# 1 实验一 运算器与存储器实验

在进行本次实验前，你需要具备以下实验环境及基础能力：

1. 装有Vivado的电脑一台；
   1. 本实验不对Vivado环境有硬性要求，但涉及Xilinx库中IP的实验，在不同版本环境下无法兼容，且低版本Vivado无法运行高版本生成的项目，为方便实验检查，应当尽量使用实验要求版本。
   2. 若未曾安装Vivado，请参考文档“**A03\_Vivado安装说明\_v1.00**”。
2. 熟悉Vivado的IDE环境，并能够使用其进行仿真、综合；
   1. 如果对Vivado不熟悉，参考文档“**A04\_Vivado使用说明\_v1.00**”。
3. 熟悉Nexys4 DDR开发板(Artix-7)；

请确保在实验进行前阅读过“**A01\_Nexys4\_DDR用户手册**”。

## 1.1实验目的

1. 了解随机存取存储器RAM的原理；
2. 了解算术逻辑单元ALU的原理；
3. 掌握调用Xilinx库IP(Block Memory Generator)实例化RAM的方法；
4. 熟悉并运用Verilog语言设计ALU。
5. 学习Verilog不同形式的编程方式，理解assign和always的区别；

## 1.2实验设备

1. 计算机1台(尽可能达到8G及以上内存)；
2. Nexys4 DDR实验开发板；
3. Xilinx Vivado开发套件(2018.1版本)。

## 1.3实验任务

本次实验包含两部分，包括ALU设计和RAM实例化。

**1.3.1 ALU设计实验**

图1.1 给出了一个具有*N*位输入和*N*位输出的算数逻辑单元的电路符号。算术逻辑单元接收说明执行哪个功能的控制信号*F*，执行对应功能后输出*N*位结果。

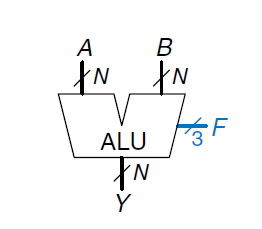


图1.1

实验要求实现以下算术运算功能，其对应的指令码及功能如下：

表1.1

|  |  |  |  |
| --- | --- | --- | --- |
| F2:0 | 功能 | F2:0 | 功能 |
| 000 | A + B(Unsigned) | 100 | ‾A |
| 001 | A - B | ***101*** | ***SLT*** |
| 010 | A AND B | 110 | 未使用 |
| 011 | A OR B | 111 | 未使用 |

本次实验将ALU输出结果通过板载七段数码管进行显示验证，原理图如图1.2所示：

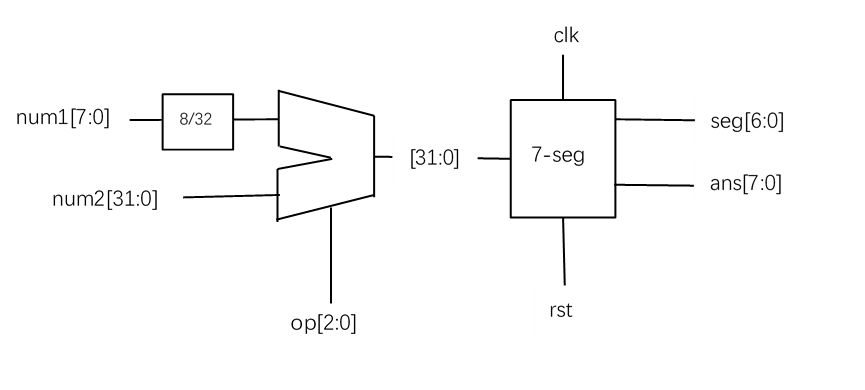


图1.2 ALU实验原理图

**实验要求：**

1. 根据ALU原理图(图1.1)，使用Verilog语言定义ALU模块，其中输入输出端口参考实验原理，运算指令码长度为[2:0]。
2. 内置一个32位num2（值为32h’01）作为输入到运算器端口A；
3. 将sw0~sw7输入到num1,经过**无符号扩展** 至32位后，输入到运算器的端口B；
4. 运算器支持“加、减、与、或、非”5种运算，需要3位（8个操作）。将sw15~sw14输入到op作为运算器的控制信号；
5. 实现SLT功能。
6. 将计算32位结果s显示到七段数码管(16进制)。
7. 验证表1.1中所有功能。
8. 给出RTL源程序（.v文件）

