版本

更新记录		文档名	实验指导书_Lab1	
		版本号	0.1	
		创建人	计算机组成原理教学组	
		创建日期	2022/1/1	
更新历史				
序号	**更新日期**	**更新人**	**版本号**	**更新内容**
1	2022/1/1	陈颖琪	0.1	初版，存储器例化与 ALU 实验

文档错误反馈：yingqichen@sjtu.edu.cn

1 实验1 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 Design Suite - HLx Editions v2020.2 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的版本。
   b) 若未曾安装 Vivado，请到官网 www.xilinx.com 下载并安装。
2) 熟悉 Vivado 的 IDE 环境，并能够使用其进行仿真、综合；
   如果对 Vivado 不熟悉，请参考文档“**Vivado Design Flow**”和“**Vivado Design Flow example test**”。
3) 熟悉 EGO1 FPGA开发板(Artix-7)；
   请确保在实验进行前阅读过“**Ego1_UserManual_v2.2**”，“**EGo1使用注意事项v1. 1**”。

1. 1 实验目的

1. 了解算术逻辑单元 ALU 的原理，熟悉并运用 Verilog 语言设计 ALU
2. 了解只读存储器ROM、随机存取存储器 RAM 的原理，掌握调用 Xilinx 库 IP(Block Memory Generator)实例化 ROM、RAM 的方法
3. 设计RISC-V单周期CPU 核心单元ALU、指令存储器，为设计CPU核提供必要基础
4. 熟悉FPGA设计开发流程，熟悉使用EGO1 FPGA开发板的按键、数码管等外设进行板级测试验证的方法

1. 2 实验设备

1. 计算机 1 台(尽可能达到 8G 及以上内存)；
2. EGO1 Xilinx FPGA实验开发板；
3. Xilinx Vivado 开发套件(2020.2 版本)。

1. 3 实验任务

本次实验包含两部分，包括 ：
1、ALU 设计及板上验证。
2、RAM 实例化板上验证

1.3.1 ALU 设计实验

图 1.1 给出了具有两个 *N* 位输入和 一个*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 aluc，执行对应功能后输出 N 位结果 aluout，和一个1bit标志位zero。aluout=0 时 zero =1，否则 zero = 0。

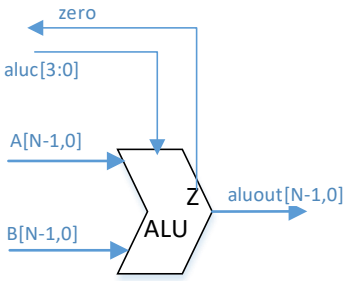


图 1.1

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

$aluc_{3:0}$	功能	$aluc_{3:0}$	功能
0000	A + B(Unsigned)	0001	A<<B
1000	A - B	0101	A>>B(logical)
0111	A AND B	1101	A>>B(arithmetic)
0110	A OR B	0010	B
0100	A XOR B	其他	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。Verilog代码实现方式参考本文第3大部分。

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[3:0]。
2. 内置一个 32 位 A（值为 0x00000005）作为输入到运算器端口 A；
3. 将 板子上sw4~sw0 的5位输入, 经过**无符号扩展** 至 32 位后，输入到运算器的端口 B；
4. 实现9种运算器功能。支持“加、减、与、或、异或、B直通、左移、逻辑右移、算数右移”9种运算。
5. 支持9种运算需要 4 位控制信号（最多支持16 个操作）。将 dip_pin[3:0] 输入到 aluc[3:0] 作为运算器的控制信号；
6. 将计算 32 位结果 aluout 显示到七段数码管(16 进制)。项目结构可参考如下图。
7. 验证表 1.1 中所有功能。
8. 给出 RTL 源程序（.v 文件）

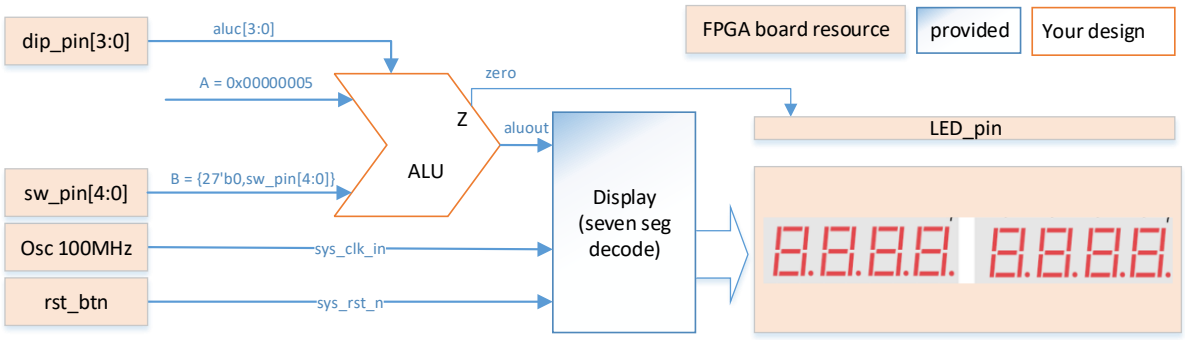


图 1.2 ALU 实验原理图

1.3.2 存储器 IP 实例化实验

本次实验使用 Vivado 的 Block Memory Generator 模拟指令在存储器中的存取过程。实验使用**单端口 ROM**。参阅 2完成指令存储器和数据存储器的例化。

