

Lab0: VIVADO介绍 && 模块封装

Lab0: VIVADO介绍 && 模块封装

任务一: VIVADO工具学习

- 创建工程
- 创建源文件
- 添加仿真文件
- 仿真结果
- RTL分析
- 综合设计
- 引脚约束
- 生成比特流
- 下板验证

任务二: 自定义模块化设计学习

- 创建源文件
- 仿真文件与结果
- 模块封装

任务三: 基本逻辑模块的原理学习、Xilinx IP 的生成

任务1: 基本模块功能

- MUX2T1_5
- MUX2T1_8
- MUX2T1_32
- MUX4T1_5
- MUX4T1_32
- MUX8T1_8
- add_32
- addc_32
- and_32
- Ext_imm16
- nor_32
- or_bit_32
- or_32
- SignalExt_32
- srl_32
- xor_32

任务2: 存储器ROM、RAM的生成及初始化

- ROM_D IP Core
- RAM_B IP Core

任务四: 利用自定义模块构建实验平台

模块代码

- muxctrl
- MUX2T1_5
- MUX4T1_5
- MUX8T1_8

引脚约束

下板验证

思考题

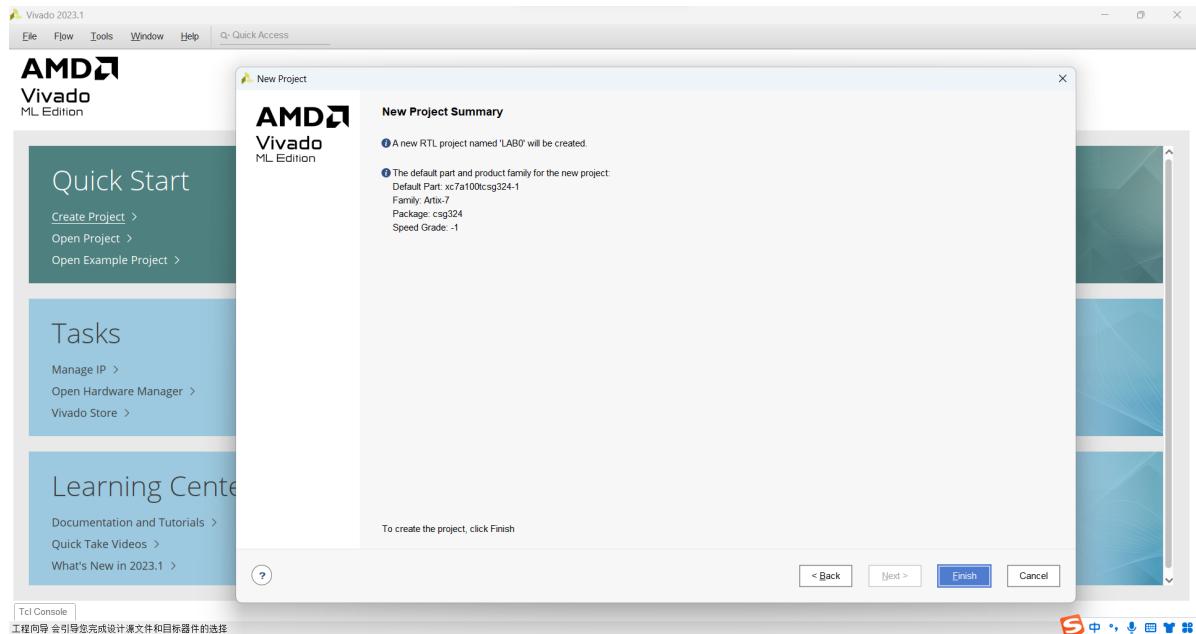
- 一
- 二
- 三
- 四

任务一：VIVADO工具学习

利用VIVADO完成water_LED的全流程设计

创建工程

如下，选择对应参数的板子型号，并创建工程，名为LAB0



创建源文件

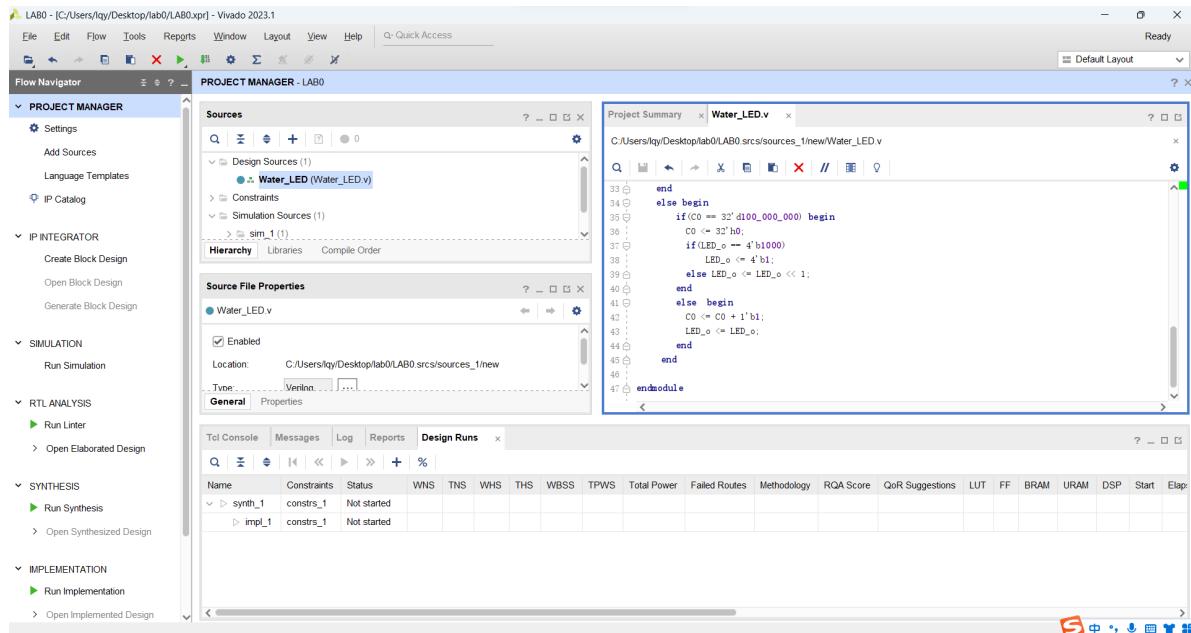
创建 Design Source 名为 water_LED， 并输入如下代码：

```
1 `timescale 1ns / 1ps
2
3 module water_LED(
4     input CLK_i,
5     input RSTn_i,
6     output reg [3:0] LED_o
7 );
8     reg [31:0] c0;
9
10    always @(posedge CLK_i)
11        if(!RSTn_i) begin
12            LED_o <= 4'b1;
13            c0 <= 32'h0;
14        end
15        else begin
16            if(c0 == 32'd100_000_000) begin
17                c0 <= 32'h0;
18                if(LED_o == 4'b1000)
19                    LED_o <= 4'b1;
20                else LED_o <= LED_o << 1;
21            end
22            else begin
23                c0 <= c0 + 1'b1;
24                LED_o <= LED_o;
```

```

25         end
26     end
27
28 endmodule