**1.3.2 存储器IP实例化实验**

本次实验使用Vivado的Block Memory Generator模拟数据在存储器中的存取过程。实验使用**单端口ROM**。初始化ROM存储器中的内容，通过开关选择相应的地址，将对应的存储器中内容读出来，并通过七段数码管显示。实验原理如图1.3所示：



图1.3

**实验要求：**

1）使用Block Memory Generator生成单端口ROM，并将指令coe文件加载；

2）将ROM中对应的32位指令取出并送往7-seg数码管显示。

3）验证ROM读取的值。

## 1.4实验环境

以下表格中红色部分需自行实现，黑色部分与实验发布包中提供。

**1.4.1 ALU实验**

表1.2

|  |  |
| --- | --- |
| |--top.v | 设计顶层文件，参照图1.2将各模块连接。 |
| |------alu.v | ALU模块，本次实验重点。 |
| |------display.v | 七段数码管显示模块文件，已提供。 |
| |-----------seg7.v | 七段数码管显示模块组成文件，已提供。 |
| |--constr.xdc | 综合实现时，约束文件，已提供。 |

**1.4.2存储器实验**

表1.3

|  |  |
| --- | --- |
| |--top.v | 设计顶层文件，参照图1.3将各模块连接。 |
| |------ram.ip | RAM IP，通过Block memory generator进行实例化。 |
| |------ram.coe | RAM初始化文件，已提供 |
| |------display.v | 七段数码管显示模块文件，已提供。 |
| |-----------seg7.v | 七段数码管显示模块组成文件，已提供。 |
| |--constr.xdc | 综合实现时，约束文件，已提供。 |

# 2 调用Xilinx库Block Memory Generator方法

## 2.1新建工程

新建一个工程 data\_ram， 参考文档“A04\_Vivado使用说明” :

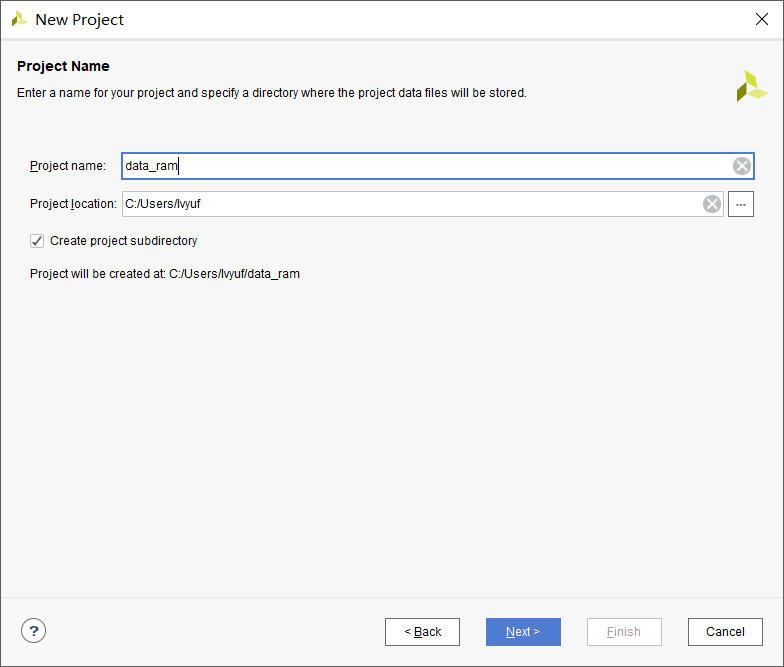


图2.1 新建工程data\_ram

## 2.2新建IP

IP核查找路径：Flow Navigator->IP Catalog->Vivado Repository ->Basic Elements->Memory Elements->Block Memory Generator

