版本

更新记录		文档名	实验指导书_Lab1	
		版本号	0.2	
		创建人	计算机组成原理教学组	
		创建日期	2017/10/8	
更新历史				
序号	**更新日期**	**更新人**	**版本号**	**更新内容**
1	2017/10/8	吕昱峰	0.1	初版，存储器与 ALU 实验
2	2018/10/8	吕昱峰	0.2	去除无用的章节前述，改为实验开始前的准备内容；删除存储器实验冗杂的拨码要求；删除验证性实验，将 0.1 版本中验证性实验改为设计实验。

文档错误反馈：lvyufeng@cqu.edu.cn

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 Vivado2018.1 的电脑一台；
   a) 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。
   b) 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；
   a) 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。

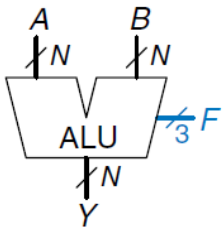


图 1.1

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

$F_{2:0}$	功能	$F_{2:0}$	功能
000	A + B(Unsigned)	100	$\bar{A}$
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：

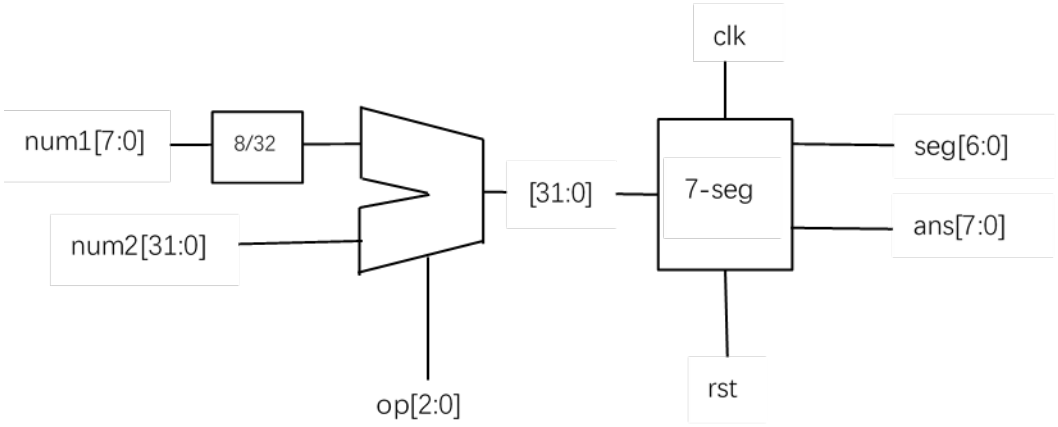


图 1.2 ALU 实验原理图

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：

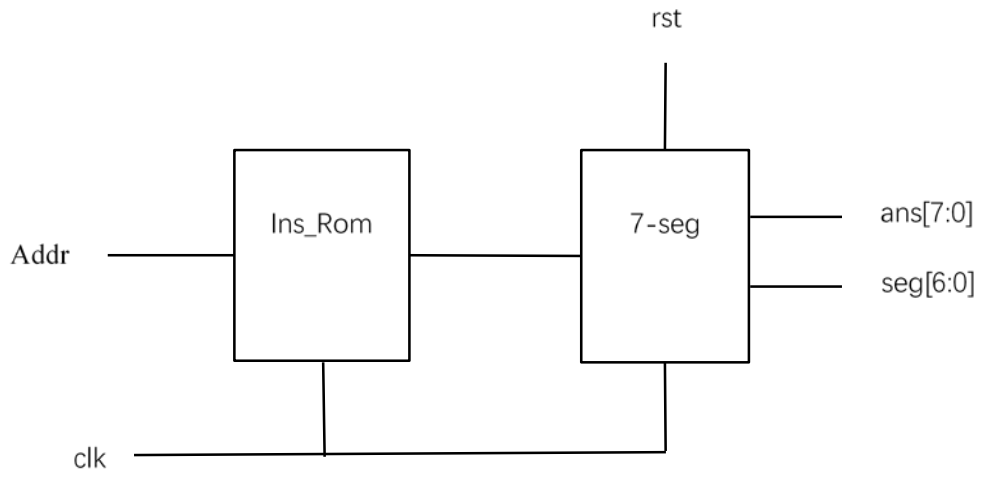


图 1.3

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”：

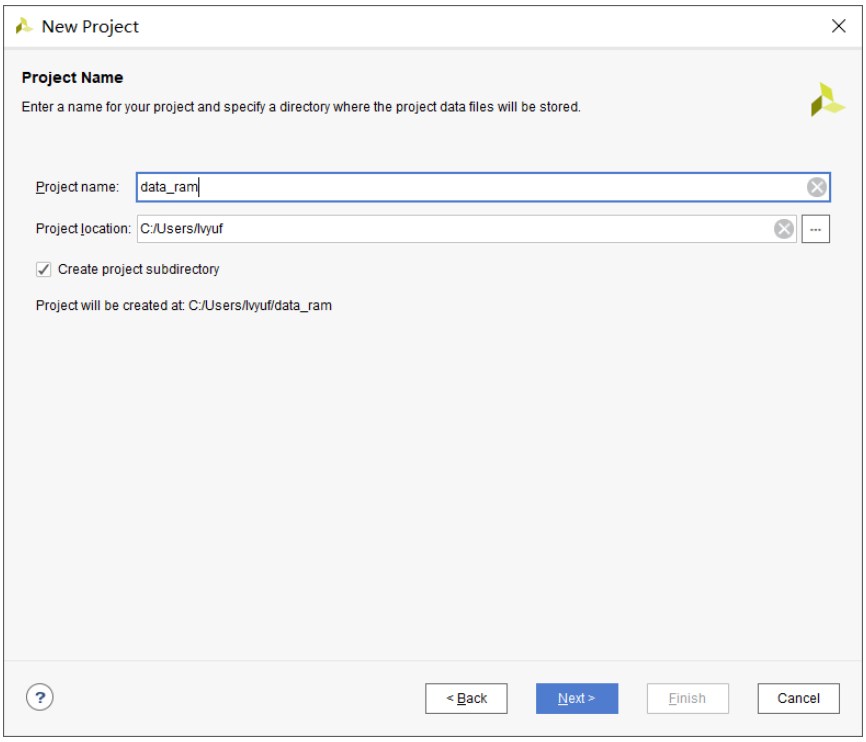


图 2.1 新建工程 data_ram

2.2 新建 IP

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

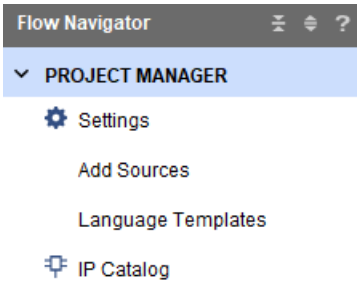


图 2.2 打开 IP 目录

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

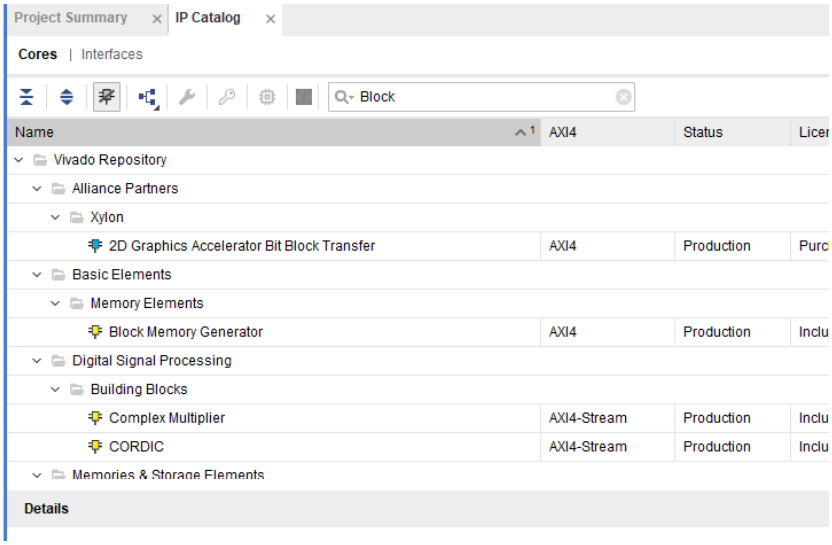


图 2.3 选择 IP 类型

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：

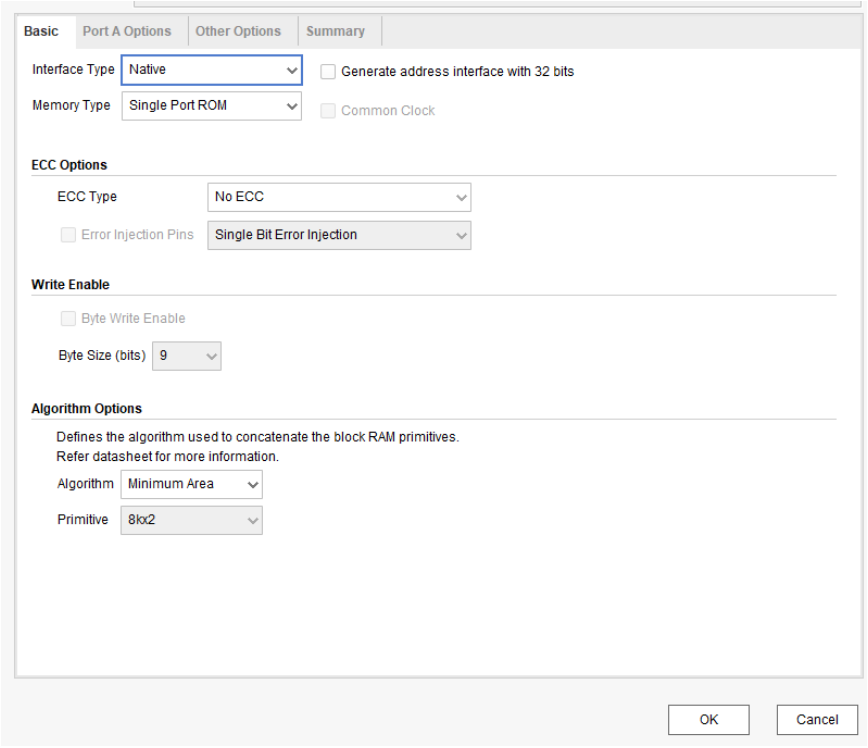


图 2.4 Memory 定制界面

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。

