . . [255:148]

// 用于初始化向量的一个域

data1[(byteNum\*8)+ : 8] = 8’b0; //如果byteNum = 1，共有8位被清零，[15 : 8]

1. **参数**

Verilog允许使用关键字parameter在模块内定义常数。参数代表常数，不能像变量那样赋值，但是每个模块实例的参数值可以在编译阶段重载。通过参数重载使得用户可以对模块实例进行定制。除此之外，还可以对参数的类型和范围进行定义。举例如下：

parameter port\_id = 5; //定义常数port\_id为5

parameter cache\_line\_width = 256; // 定义高速缓冲器总线宽度为常数256

parameter signed [15:0] WIDTH; //把参数WIDTH规定为有符号，宽度为16位

通过使用参数，用户可以更加灵活地对模块进行说明。用户不但可以根据参数来定义模块，还可以方便地通过参数重定义来改变模块的行为：通过模块实例化或使用defparam语句改变参数值。

Verilog中的局部参数使用关键字locaparam来定义，起作用等同于参数，区别在于它的值不能改变，不能通过参数重载语句（defparam）或通过有序参数列表或命名参数赋值来直接修改。例如，状态机的状态编码是不能被修改的，为了避免被意外的更改，应当将其定义为局部参数。举例如下：

locaparam state1 = 4’b0001,

state1 = 4’b0010,

state1 = 4’b0100,

state1 = 4’b1000;

除了以上几种数据类型外，Verilog中还有integer（整型）、real（实型）、time（时间型）等数据类型，同学们可以自行查阅相关资料。

1. **模块**

Verilog使用模块（module）的概念来代表一个基本的功能块。一个模块可以是一个元件，也可以是低层次的组合。常用的设计方法是使用元件构建在设计中的多个地方使用的功能块，以便进行代码重用。模块通过接口（输入和输出）被高层的模块调用，但隐藏了内部的实现细节。这样就使得设计者可以方便地对某个模块进行修改，而不影响设计的其他部分。

在Verilog中，模块声明由关键字module开始，关键字endmodule则必须出现在模块定义的结尾。每个模块具有一个模块名，由它唯一的标识这个模块。模块的端口列表则描述这个模块的输入和输出端口。

例1 模块

module <模块名>(<端口列表>);

…

<模块的内容>

…

…

endmodule

1. **端口**

端口是模块与外界环境交互的接口，例如IC芯片的输入、输出引脚就是它的端口。对于外部环境来讲，模块内部是不可见的，对模块的调用（实例引用）只能通过端口进行。这种特点为设计者提供了很大的灵活性：只要接口保持不变，模块内部的修改并不会影响到外部环境。我们也常常将端口称为终端（terminal）。

在模块的定义中包括一个可选的端口列表。如果模块和外部环境没有交换任何信号，则可以没有端口列表。例如我们在进行功能仿真时编写的测试块，通常没有端口。

module fulladd4(sum, c\_out, a, b, c\_in); //有端口列表的模块

module tb\_fulladd4; // 没有端口列表的模块，功能仿真用作测试块

端口列表中的所有端口必须在模块中进行声明，Verilog中的端口具有以下三种类型：

input 输入端口

output 输出端口

inout 输入/输出双向端口

根据端口信号的方向，端口具有三种类型：输入、输出和输入/输出。因此模块fulladd4的端口声明如例2所示。

例2 端口声明

module fulladd4(sum, c\_out, a, b, c\_in);

// 端口声明开始

output [3:0] sum;

reg [3:0] sum;

output c\_out;

reg c\_out;

input [3:0] a, b;

input c\_in;

// 端口声明结束

. . .