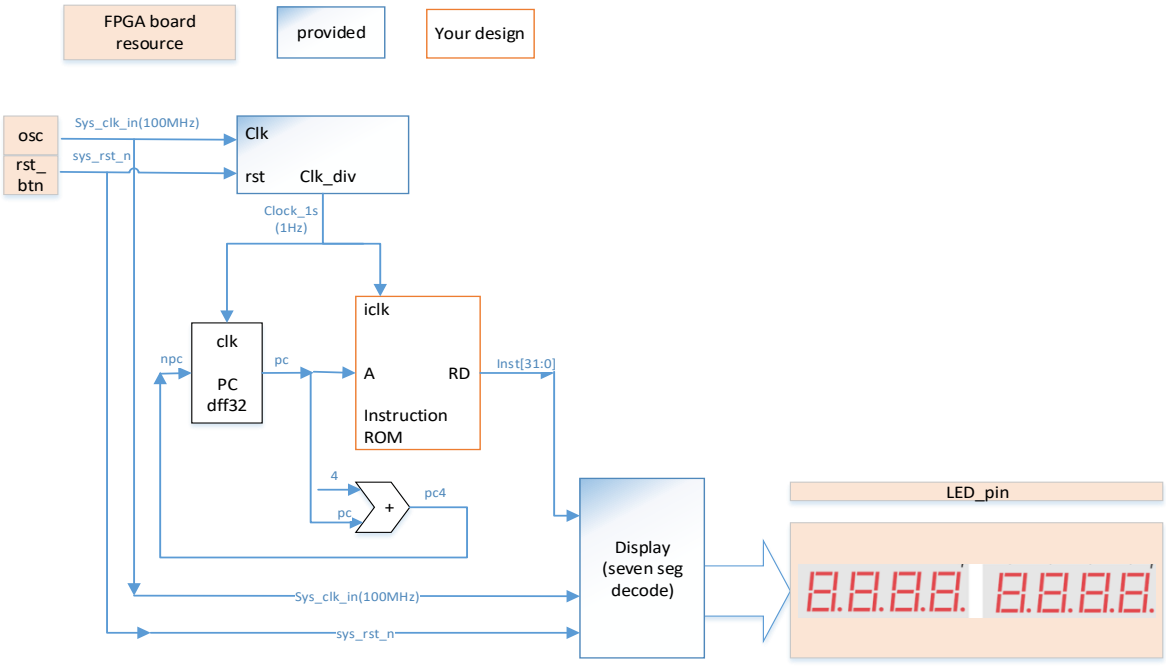


图 1.3 IROM实验原理框图

实验要求：

1）使用 Block Memory Generator 例化一个32位宽，64单元的单端口指令ROM，并将指令 coe 文件加载，存放CPU指令；

2）使用 Block Memory Generator 例化一个32位宽，32单元的单端口数据RAM，存放CPU数据；

注意：RISC-V架构支持64bit或32bit数据操作。本次实验仅要求支持32bit宽。即CPU内核仅对32bit宽数据进行操作，内部寄存器堆和存储器数据位宽均设定为32。

1. 4 实验环境

以下表格中红色部分需自行实现或者在实验发布包中提供了供参考不完整的部分，需要补充完整，黑色部分于实验发布包中提供完整代码。

1.4.1 ALU 实验

表 1.2

|--alu_test_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	七段数码管显示驱动模块，已提供。
|---------- seg7.v	七段数码管显示译码模块，已提供。
|--sc_computer_iotest.xdc	综合实现时的管脚约束文件，已提供。