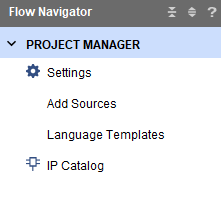


图2.2打开IP目录

或者Flow Navigator->IP Catalog，在搜索栏直接搜索Block Memory Generator

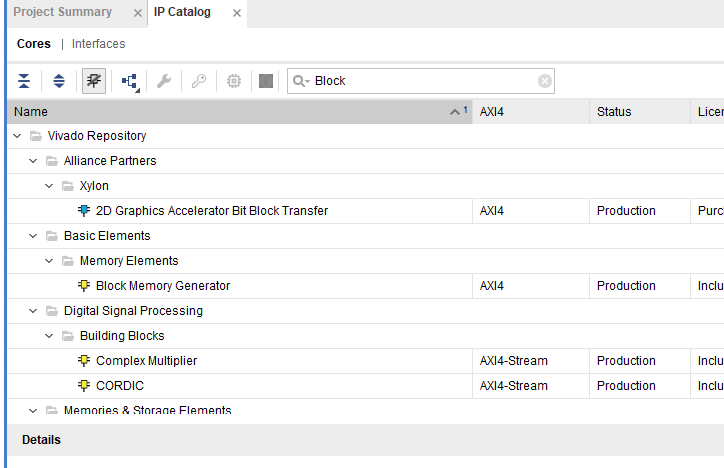


图2.3 选择IP类型

## 2.3设置RAM参数

Block Memory Generator共有四类设置，分别为Basic、端口设置、其他设置、Summary：

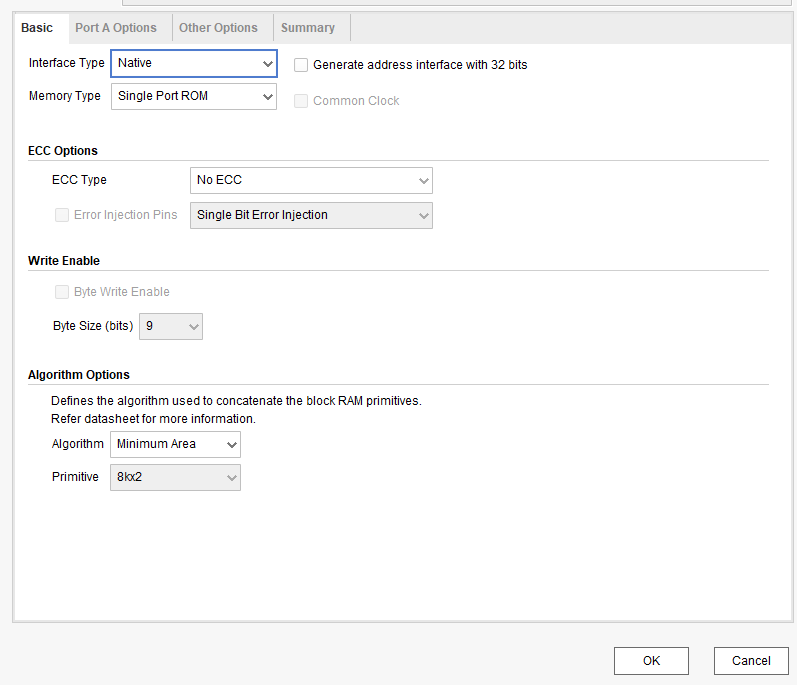


图2.4 Memory 定制界面

其中Basic需要设置存储器类型，Interface Type需选择Native，选中Generate address interface with 32bits，将地址长度设置为32位，Memory Type根据实验要求选择，其他选项无需设置。