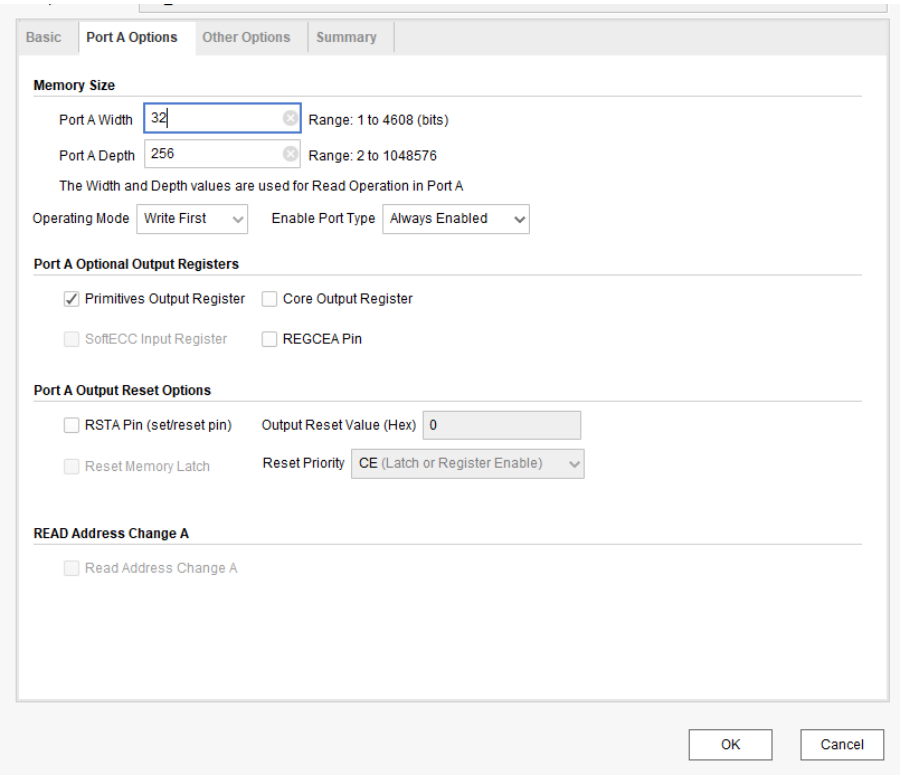


图 2.5 Memory Port A 参数设定

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

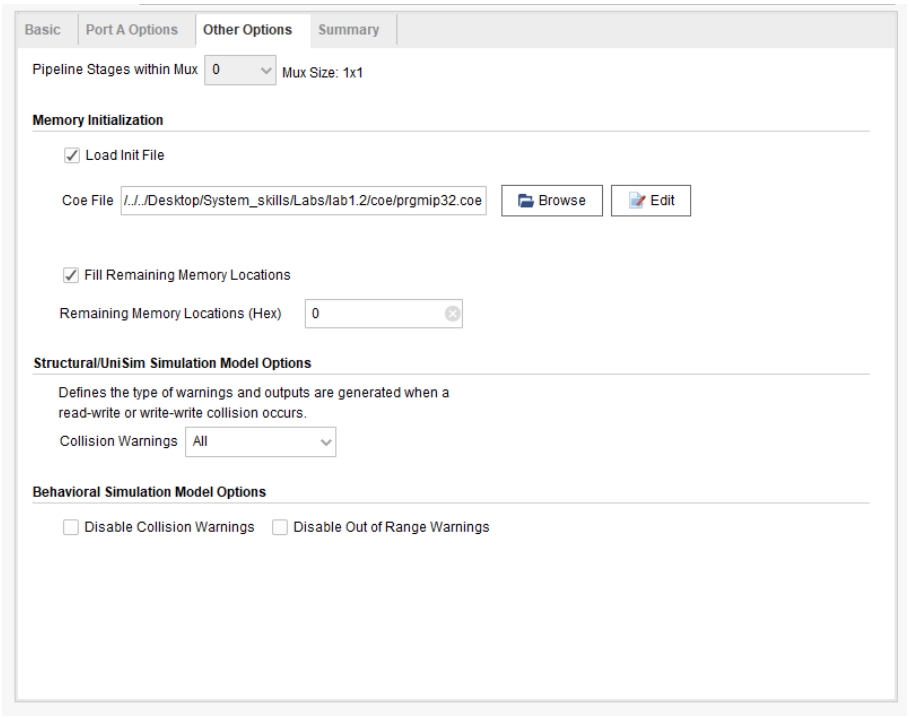


图 2.6 装载初始化数据

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：

```
module calculate(
    input wire [7:0] num1,
    input wire [2:0] op,
    output [31:0] result
    );
    wire [31:0] num2;
    wire [31:0] Sign_extend;

    assign num2 = 32'h00000001;
    assign Sign_extend={24'h000000,num1[7:0]};
    assign result = (op == 3'b000)? Sign_extend + num2:
                    (op == 3'b001)? Sign_extend - num2:
                    (op == 3'b010)? Sign_extend & num2:
                    (op == 3'b011)? Sign_extend | num2:
                    (op == 3'b100)? ~Sign_extend: 32'h00000000;
endmodule
```

图 3.1

```
module calculate(
    input wire [7:0] num1,
    input wire [2:0] op,
    output reg [31:0] result
    );
    reg [31:0] num2;
    reg[31:0] Sign_extend;
    always @(op) begin
       num2 = 32'h00000001;
       Sign_extend={24'h000000,num1[7:0]};
            case(op)
                3'b000:result = Sign_extend + num2;
                3'b001:result = Sign_extend - num2;
                3'b010:result = Sign_extend &  num2;
                3'b011:result = Sign_extend |  num2;
                3'b100:result = ~Sign_extend  ;
                default:result = 32'h00000000;
            endcase
        end
endmodule
```