1.4.2 存储器例化

参阅本文第2章完成指令和数据存储器的例化

2 调用 Xilinx 库 Block Memory Generator 方法

2.1 新建工程

新建一个工程 data_ram，注意：建立工程时，必须选择 EGO1 FPGA 开发板对应的 FPGA 器件型号： xc7a50tcsg324-1

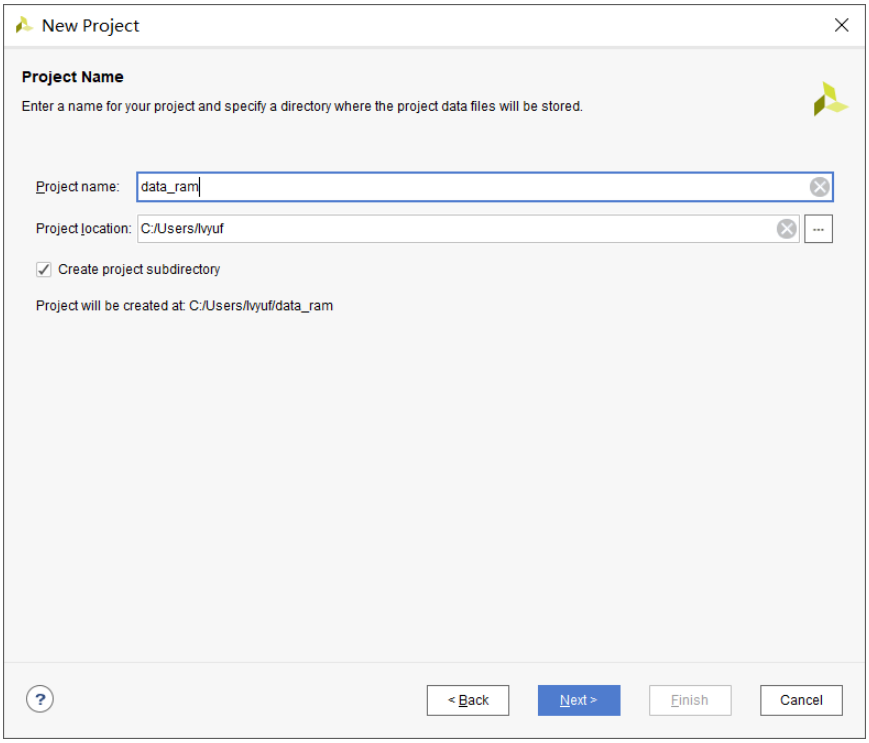


图 2.1 新建工程 data_ram

2.2 新建 IP

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

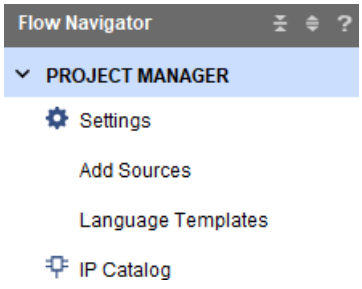


图 2.2 打开 IP 目录