```



添加仿真文件

根据步骤添加仿真文件，添加以下仿真代码：

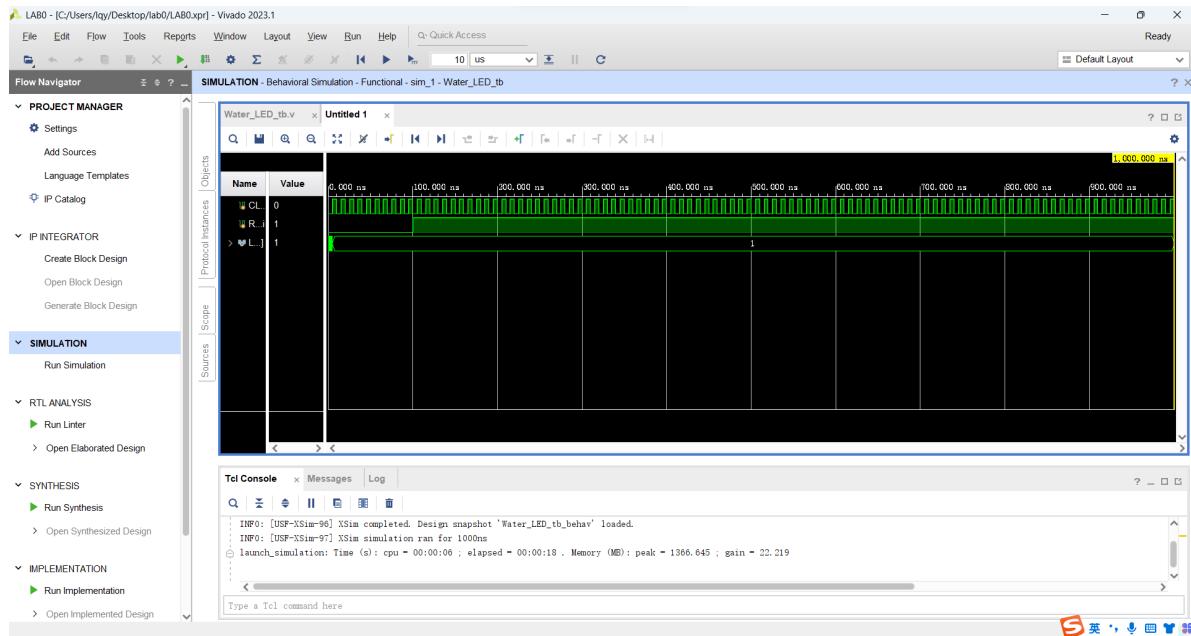
```

1 `timescale 1ns / 1ps
2
3 module Water_LED_tb;
4     reg CLK_i;
5     reg RSTn_i;
6     wire [3:0] LED_o;
7
8     Water_LED Water_LED_U(
9         .CLK_i(CLK_i),
10        .RSTn_i(RSTn_i),
11        .LED_o(LED_o)
12    );
13
14     always #5 CLK_i = ~CLK_i;
15
16     initial begin
17         CLK_i = 0;
18         RSTn_i = 0;
19         #100 RSTn_i = 1;
20
21     end
22 endmodule