<模块的内容>

. . .

endmodule

在Verilog中，所有的端口隐含地声明为wire类型，因此如果希望端口具有wire数据类型，将其声明为三种类型之一即可；如果输出类型的端口需要保存数值，则必须将其显示地声明为reg数据类型，就像例2中的reg [3:0] sum和reg c\_out。

1. **模块调用**

模块声明类似于一个模板，使用这个模板就可以创建实际的对象。当一个模块调用的时候，Verilog会根据模板创建一个唯一的模块对象，每个对象都有其各自的名字、变量、参数和输入/输出（I/O）接口。从模板创建对象的过程称为实例化（instantiation），创建的对象称为实例（instance）。以下给出模块调用的例子。

例3 模块调用

//它引用了4个T触发器

//定义名为ripple\_carry\_counter（脉动进位计数器）的模块

module repple\_carry\_counter(q, clk, reset);

output [3:0] q; // 输入/输出端口的信号和向量声明

input clk, reset; // 输入/输出端口的信号声明

// 生成了4个T触发器T\_FF的实例，每个实例都有自己的名字，每个实例都传递一组信号

// 注意每个实例都是T\_FF模块的副本

T\_FF tff0(q[0], clk, reset);

T\_FF tff1(q[1], q[0], reset);

T\_FF tff2(q[2], q[1], reset);

T\_FF tff3(q[3], q[2], reset);

endmodule

// 定义名为T\_FF（T触发器）的模块，它引用了一个D触发器。

module T\_FF(q, clk, reset);

output q;

input clk, reset;

wire d;

D\_FF dff0(q, d, clk, reset); // 调用（实例引用）D\_FF，取名为dff0

not n1(d, q); //非门（not）是Verilog语言的内部原语部件（primitive）

endmodule

// 定义名为D\_FF（D触发器）的模块。

module D\_FF(q, d, clk, reset);

output q;

input d, clk, reset;

reg q;

// 可以有多种新结构，不考虑这些结构的功能，

//只需要注意设计块是如何自顶向下的方式编写的

always @(posedge reset or negedge clk)

if (reset)

q <= 1’b0;

else

q <= d;

endmodule

1. **数据流级建模**
2. **连续赋值语句**

连续赋值语句是Verilog数据流建模的基本语句，用于对线网进行赋值。它等价于门级描述，然而是从更高的抽象角度来对电路进行描述。连续赋值语句必须以关键字assign开始，连续赋值语句具有以下特点：

* 连续赋值语句的左值必须是一个标量或向量线网，或者是标量或线网的拼接（使用{ , }对信号进行拼接），而不能是向量或向量寄存器；
* 连续赋值语句总是处于激活状态。只要任意一个操作数发生变化，表达式就会被立即重新计算，并且将结果赋给等号左边的线网；
* 操作数可以是标量或向量的线网或寄存器，也可以是函数调用；
* 赋值延迟用于控制对线网赋予新值的时间，根据仿真时间单位进行说明。赋值延迟类似于门延迟，对于描述实际电路中的时序是非常有用的。

连续赋值语句的举例如下：

//连续赋值语句，out是线网，i1和i2也是线网

assign out = i1 & i2;

//向量线网的连续赋值语句，addr是16位的向量线网

//addr1\_bits和addr2\_bits是16位向量寄存器，两者相异或

assign addr[15:0] = addr1\_bits[15:0] ^ addr2\_bits[15:0];

// 拼接操作。赋值操作符左侧是标量线网和向量线网的拼接

assign {c\_out, sum[3:0]} = a[3:0] + b[3:0] + c\_in;

1. **数据流级建模举例**

下面以二选一多路选择器对数据流级建模进行说明。

例4 数据流级建模

module mux2to1(out, i0, i1, s);

// 端口声明

output out;

input i0, i1;

input s;

// 产生输出out的逻辑方程

assign out = (~s & i0) | (s & i1); //~是逻辑取反，&是逻辑与

//下面这种描述也是正确的

//assign out = s ? i1 : i0; // ? : 是条件操作符，如果s为真，结果为i1，否则为i0

endmodule

1. **行为级建模**

在Verilog中有两种结构化的过程语句：initial和always语句，它们是行为级建模的两种基本语句。其它所有的行为语句只能出现在这两种结构化过程语句里。

与C语言不同，Verilog在本质上是并发而非顺序的。Verilog中的各个执行流程（进程）并发执行，而不是顺序执行的。每个initial语句和always语句代表一个独立的执行过程，每个执行过程从仿真时间0时刻开始执行，并且这两种语句不能嵌套使用。