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

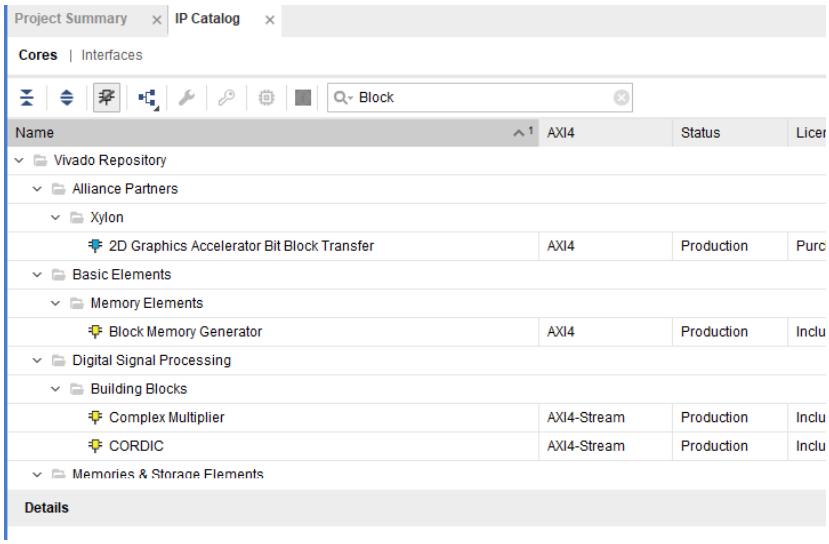


图 2.3 选择 IP 类型

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：

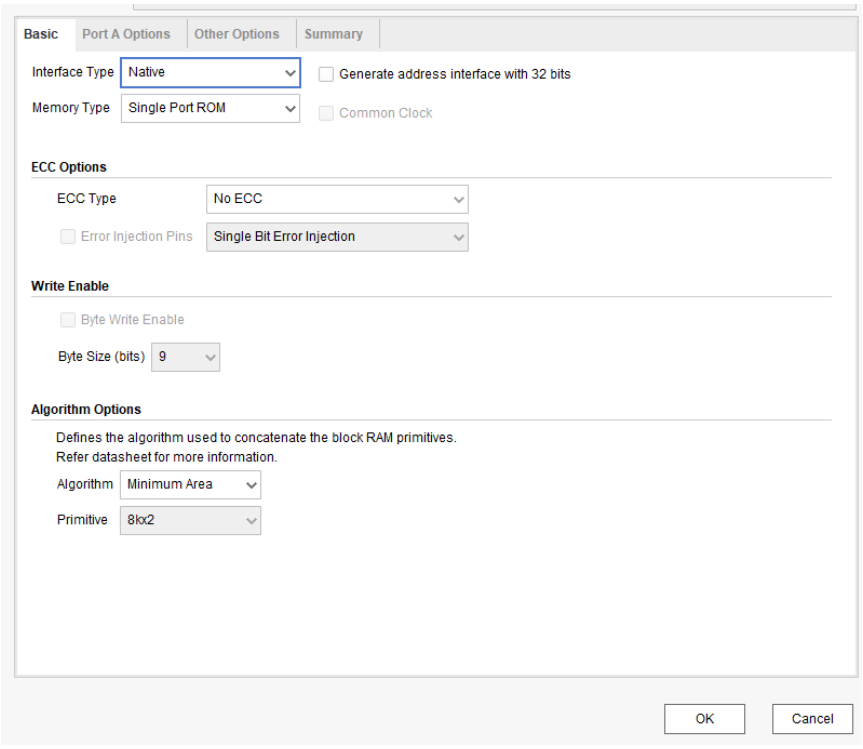


图 2.4 Memory 定制界面

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

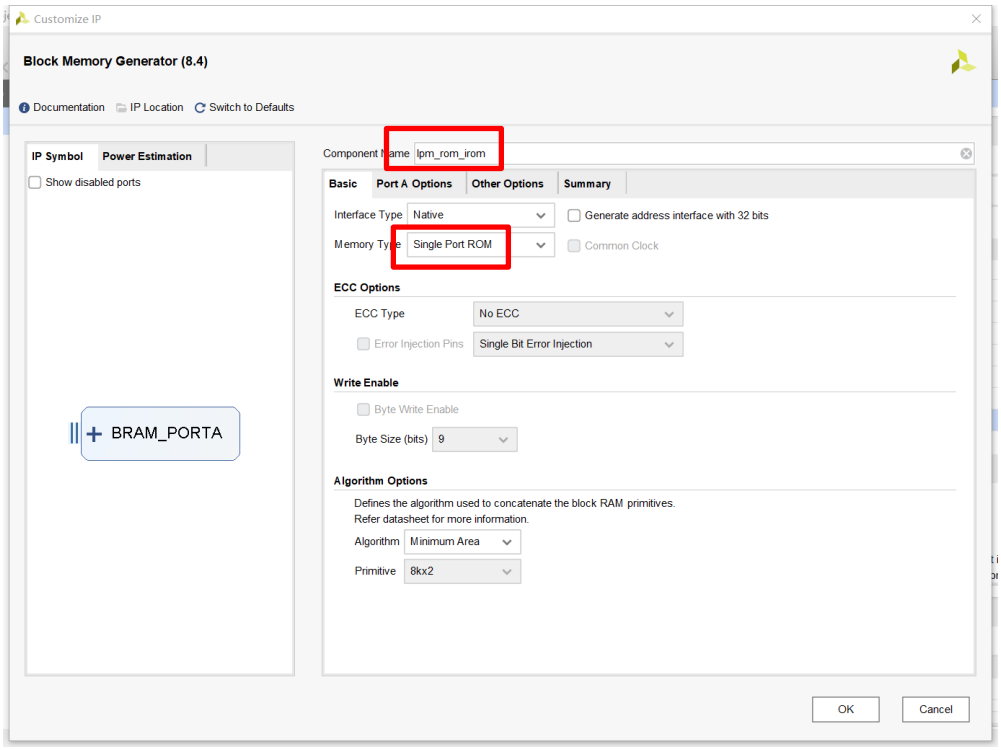
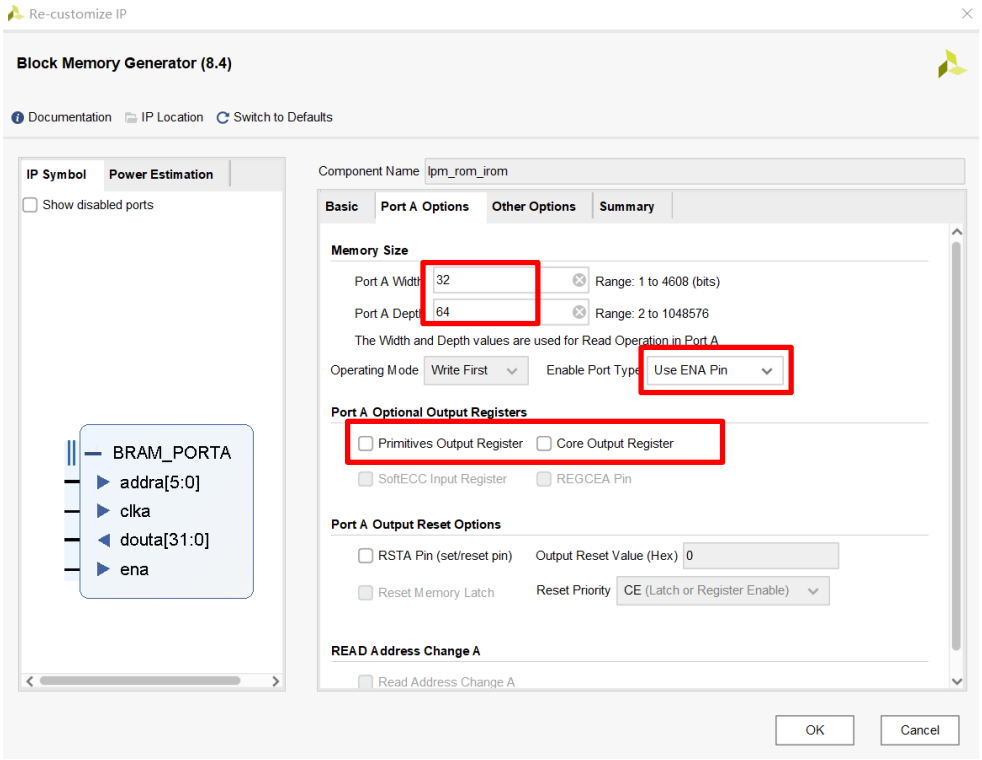