```

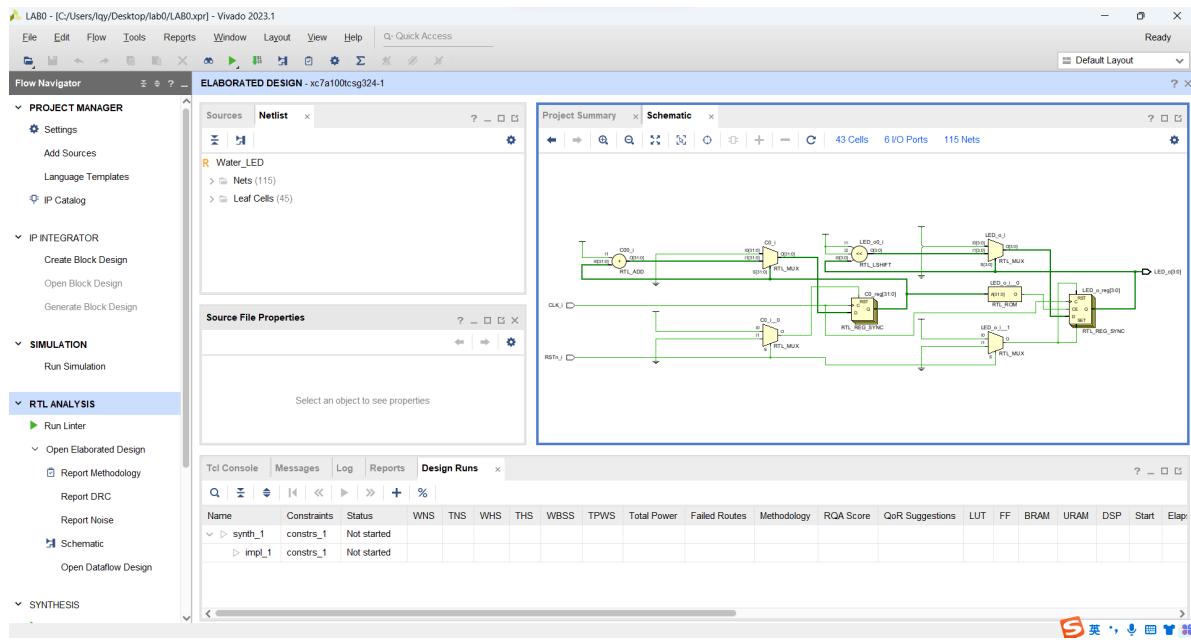
仿真结果

进行仿真后得到如下结果：



RTL分析

点击 RTL Analysis 下的 Schematic 后得到如下结果：



第一次发现VIVADO还能画出电路图

综合设计

根据相关步骤，进行综合设计

引脚约束

采用脚本约束法，根据以下管脚约束脚本进行设置：

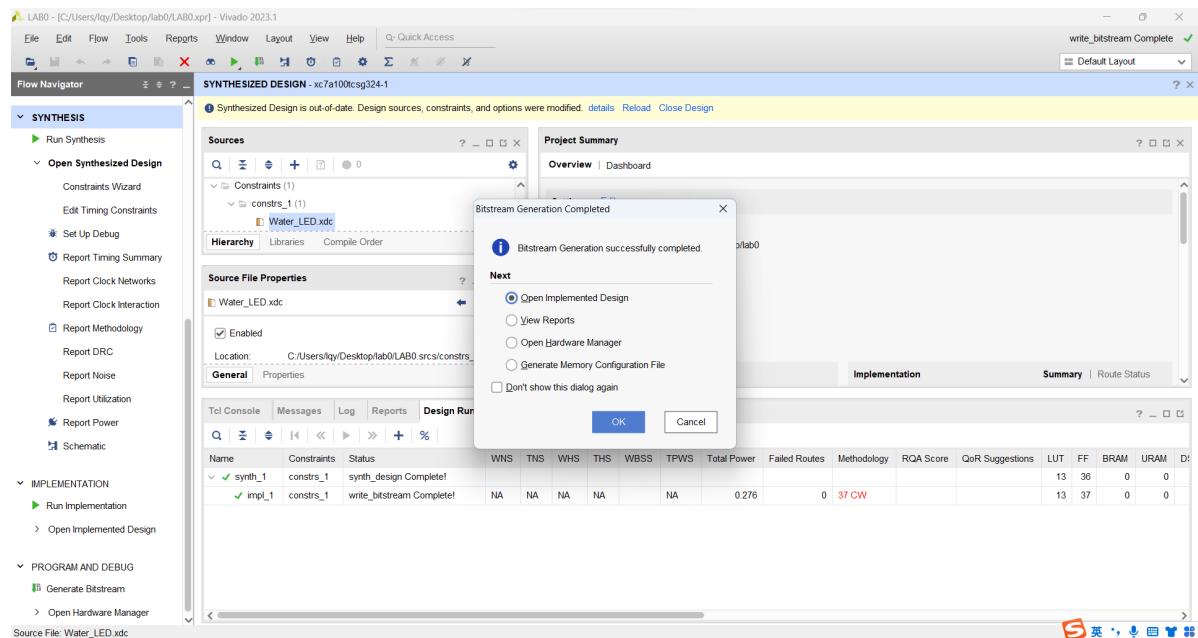
```

1 set_property PACKAGE_PIN C12 [get_ports RSTn_i]
2 set_property IOSTANDARD LVC MOS33 [get_ports RSTn_i]
3 set_property PACKAGE_PIN H17 [get_ports {LED_o[0]}]
4 set_property IOSTANDARD LVC MOS33 [get_ports {LED_o[0]}]
5 set_property PACKAGE_PIN K15 [get_ports {LED_o[1]}]
6 set_property IOSTANDARD LVC MOS33 [get_ports {LED_o[1]}]
7 set_property PACKAGE_PIN J13 [get_ports {LED_o[2]}]
8 set_property IOSTANDARD LVC MOS33 [get_ports {LED_o[2]}]
9 set_property PACKAGE_PIN N14 [get_ports {LED_o[3]}]
10 set_property IOSTANDARD LVC MOS33 [get_ports {LED_o[3]}]
11 set_property PACKAGE_PIN E3 [get_ports CLK_i]
12 set_property IOSTANDARD LVC MOS33 [get_ports CLK_i]