1. **initial语句**

所有在initial语句内的语句构成了一个initial块。Initial块从仿真0时刻开始执行，在整个仿真过程中只执行一次。如果一个模块中包括了若干个initial块，则这些initial块从仿真0时刻开始并发执行，且每个块的执行时各自独立的。如果在块内包含多条行为语句，那么需要将这些语句组成一组，一般使用关键字begin和end将它们组合为一个语句块；如果块内只有一条语句，则不必使用begin和end。这一点类似于C语言中的{ }语句块，initial块不可综合，一般只在测试块中使用。

下面举例说明initial语句

例5 initial语句

module stimulus；

reg x, y, a, b, m;

initial

m = 1’b0; // 只有一条语句，不需要使用begin-end

initial begin

#5 a = 1’b1; // 多条语句，需要使用begin-end

#10 b = 1’b0;

end

initial begin

#10 x = 1’b0;

#25 y = 1’b1;

end

initial

#50 $finish; //仿真结束

在上面的例子中，三条initial语句在仿真0时刻开始并发执行。如果某一条语句前面存在延迟#<delay>，那么对这条initial语句的仿真将会停顿下来，在经过指定的延迟时间之后再继续执行。

由于initial块语句在整个仿真期间只执行一次，因此它一般被用于初始化、信号监视、生成仿真波形等目的。

1. **always语句**

alwsys语句包括的所有行为语句构成了一个always语句块。该always语句块从仿真0时刻开始顺序执行其中的行为语句；在最后一条执行完成后，再次开始执行其中的第一条语句，如此循环往复，直到整个仿真结束。因此always语句通常用于对数字电路中一组反复执行的活动进行建模。例6说明了一种时钟发生器的建立模型的一种方法。

例6 laways语句

module clock\_gen (output reg clock);

//在0时刻把clock变量初始化

Initial

clock = 1’b0;

// 每半个周期把clock信号的值翻转一次（周期 = 20）

always

#10 clock = ~clock;

Initial

#1000 $finish;

endmodule

1. **过程赋值语句**

过程赋值语句的更新对象是寄存器、整数、实数或时间变量。这些类型在被赋值后，其值保持不变，直到被其它过程赋值语句赋予新值。Verilog包括两种类型的过程赋值语句：阻塞赋值和非阻塞赋值语句。

***阻塞赋值语句***

串行语句块中的阻塞赋值语句按顺序依次执行。阻塞赋值语句使用“=”作为赋值符。在例7中，只有当语句x = 0执行完成之后，才执行y = 1，而语句count = count + 1按顺序在最后执行。由于阻塞赋值语句是按顺序执行的，因此如果在一个begin-end块中使用了阻塞赋值语句，那么这个块语句表现的是串行行为。

例7阻塞赋值语句

reg x, y, z;

reg [15:0] reg\_a, reg\_b;

integer count;

//所有行为语句必须放在initial或always块内部

Initial begin

x = 0; y = 1; z = 1; //标量赋值

count = 0; //整型变量赋值

reg\_a = 16’b0; reg\_b = reg\_a; // 向量的初始化

#15 reg\_a[2] = 1’b1; //带延迟的位选赋值

#10 reg\_b[15:13] = {x, y, z}; //把拼接操作的结果赋值给向量的部分位（域）

count = count + 1;

end

在例7中，begin-end块中各条执行的仿真时间为：

* x = 0到reg\_b = reg\_a之间的语句在仿真0时刻执行；
* 语句reg\_a[2] = 0在仿真时刻15执行；
* 语句reg\_b[15 : 13] = {x, y, z}在仿真时刻25执行；
* 语句count = count + 1在仿真时刻25执行；
* 由于前面的语句中分别包含了15和10个时间单位的延迟，因此语句count = count + 1将在第25时刻执行。

***非阻塞赋值语句***

非阻塞赋值语句允许赋值调度，但它不会阻塞位于同一个顺序块中其后语句的执行。非阻塞赋值使用“<=”作为赋值符。读者会注意到，它与“小于等于”关系操作符是同一个符号，但在表达式中它被解释为关系操作，而在非阻塞赋值的环境下被解释成非阻塞赋值。为了说明非阻塞赋值的意义以及与阻塞赋值的区别，让我们来考虑将例7中的部分阻塞赋值改为非阻塞赋值后的结果，例8给出了修改后的语句。