图 2.5 Memory Port A 参数设定

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。可以自己定义IP名字。用提供的irom.coe文件作为初始值。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。 输出寄存器Primitives Output Register、Core Output Register不勾选。此处是以32位宽64单元rom， IP命名为lpm_rom_irom，初始值文件命名lpm_rom_irom.coe为例。



2.4 设置初始化数据

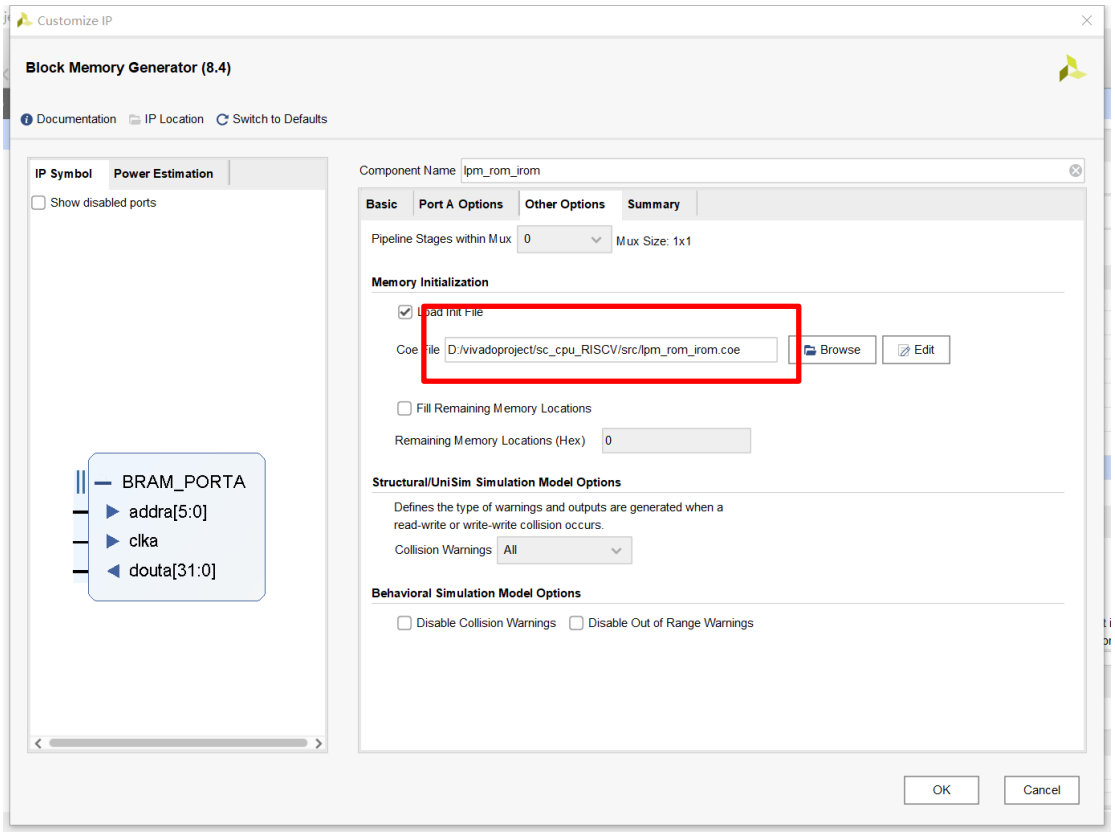


图 2.6 装载初始化数据

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

实验给出两种不同的组合逻辑实现方式：

```
wire z;
wire [31:0] s;
assign  aluout = (aluc == 4'b0000)? alua + alub:
                 (aluc == 4'b1000)? alua - alub:
                 (aluc == 4'b0111)? alua & alub:
                 (aluc == 4'b0110)?            :
                 (aluc == 4'b0100)?            :
                 (aluc == 4'b0010)? alub       :
                 (aluc == 4'b0001)?            :
                 (aluc == 4'b0101)?            :
                 (aluc == 4'b1101)?            :
                 0;
assign z = (aluout == 0);
```