```

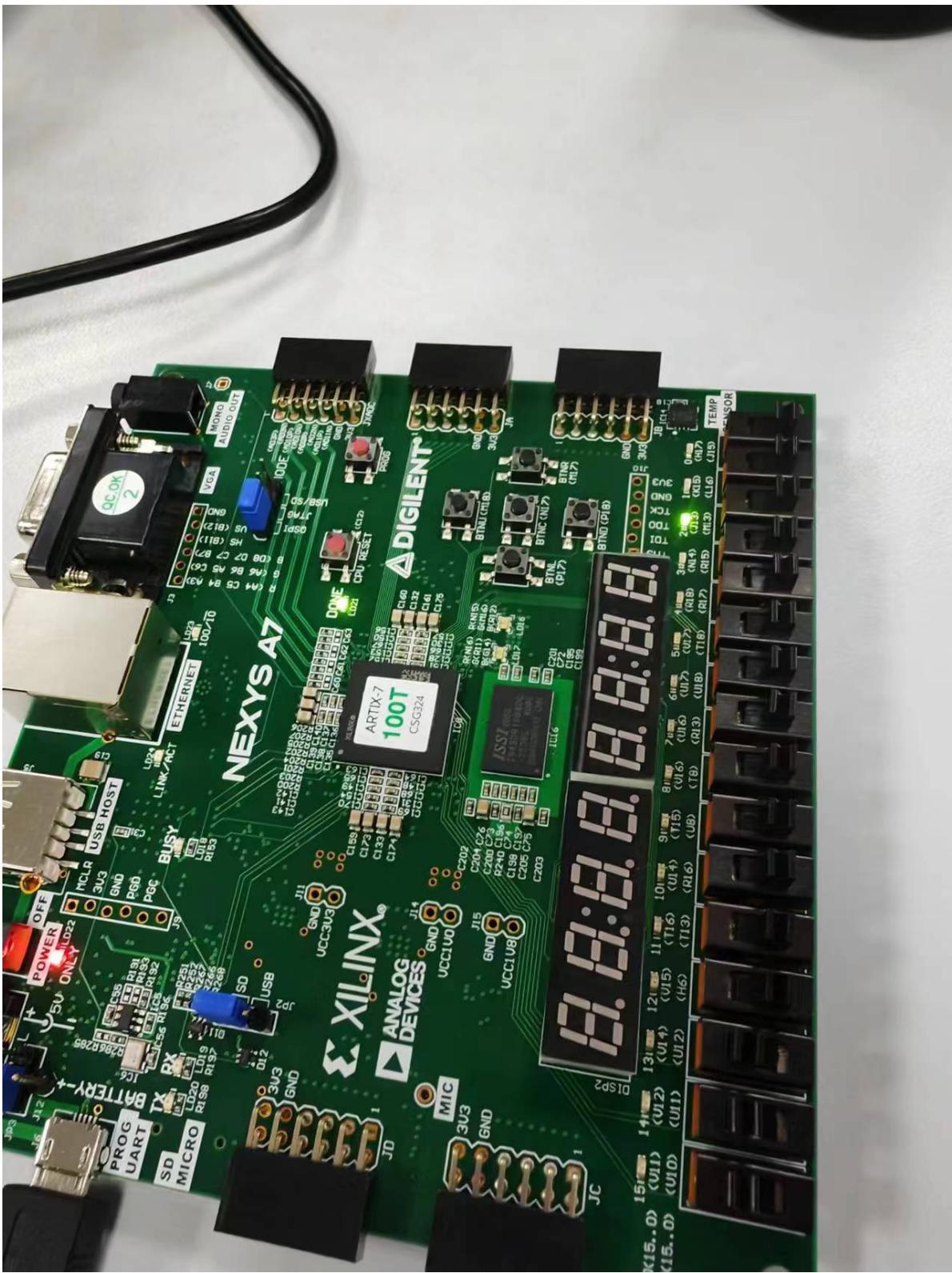
生成比特流

引脚约束后重新声称比特流，结果如下：



下板验证

连接设备后获得下板结果：



任务二：自定义模块化设计学习

利用VIVADO完成多路选择器MUX2T1_5的设计和封装

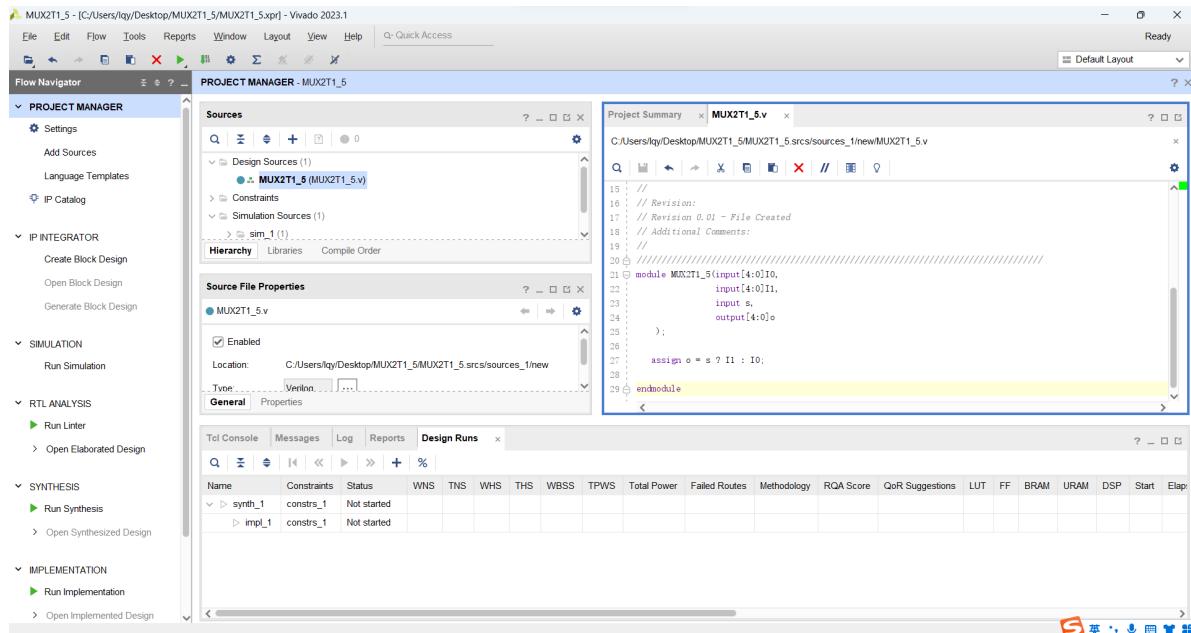
创建源文件

模块代码如下：

```

1 module MUX2T1_5(input[4:0] I0,
2                  input[4:0] I1,
3                  input s,
4                  output[4:0] o
5 );
6
7 assign o = s ? I1 : I0;
8
9 endmodule

```



仿真文件与结果

仿真激励代码：

```

1 module MUX2T1_5_tb();
2   reg [4:0] I0;
3   reg [4:0] I1;
4   reg s;
5
6   wire [4:0]o;
7   initial begin
8     s = 0;
9     I0 = 0;
10    I1 = 1;
11    #50;
12    s = 0;
13    #50;
14    s = 1;
15    #50;
16    I0 = 4'h5;
17    I1 = 4'hA;
18    #50;
19    s = 0;
20    #50;
21    s = 1;
22    #50;
23    s = 0;

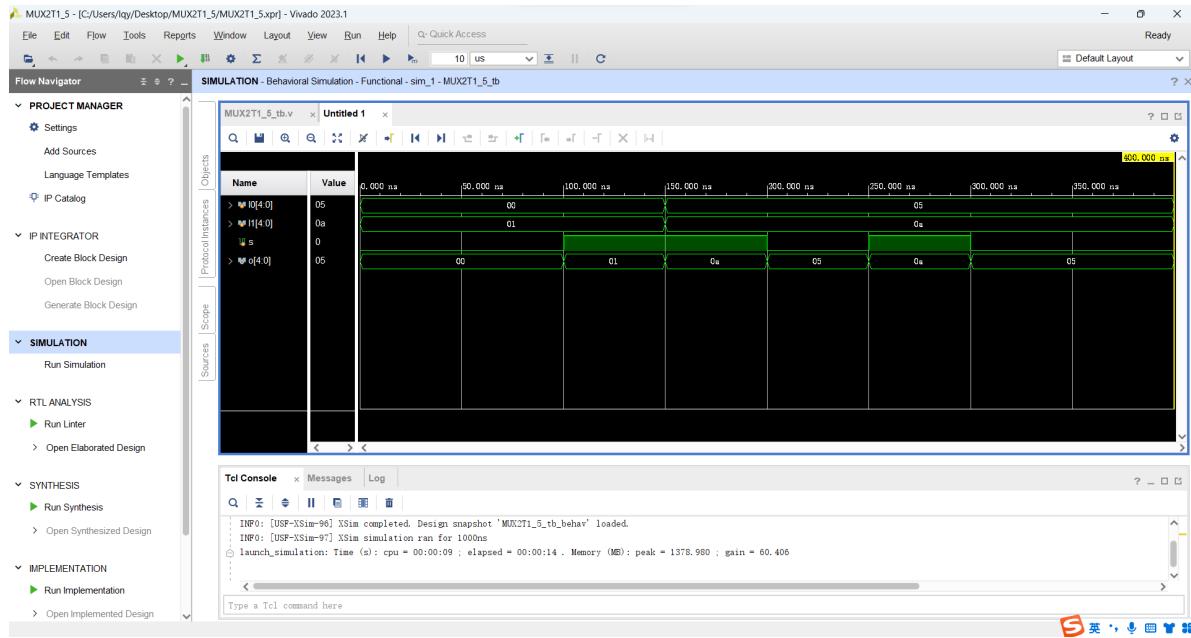
```

```

24      #100 $stop;
25
26 end
27
28 MUX2T1_5 MUX(
29     .I0(I0),
30     .I1(I1),
31     .S(S),
32     .O(O)
33 );
34
35 endmodule

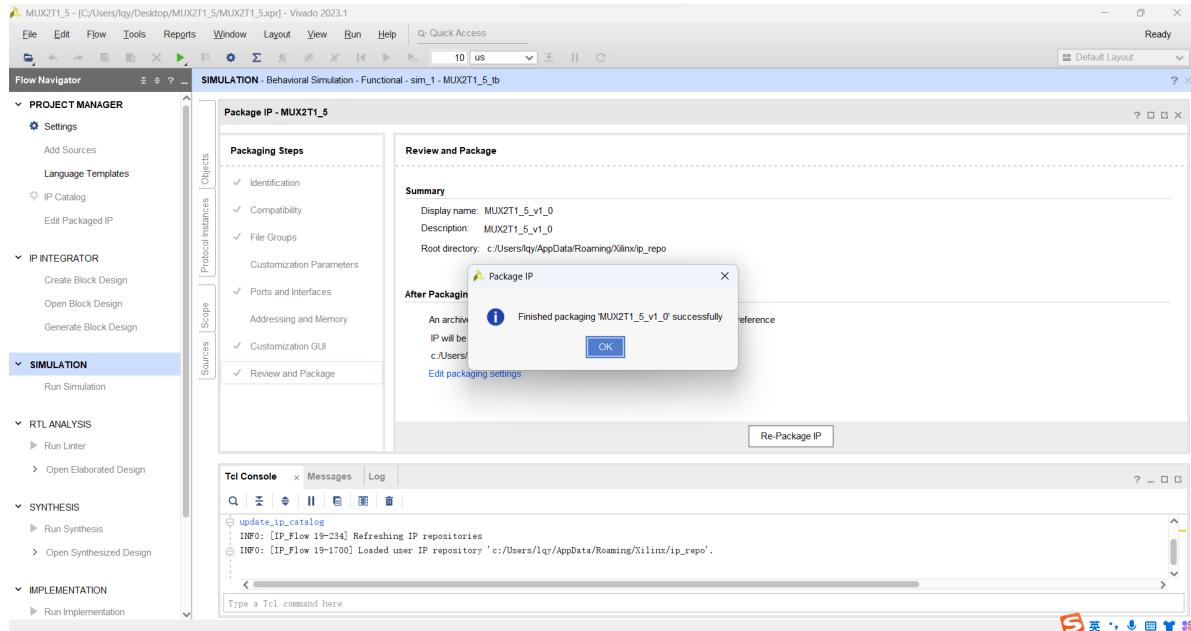
```

仿真结果：



模块封装

根据一系列封装操作进行封装，结果如下：



就完成了包含源文件的模块封装

任务三：基本逻辑模块的原理学习、Xilinx IP 的生成

任务1：基本模块功能

MUX2T1_5

见[MUX2T1_5](#)