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。

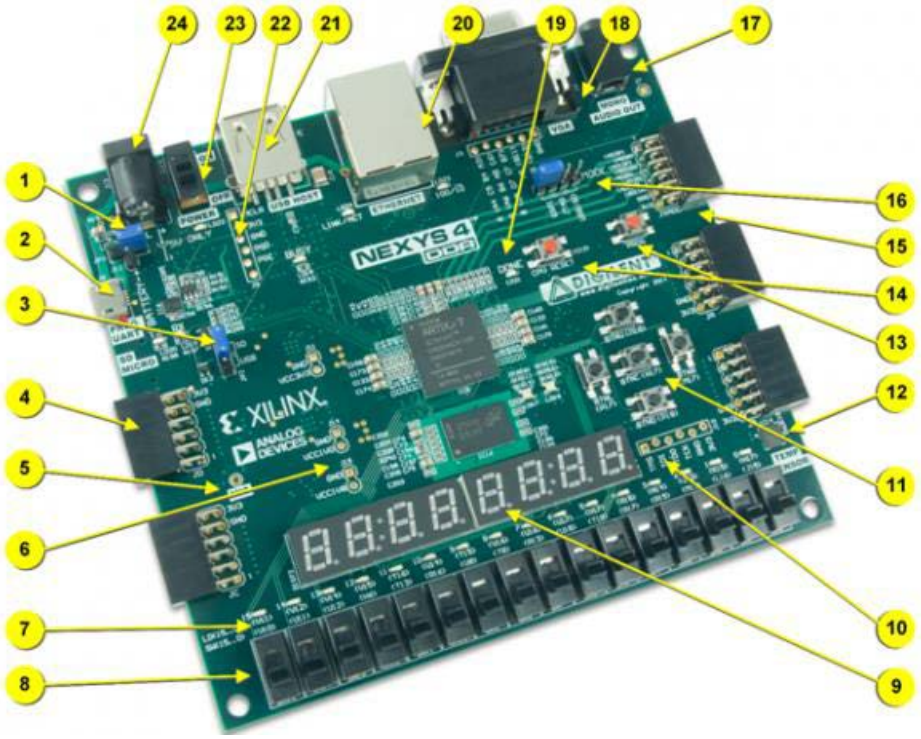
表 3.2

	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:

