浙江大學

计算机逻辑设计基础

Lab 4

Author: 苏煜程

Student ID: 3220105481

Date: 2023 年 10 月 23 日

浙江大学实验报告

课程名称：计算机逻辑设计基础

实验名称：EDA 实验平台与实验环境运用

学生姓名：苏煜程　　专业：人工智能（图灵班）　　学号：3220105481

同组学生姓名：张延泽　　指导老师：董亚波

实验地点：东 4-509　　实验日期：2023 年 10 月 12 日

1 实验目的

1. 熟悉 Verilog HDL 语言并能用其建立基本的逻辑部件，在 Xilinx ISE 平台进行输入、编辑、调试、行为与仿真与综合后功能仿真
2. 熟悉掌握 SWORD FPGA 开发平台，同时在 ISE 平台上进行时序约束、引脚约束及映射布线后时序仿真
3. 运用 Xilinx ISE 工具将设计验证后的代码下载到实验板上，并在实验板上验证

2 实验任务

1. 熟悉 ISE 工具软件的运行环境与安装过程
2. 设计简单组合逻辑电路，采用图形输入逻辑功能描述，建立 FPGA 实现数字系统的 Xilinx ISE 设计管理工程，并进行编辑、调试、编译、行为仿真，时序约束、引脚指定（约束）、映射布线后时序仿真及 FPGA 编程代码下载与运行验证
3. 设计简单时序逻辑电路，采用 Verilog 代码输入逻辑功能描述，建立 FPGA 实现数字系统的 ISE 设计管理工程，并进行编辑、调试、编译、行为仿真，时序约束、引脚约束、映射布线后时序仿真及 FPGA 编程代码下载与运行验证

3 实验原理

问题 1　某三层楼房的楼梯通道共用一盏灯，每层楼都安装了一只开关并能独立控制该灯，请设计楼道灯的控制电路。

分析楼道灯的事件行为，用组合电路实现，用拨动开关作为电路输入 S_1, S_2, S_3 ，电路输出为 F.

变量赋值:

- 开关往下为 1，往上为 0。
- 输出灯亮为 1，灯暗为 0。

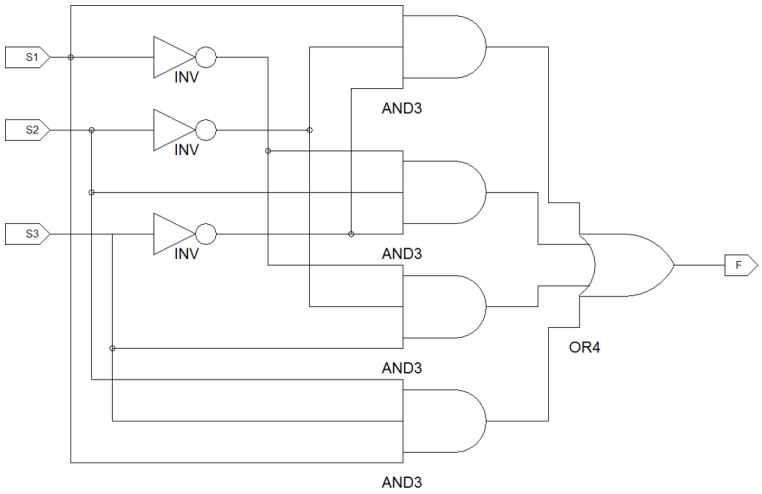
编写真值表如下：

S_3	S_2	S_1	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

根据真值表分析输入输出关系：

$$F = S_1\bar{S}_2\bar{S}_3 + \bar{S}_1S_2\bar{S}_3 + \bar{S}_1\bar{S}_2S_3 + S_1S_2S_3$$

得到电路图如下：



问题 2 增加控制要求，灯打开后，延时若干秒自动关闭，请重新设计楼道灯的控制电路。

分析楼道灯的事件行为，用时序电路实现，用按钮开关作为电路输入 S_1, S_2, S_3 ，电路输出为 F。

变量赋值:

- 开关往下为 1，往上为 0。
- 输出灯亮为 1，灯暗为 0。

编写 Verilog 代码实现功能。

4 实验内容与测试步骤

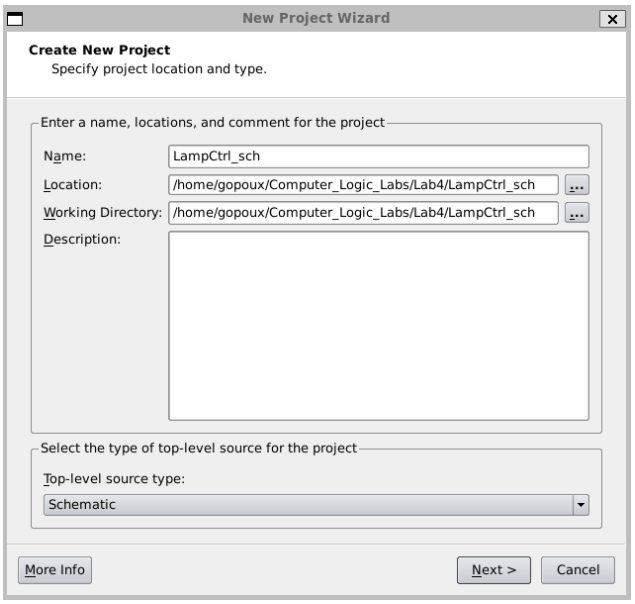
4.1 图形方式输入逻辑功能描述

4.1.1 建立楼道控制的工程

1. 依次点击菜单 File → New Project.