MUX2T1_8

```
1 module MUX2T1_8(input[7:0]I0,
2                   input[7:0]I1,
3                   input s,
4                   output[7:0]o
5 );
6   assign o = s ? I1 : I0;
7 endmodule
```

MUX2T1_32

```
1 module MUX2T1_32(input[31:0]I0,
2                   input[31:0]I1,
3                   input s,
4                   output[31:0]o
5 );
6   assign o = s ? I1 : I0;
7 endmodule
```

MUX4T1_5

见[MUX4T1_5](#)

MUX4T1_32

```
1 module MUX4T1_32(      input [1:0]s,
2                     input [31:0]I0,
3                     input [31:0]I1,
4                     input [31:0]I2,
5                     input [31:0]I3,
6                     output reg[31:0]o
7 );
8   always@*
9     case(s)
10       2'b00: o = I0;
11       2'b01: o = I1;
12       2'b10: o = I2;
13       2'b11: o = I3;
14     endcase
15   endmodule
```

MUX8T1_8

见[MUX8T1_8](#)

add_32

```
1 module add_32(  input [31:0] a,
2                  input [31:0] b,
3                  output [31:0]c
4 );
5     assign c = a + b;
6 endmodule
```

addc_32

```
1 module addc_32(  input [31:0] A,
2                  input [31:0] B,
3                  input C0,
4                  output [32:0] s
5 );
6     wire B_Notation = C0 ^ 1'b0;
7     assign s = {1'b0,A} + {B_Notation,B} + C0;
8 endmodule
```

and_32

```
1 module and32(  input [31:0] A,
2                  input [31:0] B,
3                  output [31:0] res
4 );
5     assign res = A & B;
6 endmodule
```

Ext_imm16

```
1 module Ext_imm16( input [15:0] imm_16,
2                     output[31:0] Imm_32
3
4 );
5     assign Imm_32 = {{16{imm_16[15]}},imm_16[15:0]};
6 endmodule
```

nor_32

```
1 module nor32(
2     input [31:0] A;
3     input [31:0] B;
4     output [31:0] res
5 );
6     assign res = ~(A | B);
7 endmodule
```

or_bit_32

```
1 module or_bit_32(      input [31:0] A,
2                         output o
3 );
4     assign o = (A==0)? 1: 0;
5 endmodule
```

or_32

```
1 module or_32(    input [31:0] A,
2                  input [31:0] B,
3                  output [31:0] res
4 );
5     assign res = A | B;
6 endmodule
```

SignalExt_32

```
1 module SignalExt_32(   input s,
2                         output [31:0] so
3 );
4     assign so = {32{s}};
5 endmodule
```

srl_32

```
1 module srl_32(    input [31:0] A,
2                   input [31:0] B,
3                   output [31:0] res
4 );
5     assign res = A >> B[4:0];
6 endmodule
```

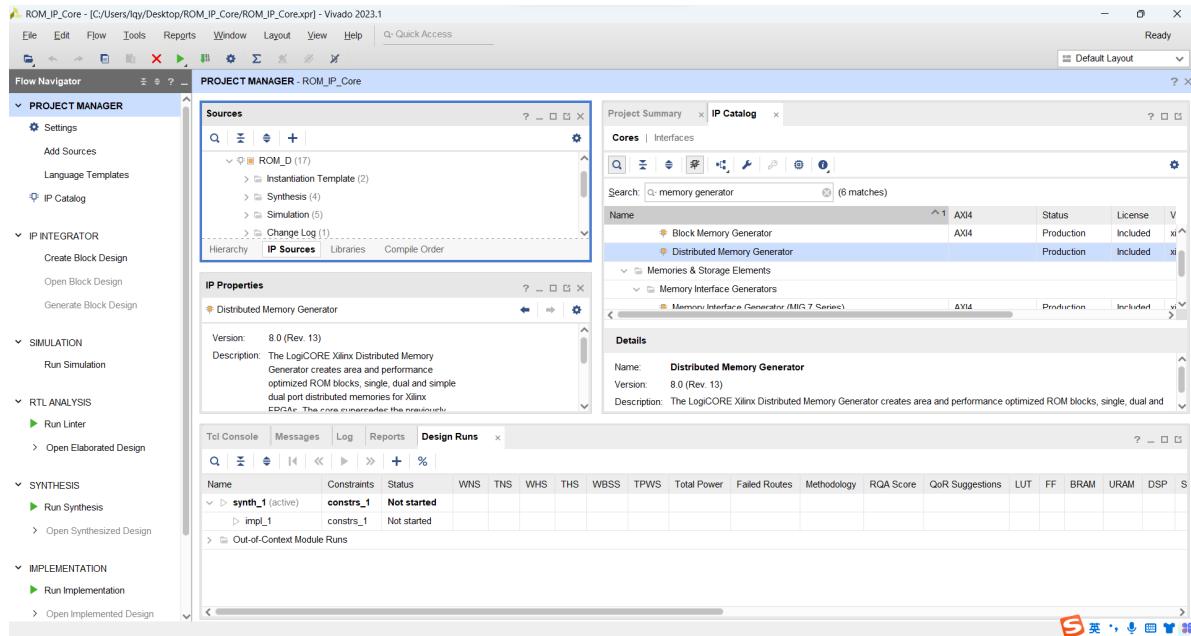
xor_32

```
1 module xor_32(
2     input [31:0] A;
3     input [31:0] B;
4     output [31:0] res
5 );
6     assign res = A ^ B;
7 endmodule
```