图 3.1

```
1  reg [31:0] s;
2  reg        z;
3  always @ (alua or alub or aluc)
4      begin
5          casex (aluc)
6              4'b0000: aluout = alua + alub;                    //0000 ADD
7              4'b1000: aluout = alua - alub;                    //1000 SUB
8              4'b0111: aluout = alua & alub;                    //0111 AND
9              4'b0110: aluout =                          //0110 OR
10             4'b0100: aluout =                          //0100 XOR
11             4'b0010: aluout = alub;                       //0010 LUI:{lui_imm ,12'b0}
12             4'b0001: aluout =                          //0001 SLL: rd <- (rs1 << rs2)
13             4'b0101: aluout =                          //0101 SRL: rd <- (rs1 >>rs2) (logical)
14             4'b1101: aluout =                          //1101 SRA: rd <- (rs1 >> rs2) (arithmetic)
15             default: aluout = 0;
16         endcase
17         if (aluout == 0 )  z = 1;
18            else z = 0;
19     end
```

图 3.2

二者区别对比如下:

表 3.1

	assign 方式	always 方式
s	Wire	Reg
z	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 3.2 中 always 的触发信号 ，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。

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。

4 实验报告要求

实验报告应简明扼要，重点突出。具体内容包括以下几部分：

1．实验目的。

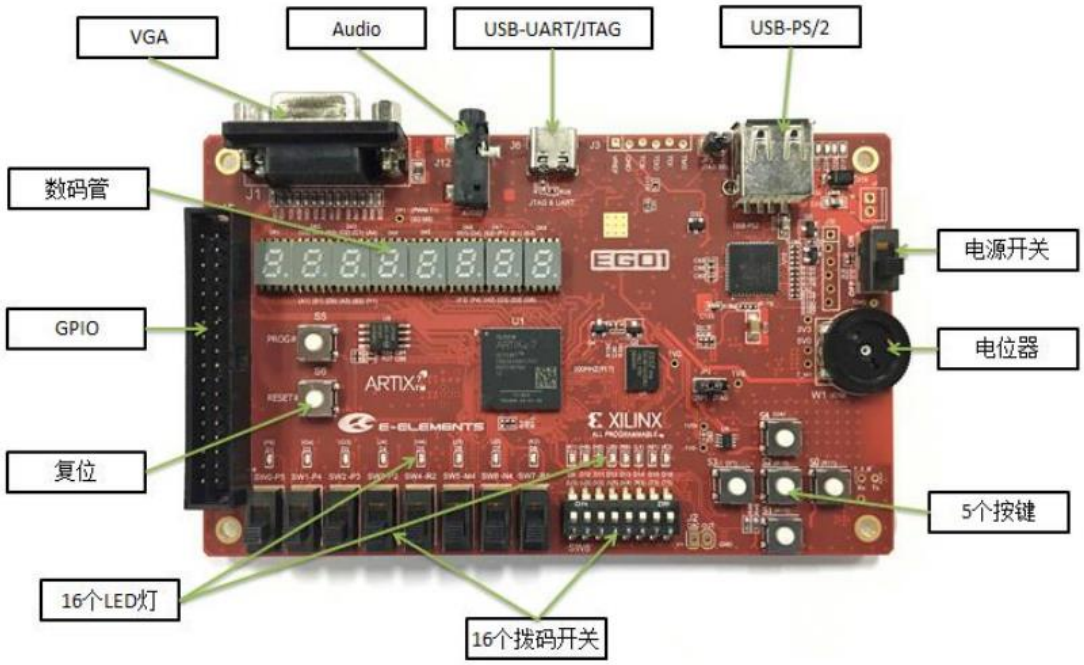
2．本次实验所用的实验平台、仪器以及其它实验器材和部件等。

3．实验任务。这部分是实验报告最主要的内容。根据任务要求，描述设计原理与思路，设计的实现方式，结构框架等。（不要仅贴整段Verilog 程序代码，而是以表述介绍设计思路为主，如有必要，可贴关键代码段，并加注释）。结合实验原理，描述设计的全过程和实验步骤。