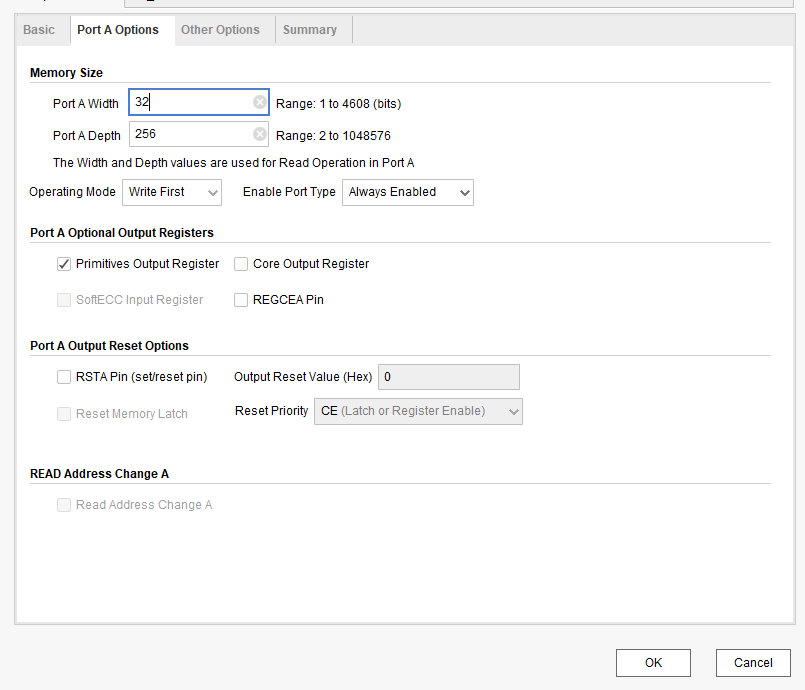


图2.5 Memory Port A 参数设定

端口设置需要设置**数据字宽度**及**阵列深度，**根据实验要求，字宽均为32位，阵列深度需根据需求自定义，但不可超过155520字。

写数据端口默认开启写使能，读数据端口默认不开启，可根据自己需求选择Enable Port Type。

## 2.4设置初始化数据

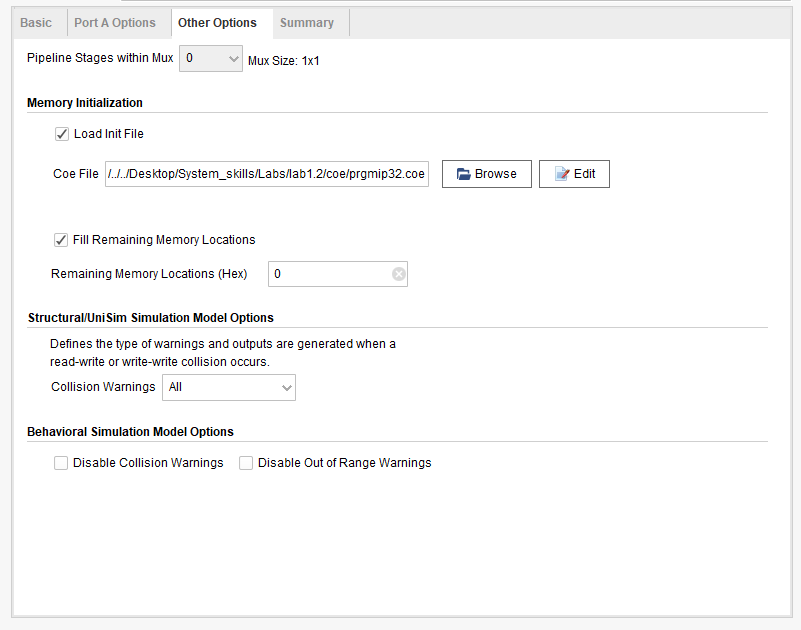


图2.6 装载初始化数据

其他设置主要用于加载coe文件，在上图中，需要勾选“Load Init File”,并选中需要装载的初始化文件(.coe 文件)。 .coe 文件为 Vivado 中存储器初始化文件，其格式如下：

1 memory\_initialization\_radix = 16;

2 memory\_initialization\_vector =

3 24010001

4 00011100

5 ……

第一行指定了初始化数据格式，此处为 16 进制，也可以设置为 2 进制。第二行说明从第三行开始为初始化的数据向量，由于宽度为 32 位，故一个初始化向量为 32 位数据。初始化向量之间必须用空格或换行符隔开，此处使用换行符，故一行为一个初始化向量。初始化数据会从 RAM 中的 0 地址处开始依次填充。当初始化数据格式设置为 2 进制时，后续的初始化向量需要用二进制编写。