任务2：存储器ROM、RAM的生成及初始化

ROM_D IP Core

根据相关步骤创建ROM的IP核

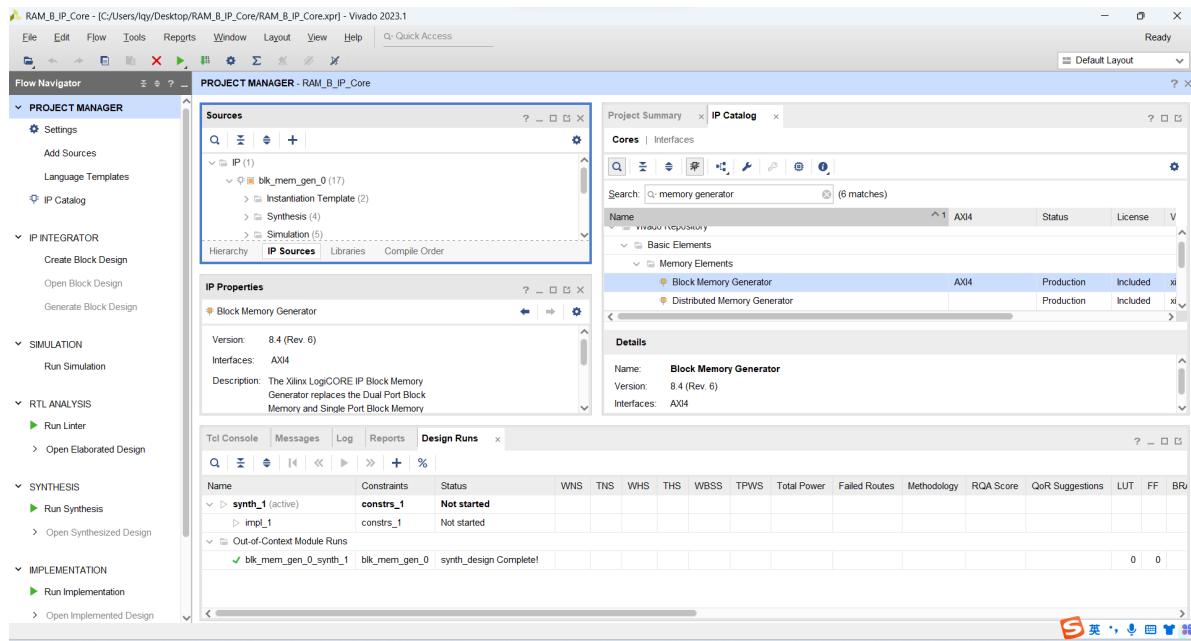


初始化的 ROM.coe 代码如下：

```
1 memory_initialization_radix=16;
2 memory_initialization_vector=
3 00000000, 11111111, 22222222, 33333333, 44444444, 55555555, 66666666,
4 77777777, 88888888, 99999999, aaaaaaaaaa, bbbbbbbb,
5 cccccccc, dddddddd, eeeeeeee, ffffffff, 557EF7E0, D7BDFBD9, D7DBFDB9,
6 DFCFFCFB, DFCFBFFF, F7F3DFFF, FFFFDF3D, FFFF9DB9, FFFFBCFB, DFCFFCFB,
7 DFCFBFFF, D7DB9FFF, D7DBFDB9, D7BDFBD9, FFFF07E0, 007E0FFF, 03bdf020,
8 03def820, 08002300;
```

RAM_B IP Core

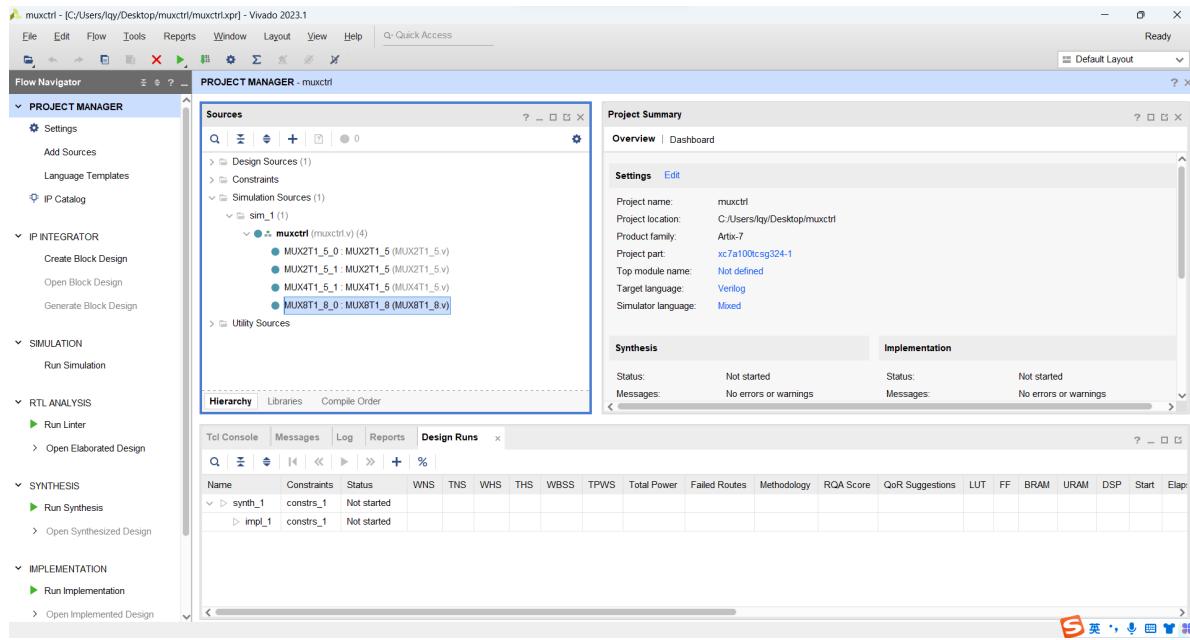
根据相关步骤创建RAM的IP核，结果如下：



任务四：利用自定义模块构建实验平台

采用Verilog编码的设计方法，调用自定义模块，完成多选器控制LED的设计并利用NEXYS_A7实验板进行硬件验证

模块代码



muxctrl

```
1 module muxctrl(
2     input wire [4:0]I0,
3     input wire [4:0]I1,
4     input wire [1:0]S,
5     input wire S1,
6     input wire [2:0]S2,
7     output wire [4:0]O_0
8 );
9     wire [4:0] MUX2T1_5_O;
10    wire [4:0] MUX2T1_5_1_O;
11    wire [7:0] MUX8T1_8_O;
12    MUX2T1_5 MUX2T1_5_0
13        (.I0(I0),
14         .I1(I1),
15         .O(MUX2T1_5_O),
16         .S(S1));
17    MUX2T1_5 MUX2T1_5_1
18        (.I0(I0),
19         .I1(I1),
20         .O(MUX2T1_5_1_O),
21         .S(1'b1));
22    MUX4T1_5 MUX4T1_5_1
23        (.I0(MUX8T1_8_O[3:0]),
24         .I1({MUX2T1_5_1_O[0], MUX2T1_5_0_O[3:0]}),
25         .I2(5'b1),
26         .I3(5'b0),
27         .O(O_0),
28         .S(S));
```

```

29      MUX8T1_8 MUX8T1_8_0
30      (.I0({MUX2T1_5_0_o[3:0],MUX2T1_5_1_o[3:0]}),
31      .I1({MUX2T1_5_1_o[3:0],MUX2T1_5_0_o[3:0]}),
32      .I2(8'b1),
33      .I3(8'b1),
34      .I4(8'b1),
35      .I5(8'b1),
36      .I6(8'b1),
37      .I7(8'b1),
38      .o(MUX8T1_8_0_o),
39      .s(s2));
40 endmodule

```

MUX2T1_5

```

1 module MUX2T1_5(
2     input[4:0]I0,
3     input[4:0]I1,
4     input s,
5     output[4:0]o
6 );
7     assign o = s ? I1 : I0;
8 endmodule

```

MUX4T1_5

```

1 module MUX4T1_5(
2     input [1:0]s,
3     input [4:0]I0,
4     input [4:0]I1,
5     input [4:0]I2,
6     input [4:0]I3,
7     output reg[4:0]o
8 );
9     always@*
10         case(s)
11             2'b00: o = I0;
12             2'b01: o = I1;
13             2'b10: o = I2;
14             2'b11: o = I3;
15         endcase
16 endmodule

```

MUX8T1_8

```

1 module MUX8T1_8(
2     input [2:0]s,
3     input [7:0]I0,
4     input [7:0]I1,
5     input [7:0]I2,
6     input [7:0]I3,
7     input [7:0]I4,
8     input [7:0]I5,

```

```

9   input [7:0]I6,
10  input [7:0]I7,
11  output reg[7:0]o
12 );
13 always@*
14   case(s)
15     3'b000: o = I0;
16     3'b001: o = I1;
17     3'b010: o = I2;
18     3'b011: o = I3;
19     3'b100: o = I4;
20     3'b101: o = I5;
21     3'b110: o = I6;
22     3'b111: o = I7;
23   endcase
24 endmodule