例8非阻塞赋值语句

reg x, y, z;

reg [15:0] reg\_a, reg\_b;

integer count;

//所有行为语句必须放在initial或always块内部

Initial begin

x = 0; y = 1; z = 1; //标量赋值

count = 0; //整型变量赋值

reg\_a = 16’b0; reg\_b = reg\_a; // 向量的初始化

reg\_a[2] <= #15 1’b1; //带延迟的位选赋值

reg\_b[15:13] <= #10 {x, y, z}; //把拼接操作的结果赋值给向量的部分位（域）

count <= count + 1;

end

在这个例子中，从x = 0到reg\_b = reg\_a之间的语句是在仿真0时刻顺序执行的，之后的3条非阻塞赋值语句在reg\_b = reg\_a执行完成后并发执行。

* reg\_a[2] = 1’b1被调度到15个单位之后执行，即仿真时刻为15；
* reg\_b[15:13] = {x, y, z}被调度到10个时间单位之后执行，即仿真时刻为10；
* count <= count + 1被调度到无任何延迟执行，即仿真时刻为0。

从上面的分析中可以看到，仿真器将非阻塞赋值调度到相应的仿真时刻，然后继续执行后面的语句，而不是停下来等待赋值的完成。一般情况下，非阻塞赋值是在当前仿真时刻的最后一个时间同步，即阻塞只完成之后才执行。

在上面的例子中，我们把阻塞和非阻塞赋值语句混合在一起使用，目的是想清楚地比较和说明它们的行为。需要提醒大家注意的是，不要在同一个always块中混合使用阻塞和非阻塞赋值语句。通常情况下，组合逻辑中使用阻塞赋值语句，时序逻辑中使用非阻塞赋值语句。

此外，还有条件语句（if-else-if）、多路分支语句（case- default-endcase）及循环语句（for和while），其语法与C语言类似，不再一一讲述。

1. **行为级建模举例**

同样以二选一多路选择器为例，来说明行为级建模过程。

例9行为级建模

module mux2to1(out, i0, i1, s);

// 端口声明

output out;

input i0, i1;

input s;

//输出端口被声明为寄存器类型变量

reg out;

// 若输入信号改变，则重新计算输出信号out

//造成输出信号out重新计算的所有输入信号必须写入always @(. . .)的电平敏感列表中

always @(s, i0, i1) begin

if(s) out = i1; //s为1输出i1，否则输出i0

else out = i0;

end

endmodule

在实际的工程应用各种建模方式不是独立使用的，通常是混合起来使用，使我们的设计更灵活。

1. **实际应用**

下面给出了74LS138译码器、74LS148优先编码器以及流水灯实现的Verilog代码，作为参考。

1. **74LS138译码器Verilog代码：**

*功能模块：*

`timescale 1ns / 1ps

module ls138( //模块名称

input en, //使能端

input [2:0] sel, //3位输入端

output [7:0] y //8位输出端

);

reg [7:0] y; //reg型变量在always块中才能被赋值，所以将输出端y声明为reg型

always @(en, sel) begin //always块开始,括号内是敏感信号，当信号变化时，执行always块

if(!en) y = 8'b11111111; //使能信号en无效（为低）时，输出默认状态

else begin

case(sel) //case块中是8种状态的译码

3'b000: y = 8'b1111\_1110; //数字中的下划线没有实际意义，仅用作连接，

//使数据易读

3'b001: y = 8'b1111\_1101;

3'b010: y = 8'b1111\_1011;

3'b011: y = 8'b1111\_0111;

3'b100: y = 8'b1110\_1111;

3'b101: y = 8'b1101\_1111;

3'b110: y = 8'b1011\_1111;

3'b111: y = 8'b0111\_1111;

default: y = 8'b1111\_1111; //默认状态

endcase

end

end //always块结束

endmodule

*测试模块：*

`timescale 1ns / 1ps

module tb\_ls138;