4．实验结果演示总结。板上验证结果可以有视频照片演示的截图，仿真结果可以有仿真波形关键部分的截图和解释，不要只贴截图波形。必须有对应的解释，指出该部分截图要说明什么问题。

5．整理、归纳做完的实验内容并解释相应的实验现象或问题。最后也可总结心得体会。

附录 A　EGO1 开发板基本信息



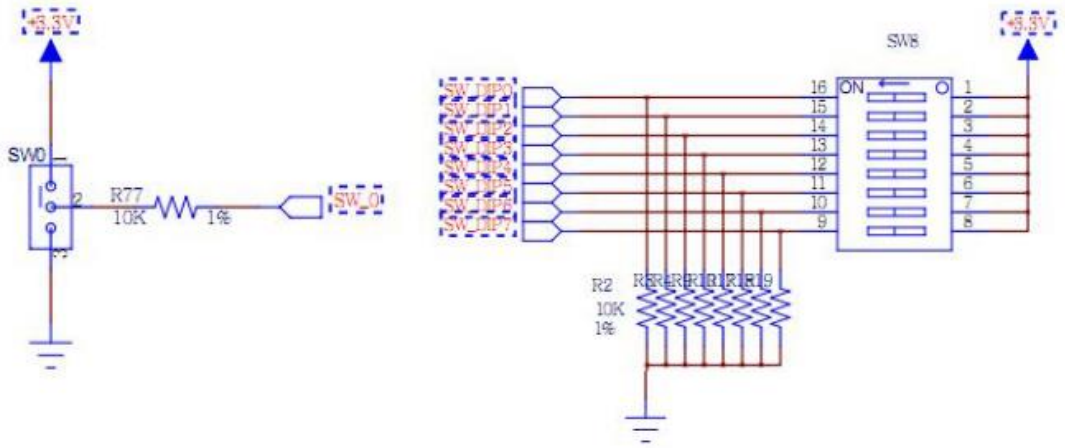
附录 B 引脚说明（参阅Ego1_UserManual_v2.2.pdf）

100MHz 时钟：P17

逻辑复位 RST：S6

拨码开关：

开关包括 8 个拨码开关和一个 8 位 DIP 开关。



管脚约束如下：

名称	原理图标号	FPGA IO PIN
SW0	SW_0	R1
SW1	SW_1	N4
SW2	SW_2	M4
SW3	SW_3	R2
SW4	SW_4	P2
SW5	SW_5	P3
SW6	SW_6	P4
SW7	SW_7	P5
SW8	SW0	T5
	SW1	T3
	SW2	R3
	SW3	V4
	SW4	V5
	SW5	V2
	SW6	U2
	SW7	U3