这里只需要注意一个问题，Fill Remaining Memory Locations需要选中，以防读数据操作时，地址超过coe文件已有数据范围，导致异常。

# 3 Verilog不同实现方式

验证实验给出两种不同的组合逻辑实现方式：

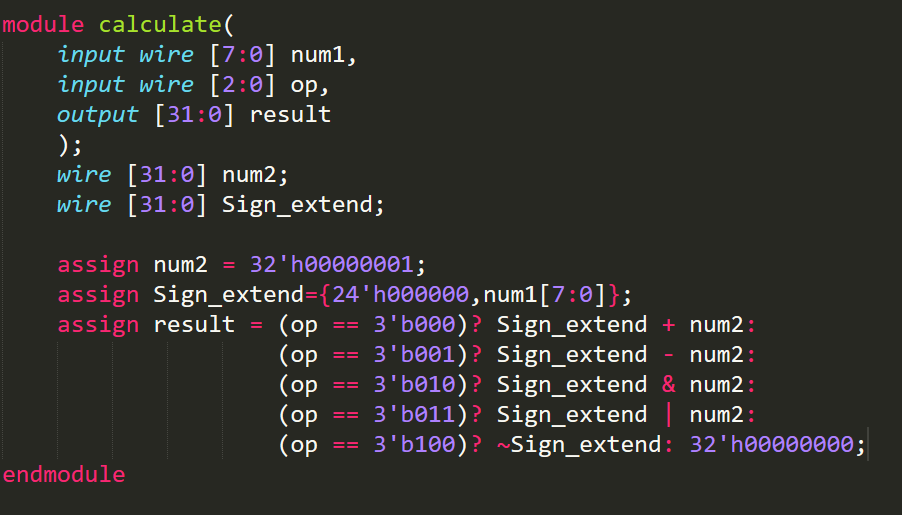


图3.1

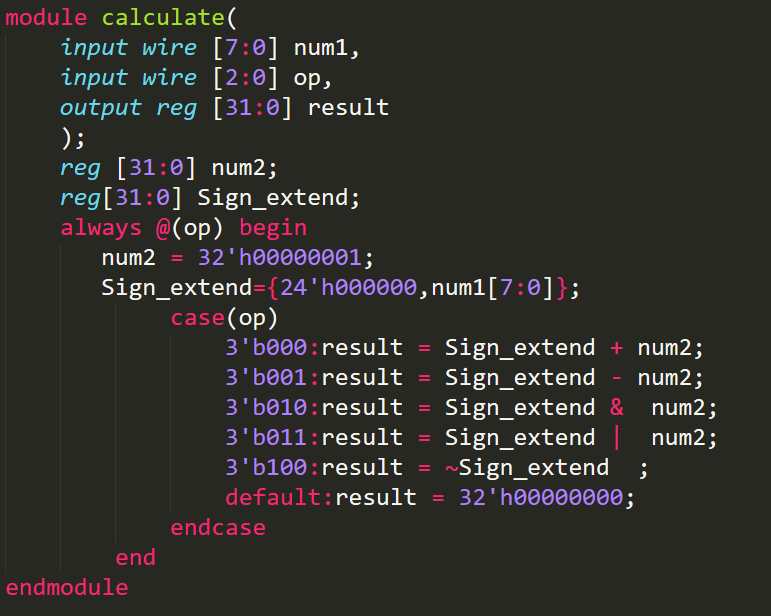


图3.2

二者区别对比如下:

表3.1

|  |  |  |
| --- | --- | --- |
|  | assign方式 | always方式 |
| result | Wire | Reg |
| num2 | Wire | Reg |
| Sign\_extend | Wire | Reg |

Always即可实现组合逻辑，也可实现时序逻辑：

         （1）always @（a or b or c）或always@（\*）形式的，即不带时钟边沿的，综合出来还是组合逻辑；

         （2）always @（posedge clk）形式的，即带有边沿的，综合出来一般是时序逻辑，会包含触发器（Flip-Flop）

观察图1.4中always的触发信号op，可以得知其为组合逻辑。在这里，给出两种方式在FPGA资源及编程难度上的对比。

表3.2

|  |  |  |
| --- | --- | --- |
|  | Assign方式 | Always方式 |
| 资源占用 | 使用wire变量，综合得到的组合逻辑全部由导线构成，不占用FPGA资源 | 由于内部赋值与输出都需要使用reg变量，同为组合逻辑，需占用一定资源 |
| 编程及理解难度 | 面向信号进行状态描述，一般需要考虑单个信号在所有状态下的可能性，需要完全理解后进行编程。  阅读者理解难度大。 | 面向实验需求描述，always模块内书写方式与高级语言相仿，可以按照实验要求逐一列举。  阅读者理解较容易。 |