// 输入

reg en;

reg [2:0] din;

// 输出

wire [7:0] dout;

// 实例化被测试模块ls138

ls138 ls138\_init (

.en(en),

.sel(din),

.y(dout)

);

initial begin

// 初始化输入信号

din = 0;

en = 0; //使能无效

#100; //延迟100个时钟

en = 1; //使能有效

end

always #30 din = din + 1; //每30个时钟，输入加1（改变一个状态）

endmodule

1. **74LS148优先编码器Verilog代码：**

*功能模块：*

`timescale 1ns / 1ps

module ls148( //模块名称

input [7:0] I, //8位输入端

input EI, //使能输入端

output [2:0] A, //3位输出端

output GS, //输出端GS

output EO //输出端EO

);

reg [2:0] A;

reg GS;

reg EO;

always @ ( I, EI ) begin

if ( EI ) begin

A = 3'b111;

GS = 1;

EO = 1;

end

else if ( I[7] == 0 ) begin //I[7]有最高优先级

A = 3'b000;

GS = 0;

EO = 1;

end

else if ( I[6] == 0 ) begin

A = 3'b001;

GS = 0;

EO = 1;

end

else if ( I[5] == 0 ) begin

A = 3'b010;

GS = 0;

EO = 1;

end

else if ( I[4] == 0 ) begin

A = 3'b011;

GS = 0;

EO = 1;

end

else if ( I[3] == 0 ) begin

A = 3'b100;

GS = 0;

EO = 1;

end

else if ( I[2] == 0 ) begin

A = 3'b101;

GS = 0;

EO = 1;

end

else if ( I[1] == 0 ) begin

A = 3'b110;

GS = 0;

EO = 1;

end

else if ( I[0] == 0 ) begin

A = 3'b111;

GS = 0;

EO = 1;

end

else begin

A = 3'b111;

GS = 1;

EO = 0;

end

end

endmodule

*测试模块：*

`timescale 1ns / 1ps

module tb\_ls148;

//输入

reg EI;

reg [7:0] I;

//输出

wire [2:0] A;

wire GS;

wire EO;

//实例化被测试模块

ls148 ls148\_inst(

.EI(EI),

.I(I),

.A(A),

.GS(GS),

.EO(EO)

);

//初始化输入信号

initial begin

//初始化输入信号

EI = 1;

I = 8'b1111\_1111;

#100

EI = 0;

end

always #5 I = I - 1;

endmodule

1. **流水灯Verilog代码：**

`timescale 1ns / 1ps

module flowLED(

input clk, //100MHz时钟

input rst, //复位

input on\_off, //控制启停，0-off，1-on

input DIR, //控制移动方向，0-左移，1-右移

output [3:0] LED //4位LED

);

//10ms = 1/100MHz \* 1000\_000

parameter T100MS = 999\_9999;

parameter cntWidth = 24;

reg [cntWidth - 1 : 0] cnt; //计数

reg clken; //时钟使能

//计数延迟

always @(posedge clk, negedge rst) begin

if(!rst) begin //rst为低时，复位计数和时钟使能

cnt <= 0;

clken <= 0;

end

else if(cnt == T100MS) begin

cnt <= 0; //每延迟10ms，重新计数

clken <= 1; //每延迟10ms，时钟使能1次

end

else begin

cnt <= cnt + 1'b1;

clken <= 0;

end

end

reg [3:0] LEDr;

always @(posedge clk, negedge rst) begin

if(!rst) //rst为低时，复位LEDr

LEDr <= 4'b0001;

else if(on\_off && clken) begin //on\_off = 1,使能有效时，开始移位

if(!DIR)

LEDr <= {LEDr[2:0], LEDr[3]}; //左移

else if(DIR)

LEDr <= {LEDr[0], LEDr[3:1]}; //右移

end

else

LEDr <= LEDr;

end

assign LED = rst ? LEDr : 4'b0000; //rst = 0时，LED复位

endmodule

***如果想更全面的了解和掌握Verilog HDL，请同学们自行阅读夏宇闻老师编写的《Verilog HDL数字设计与综合》。***