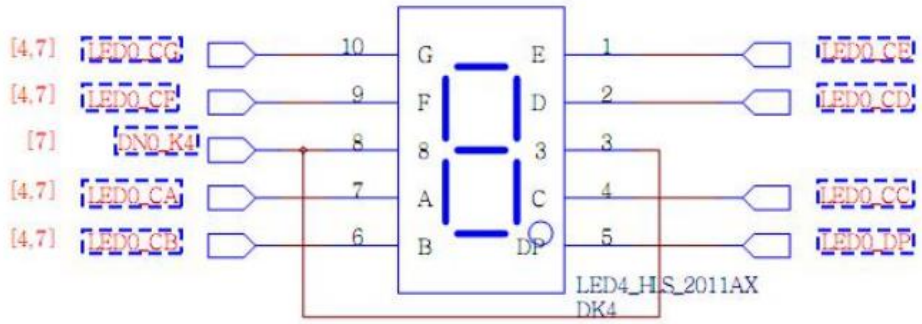
LED灯：

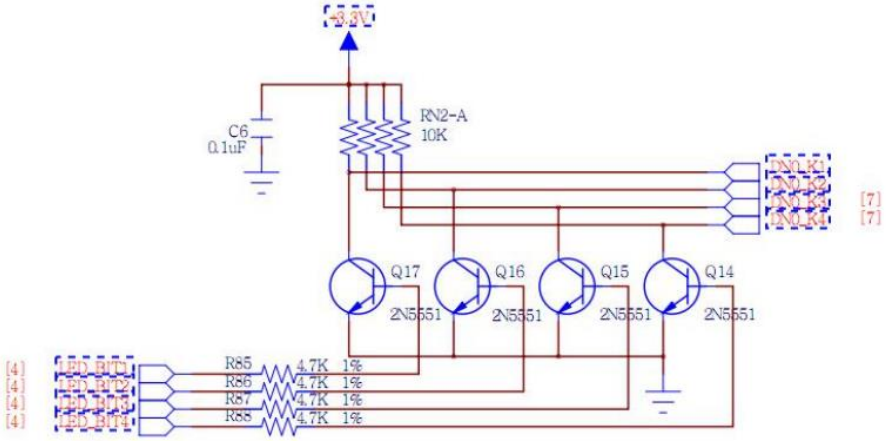
管脚约束如下：

名称	原理图标号	FPGA IO PIN	颜色
D1_0	LED1_0	K3	Green
D1_1	LED1_1	M1	Green
D1_2	LED1_2	L1	Green
D1_3	LED1_3	K6	Green
D1_4	LED1_4	J5	Green
D1_5	LED1_5	H5	Green
D1_6	LED1_6	H6	Green
D1_7	LED1_7	K1	Green
D2_0	LED2_0	K2	Green
D2_1	LED2_1	J2	Green
D2_2	LED2_2	J3	Green
D2_3	LED2_3	H4	Green
D2_4	LED2_4	J4	Green
D2_5	LED2_5	G3	Green
D2_6	LED2_6	G4	Green
D2_7	LED2_7	F6	Green

附录 C　七段数码管的使用

数码管为共阴极数码管，即公共极输入低电平。共阴极由三极管驱动，FPGA 需要提供正向信号。同时段选端连接高电平，数码管上的对应位置才可以被点亮。因此，FPGA 输出有效的片选信号和段选信号都应该是高电平。





管脚约束如下：

名称	原理图标号	FPGA IO PIN
A0	CA0	B4
B0	CB0	A4
C0	CC0	A3
D0	CD0	B1
E0	CE0	A1
F0	CF0	B3
G0	CG0	B2
DP0	DP0	D5
A1	CA1	D4
B1	CB1	E3
C1	CC1	D3
D1	CD1	F4
E1	CE1	F3
F1	CF1	E2
G1	CG1	D2
DP1	DP1	H2
DN0_K1	BIT1	G2
DN0_K2	BIT2	C2
DN0_K3	BIT3	C1
DN0_K4	BIT4	H1
DN1_K1	BIT5	G1
DN1_K2	BIT6	F1
DN1_K3	BIT7	E1
DN1_K4	BIT8	G6

实验板上有两组 4 位带小数点的七段数码管。其中 DN0_K1~ DN0_K4和DN1_K1~ DN1_K4是左右各四个共8个数码管的位使能信号，对应下图中A[7:0]。而A0~G0/DP 0和A1~G1/DP 1则对应左右各四个数码管的七个段以及小数点的触发信号。需要注意信号触发电平，位使能信号是高有效触发。七个段码的A0~G0/DP 0和A1~G1/DP 1是低有效。

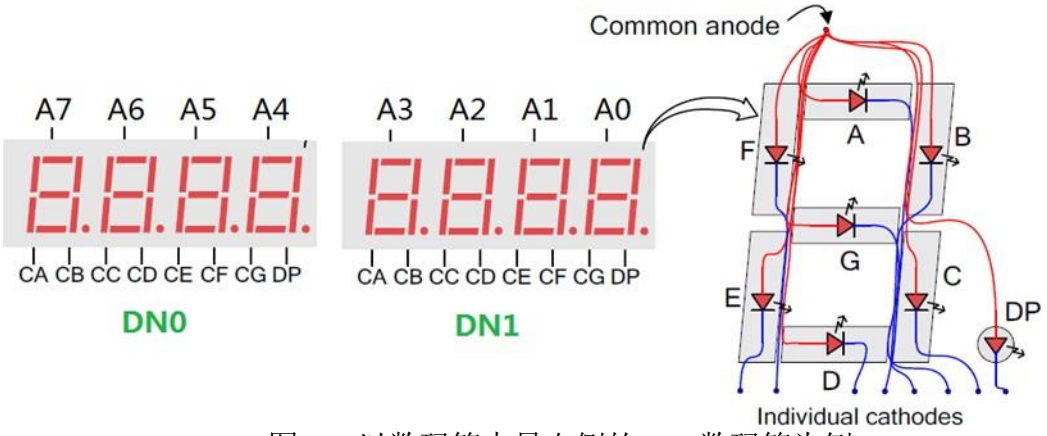


图 1-7 以数码管中最右侧的 A0 数码管为例

图 1-7 以数码管中最右侧的 A0 数码管为例说明了板卡上的 7-段数码管的连接方式。

4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 接到一组数码管中每一个数码管 A 段的负极,CB 接到这 4 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]**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 1 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。

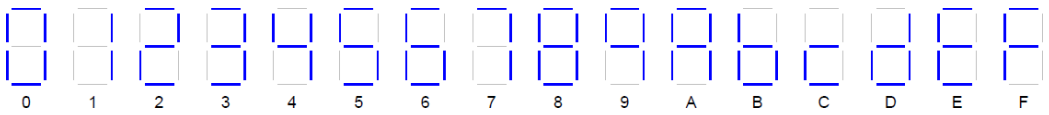


图 1-8 数码管显示 0 到 F 时点亮的段

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。