```

引脚约束

```

1 #switch
2 set_property IOSTANDARD LVCMOS33 [get_ports {s[0]}]
3 set_property PACKAGE_PIN J15      [get_ports {s[0]}]
4 set_property IOSTANDARD LVCMOS33 [get_ports {s[1]}]
5 set_property PACKAGE_PIN L16      [get_ports {s[1]}]
6 set_property IOSTANDARD LVCMOS33 [get_ports s1]
7 set_property PACKAGE_PIN M13      [get_ports s1]
8 set_property IOSTANDARD LVCMOS33 [get_ports {s2[0]}]
9 set_property PACKAGE_PIN R15      [get_ports {s2[0]}]
10 set_property IOSTANDARD LVCMOS33 [get_ports {s2[1]}]
11 set_property PACKAGE_PIN R17      [get_ports {s2[1]}]
12 set_property IOSTANDARD LVCMOS33 [get_ports {s2[2]}]
13 set_property PACKAGE_PIN T18      [get_ports {s2[2]}]
14 set_property IOSTANDARD LVCMOS33 [get_ports {I0[0]}]
15 set_property PACKAGE_PIN U18      [get_ports {I0[0]}]
16 set_property IOSTANDARD LVCMOS33 [get_ports {I0[1]}]
17 set_property PACKAGE_PIN R13      [get_ports {I0[1]}]
18 set_property IOSTANDARD LVCMOS18 [get_ports {I0[2]}]
19 set_property PACKAGE_PIN T8       [get_ports {I0[2]}]
20 set_property IOSTANDARD LVCMOS18 [get_ports {I0[3]}]
21 set_property PACKAGE_PIN U8       [get_ports {I0[3]}]
22 set_property IOSTANDARD LVCMOS33 [get_ports {I0[4]}]
23 set_property PACKAGE_PIN R16      [get_ports {I0[4]}]
24 set_property IOSTANDARD LVCMOS33 [get_ports {I1[0]}]
25 set_property PACKAGE_PIN T13      [get_ports {I1[0]}]
26 set_property IOSTANDARD LVCMOS33 [get_ports {I1[1]}]
27 set_property PACKAGE_PIN H6       [get_ports {I1[1]}]
28 set_property IOSTANDARD LVCMOS33 [get_ports {I1[2]}]
29 set_property PACKAGE_PIN U12      [get_ports {I1[2]}]
30 set_property IOSTANDARD LVCMOS33 [get_ports {I1[3]}]
31 set_property PACKAGE_PIN U11      [get_ports {I1[3]}]
32 set_property IOSTANDARD LVCMOS33 [get_ports {I1[4]}]
33 set_property PACKAGE_PIN V10      [get_ports {I1[4]}]
34
35 set_property IOSTANDARD LVCMOS33 [get_ports {o_0[0]}]
36 set_property PACKAGE_PIN H17      [get_ports {o_0[0]}]

```

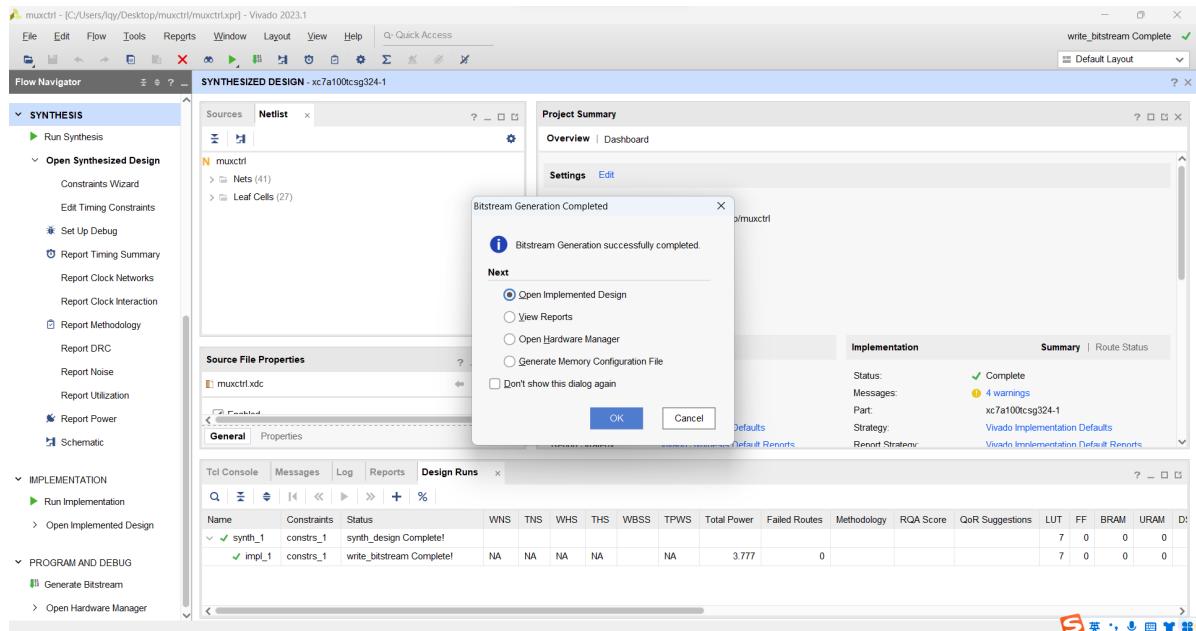
```

37 set_property IOSTANDARD LVCMOS33 [get_ports {o_0[1]}]
38 set_property PACKAGE_PIN K15      [get_ports {o_0[1]}]
39 set_property IOSTANDARD LVCMOS33 [get_ports {o_0[2]}]
40 set_property PACKAGE_PIN J13      [get_ports {o_0[2]}]
41 set_property IOSTANDARD LVCMOS33 [get_ports {o_0[3]}]
42 set_property PACKAGE_PIN N14      [get_ports {o_0[3]}]
43 set_property IOSTANDARD LVCMOS33 [get_ports {o_0[4]}]
44 set_property PACKAGE_PIN R18      [get_ports {o_0[4]}]

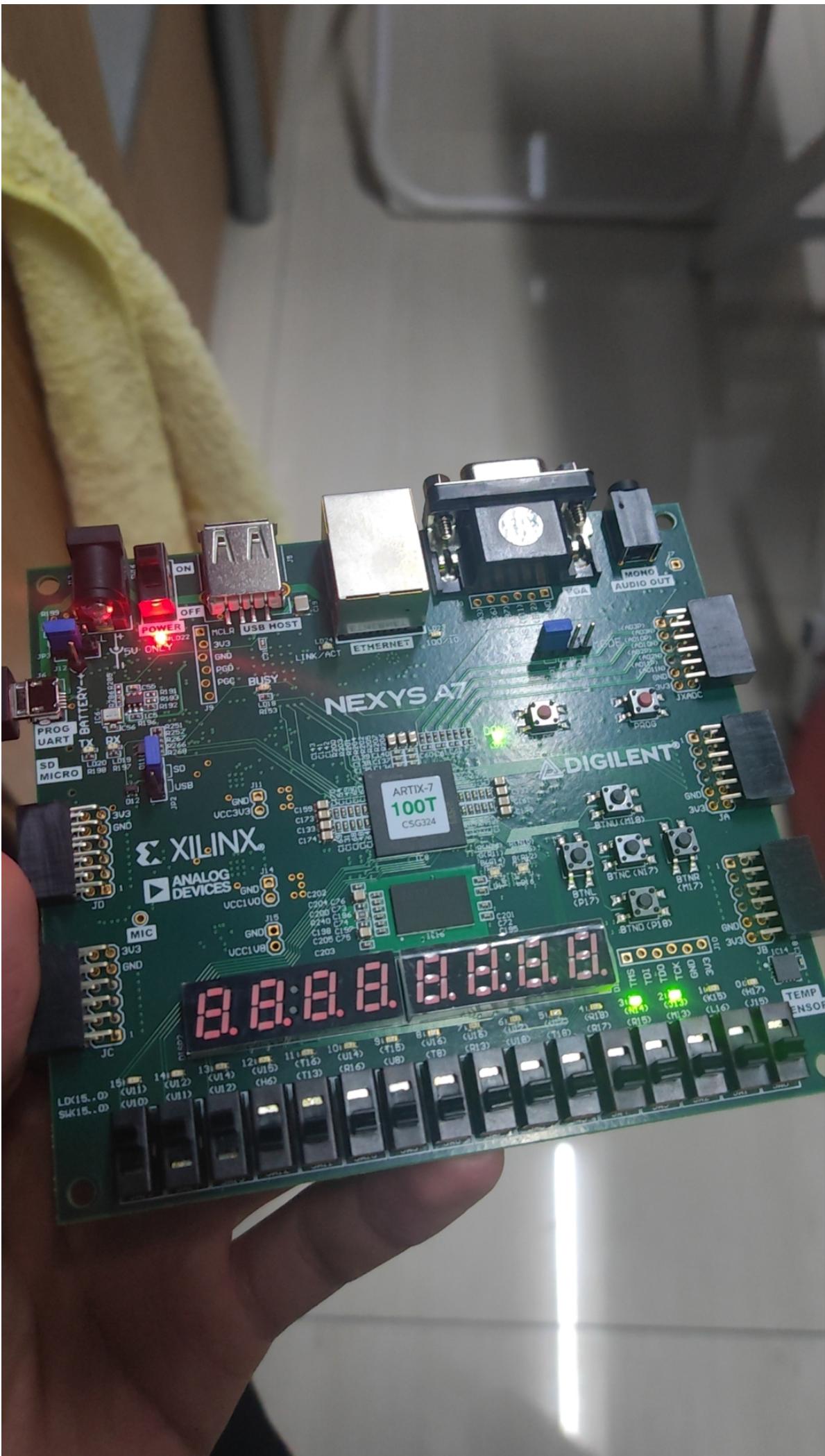
```