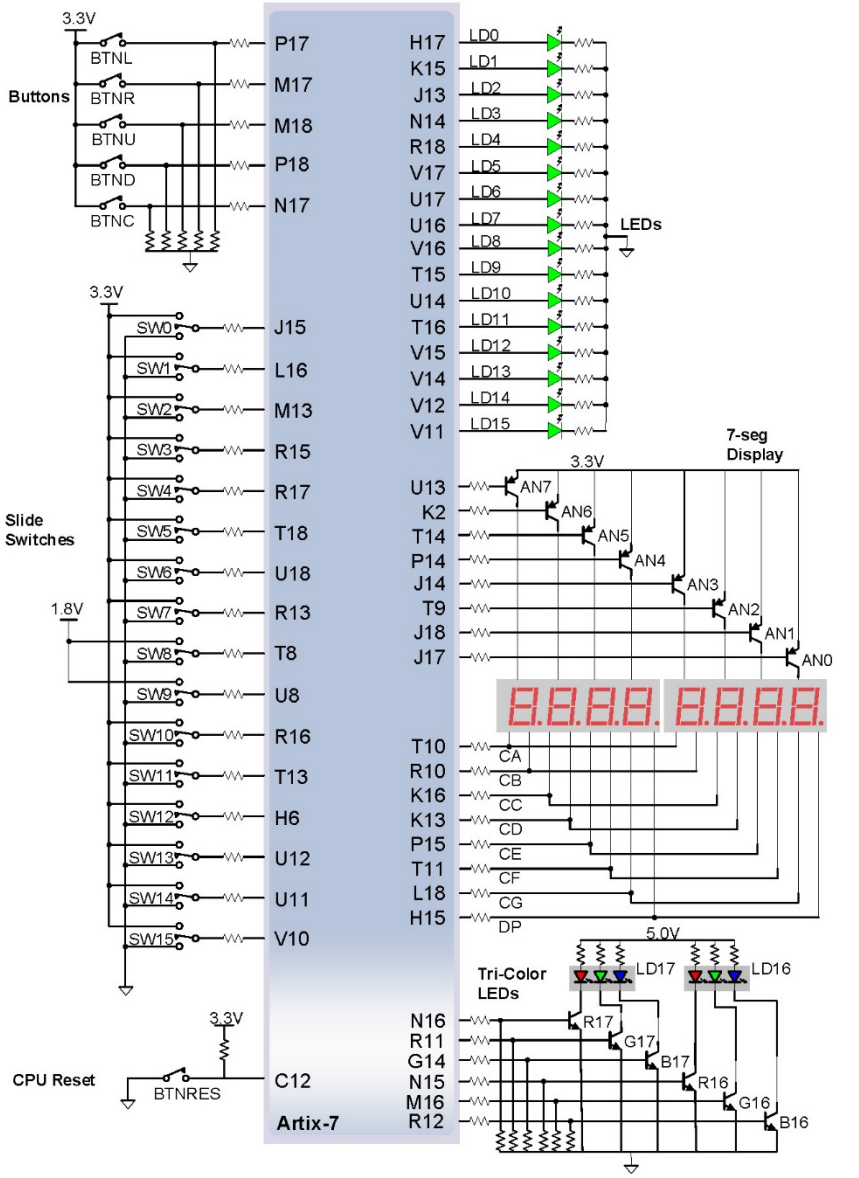


Figure 16. General Purpose I/O devices on the Nexys4 DDR.

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。

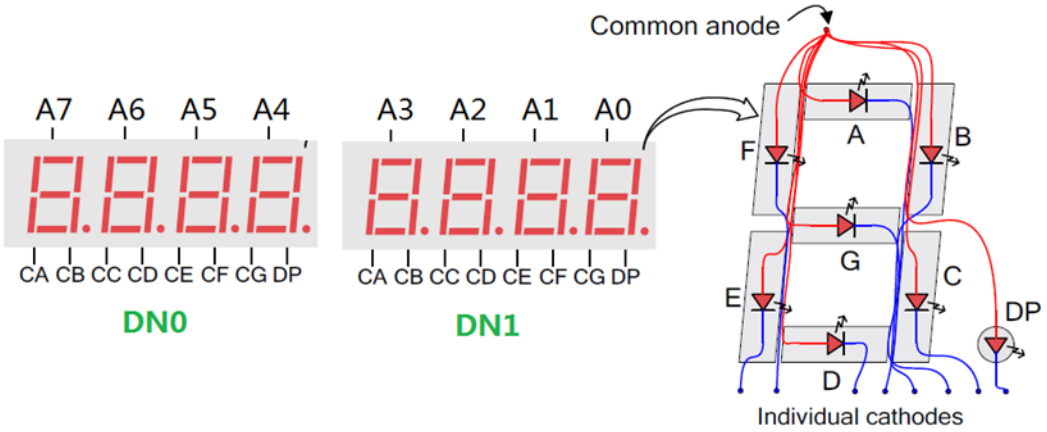
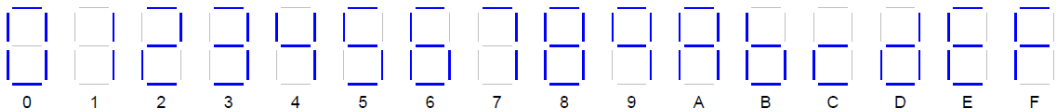


图 1-7 以数码管中最右侧的 A0 数码管为例说明了 Nexys4 DDR 板卡上的 7-段数码管的连接方式。8 个位中的各个相应的段及小数点分别连接到一组低电平触发的引脚上，他们被称为 CA、CB、CC、…、CG、DP，其中，CA 接到这 8 个数码管中每一个数码管 A 段的负极,CB 接到这 8 个数码管中每一个数码管 B 段的负极，以此类推。

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。