表1.3

**注意:Imagination及龙芯高校开源计划代码均采取assign方式，实验推荐使用assign方式实现组合逻辑。设计实验中新增内容使用assign方式增加输出信号更为简单。**

# 附录A Nexys4 DDR开发板基本信息

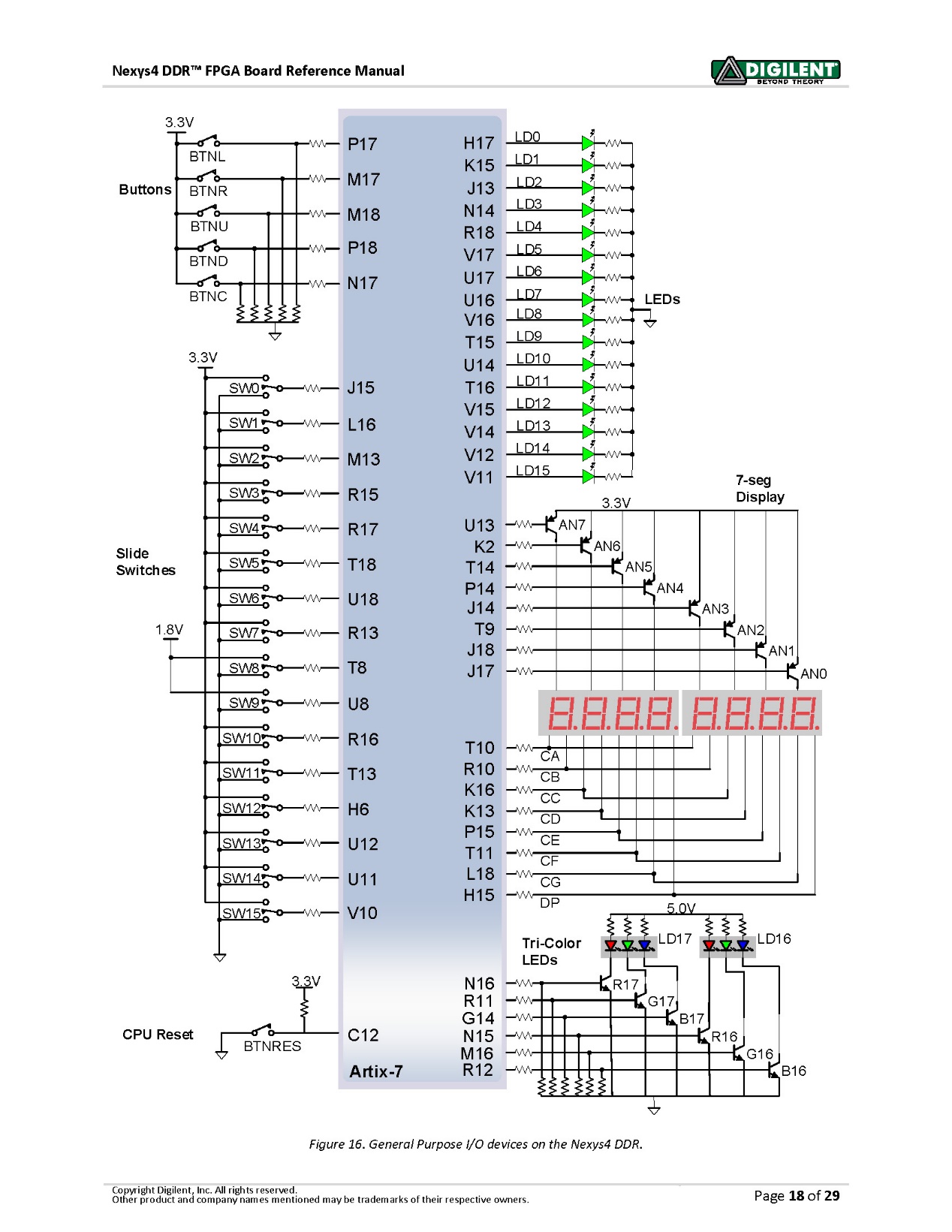


|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **描述** | **序号** | **描述** |
| **1** | 选择供电跳线 | **13** | FPGA 配置复位按键 |
| **2** | UART/ JTAG 共用 USB 接口 | **14** | CPU 复位按键 (用于软核) |
| **3** | 外部配置跳线柱(SD / USB) | **15** | 模拟信号 Pmod 端口(XADC) |
| **4** | Pmod 端口 | **16** | 编程模式跳线柱 |
| **5** | 扩音器 | **17** | 音频连接口 |
| **6** | 电源测试点 | **18** | VGA 连接口 |
| **7** | 16 个 LED | **19** | FPGA 编程完成 LED |
| **8** | 16 个拨键开关 | **20** | 以太网连接口 |
| **9** | 8 位 7 段数码管 | **21** | USB 连接口 |
| **10** | 可选用于外部接线的 JTAG 端口 | **22** | (工业用) PIC24 编程端口 |
| **11** | 5 个按键开关 | **23** | 电源开关 |
| **12** | 板载温度传感器 | **24** | 电源接口 |