下板验证

生成比特流：



下板结果：



思考题

说明约束文件代码中的参数的意义作用，说明ROM/RAM的参数意义和应用场景

- 约束文件

以管脚约束脚本为例：

大体上代码分为两种：

```
1 # 1
2 set_property PACKAGE_PIN C12 [get_ports RSTn_i]
3 set_property PACKAGE_PIN H17 [get_ports {LED_o[0]}]
4 set_property PACKAGE_PIN K15 [get_ports {LED_o[1]}]
5 set_property PACKAGE_PIN J13 [get_ports {LED_o[2]}]
6 set_property PACKAGE_PIN N14 [get_ports {LED_o[3]}]
7 set_property PACKAGE_PIN E3 [get_ports CLK_i]
8
9 # 2
10 set_property IOSTANDARD LVCMOS33 [get_ports RSTn_i]
11 set_property IOSTANDARD LVCMOS33 [get_ports {LED_o[0]}]
12 set_property IOSTANDARD LVCMOS33 [get_ports {LED_o[1]}]
13 set_property IOSTANDARD LVCMOS33 [get_ports {LED_o[2]}]
14 set_property IOSTANDARD LVCMOS33 [get_ports {LED_o[3]}]
15 set_property IOSTANDARD LVCMOS33 [get_ports CLK_i]
```

第一部分规定了模块代码中的参数对应的使用引脚，如 `RSTn_i` 对应引脚 `C12`，`LED_o[0]` 对应引脚 `H17`

第二部分则是使用的引脚的电平标准为 `LVCMOS33`

- ROM

```
1 ROM_B your_instance_name (
2   .a(a),           // input [9 : 0] a
3   .spo(spo)        // output [31 : 0] spo
4 );
```

`[9:0] a` 表示存储容量为1024个单元，`[31:0] spo` 代表每个存储单元的字长为32bit

ROM通常用于存储固件、程序代码等其他通常不需要改变的数据

- RAM

```
1 RAM_U3 (
2   .c1ka(c1k_m),      // input c1ka
3   .wea(data_ram_we), // input [0 : 0] wea
4   .addr(ram_addr),   // input [9 : 0] addr
5   .dina(ram_data_in), // input [31:0] dina
6   .douta(ram_data_out)// output [31:0] douta
7 );
```

`c1ka` 代表输入时钟周期, `[9:0] addra` 代表输入的位宽为32bit, `[31:0] dina` 代表写入的存储单元地址, `[31:0] douta` 代表读出的存储单元地址

二

总结一下自己数逻学习了什么内容, 包括理论和实验部分

总体来说, 数逻学习了数字逻辑电路的组成和设计

- 理论

理论部分从最开始的布尔代数开始, 到逻辑门, 简单的逻辑运算, 之后是组合逻辑电路的实现, 然后是时序逻辑电路的实现, 后续还学了状态模型、电路设计, 以及最后部分的寄存器等的实现, 总体来说是基本的计算机组成的底层理论原理

- 实验

实验部分从最开始的电路的基础知识到后面使用Verilog参与对逻辑门、选择器、寄存器、计数器等的实现, 以及最后的大程, 学会使用Verilog编写相关的电路代码, 学会使用VIVADO工具, 学会电路分析与设计

三

查查资料, 随便讨论讨论自己对于硬件发展前沿的理解

查阅相关资料后了解到, 下一步硬件发展将可能是革命性的, 由于经过几代发展的计算机硬件基本建立在半导体材料之上, 当下硬件的发展已经逼近物理极限, 各种技术都到达一个瓶颈期, 因此想要有硬件更大程度的发展, 可能是突破半导体材料的一次材料革命后, 出现新的技术革命, 来带动硬件新一轮的发展

四

说说自己对于这门课的期望 (理论、实验、老师、助教都可以, 随便谈谈)

非常期望能够在计算机组成这一门课学到很多新的有用的知识。本人专业为计算机科学与技术, 对硬件和软件都有相当浓厚的兴趣。硬件的学习可能某种程度上来说会略显枯燥, 但是如果真正掌握其中硬件的组成原理对于我个人来说非常具有成就感, 即完全读透的感觉是非常的爽快。对硬件能够侃侃而谈是我的目标, 同时这也会加深我对计算机科学这一学科的深层次理解。希望通过这门课能够学到真正有用的东西。

同时, 希望自己能够不受考试过多的约束而更加灵活、自由地学习这门课。上学期数字逻辑设计开始之时我还没有这样的觉悟, 以至于到课程中后期变成疲于应付实验课和理论课的双重压力, 而没有学好学透彻, 最后沦为考试的阶下囚。这学期希望能够更加熟练掌握相关知识, 提前做好学习准备

希望老师助教能够更加耐心解答, 尤其是有些步骤如果跳步, 可能对像我一样的计算机小白非常unfriendly, 需要自己课后花更多时间去重新学习 😊