# 附录B Nexys4 DDR引脚说明

**100MHz时钟 E3**

**按键、拨码管、LED、七段数码管、Reset:**

****

# 附录C 七段数码管的使用

Nexys4 DDR实验板上有两个4位带小数点的七段数码管，图1-6显示了它们与主芯片的连接方式。其中A7~A0是数码管8个位的使能信号，而CA~CG/DP则对应各个位上七个段以及小数点的触发信号。需要注意的是，使能信号和触发信号都是低电平触发的。

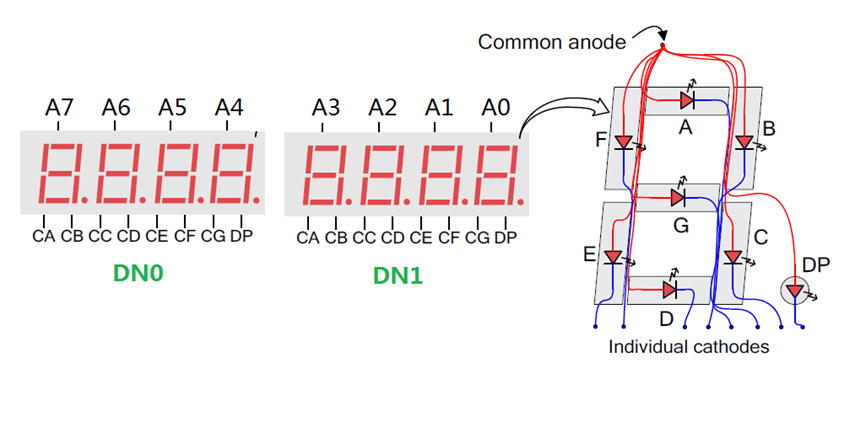
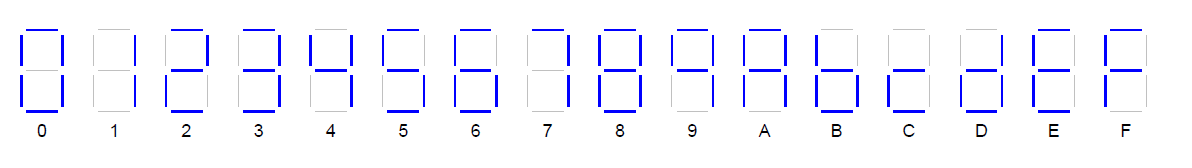
****

图1-7 以数码管中最右侧的A0数码管为例说明了Nexys4 DDR板卡上的7-段数码管的连接方式。8个位中的各个相应的段及小数点分别连接到一组低电平触发的引脚上，他们被称为CA、CB、CC、…、CG、DP，其中，CA接到这8个数码管中每一个数码管A段的负极,CB接到这8个数码管中每一个数码管B段的负极，以此类推。

此外，每一个数码管都有一个使能信号**A[7:0]**。**A[7:0]**通过一个反相器接到对应数码管的每一个段的正极上。比如说，只有到A[0]为0的时候，最右侧数码管的显示才会受到CA…CG这几个信号的驱动。

图1-8中列出了数码管显示0到F时点亮的段。比如说在显示数字0的时候，除了中间的G段外其他的段都被点亮了。而数字1只点亮了B段和C段。



要想让每个数码管显示不同的数字，使能信号（A[7:0]）和段信号（CA…CG）必须依次地被持续驱动，数码管之间的刷新速度应该足够快这样就看不出来数码管之间在闪烁。举个例子，如果想在数码管0上显示数字3而数码管1上显示数字9，可以先把CA…CG设置为显示数字3，并拉低A[1]信号，然后再把CA…CG设置为显示数字9并拉高A[1]拉低A[2]。刷新频率可以设置为2ms刷新一次，这样人眼就看不出闪烁了。