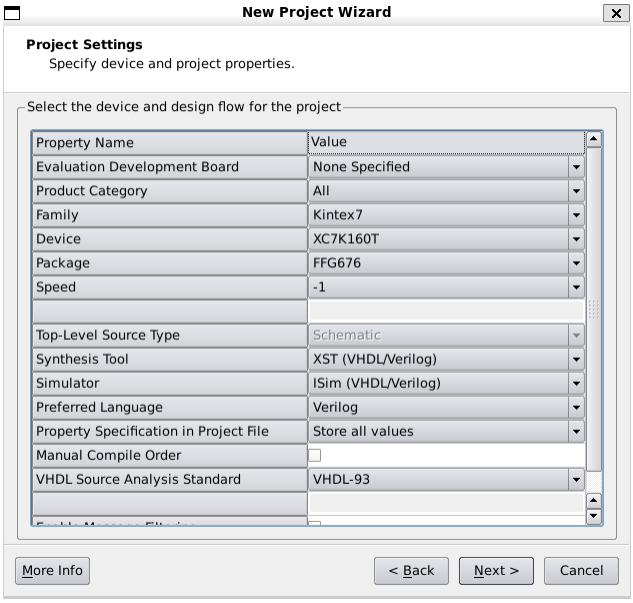
2. 在对话框中设置：

   - Project Name: `LampCtrl_sch.ise`
   - Top-Level Source Type: Schematic



3. 确认后，点击 Next 到设备属性页，设置：

   - Family: Kintex7
   - Device: XC7K160T
   - Package: FFG676
   - Speed: -1

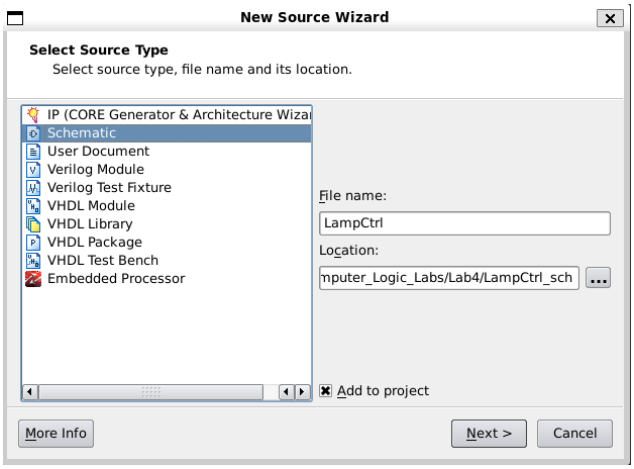


4. 确认后，一直点击 Next 直到创建工程结束。

4.1.2 创建原理图文件

1. 在 Sources 窗口 Sources 选项卡设备型号名处右键菜单选择 New Source.

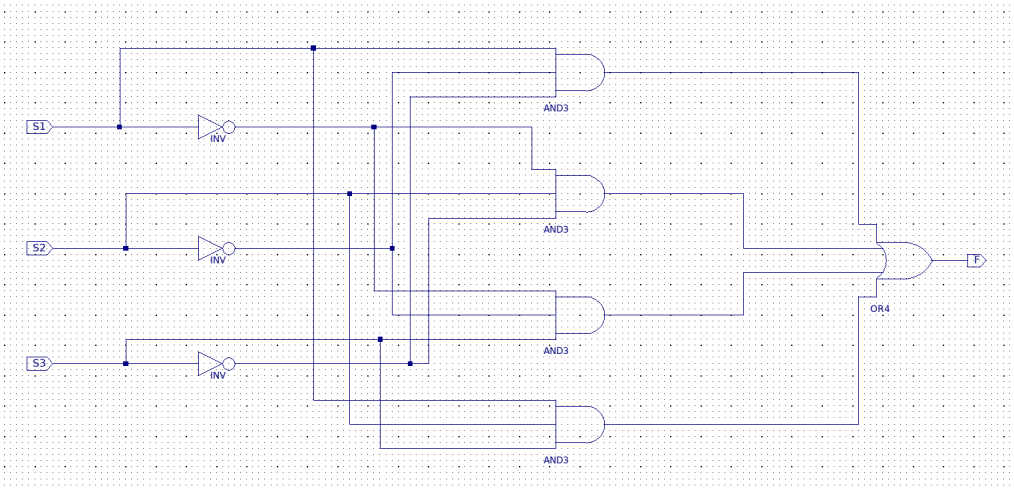
2. 新建源文件向导中选择源文件类型为 Schematic，输入文件名 `LampCtrl`，勾选 Add to Project



3. 连续点击 Next，最后点击 Finish ；在 Sources 窗口中双击刚新建的文件图标，进入电路原理图编辑窗口。

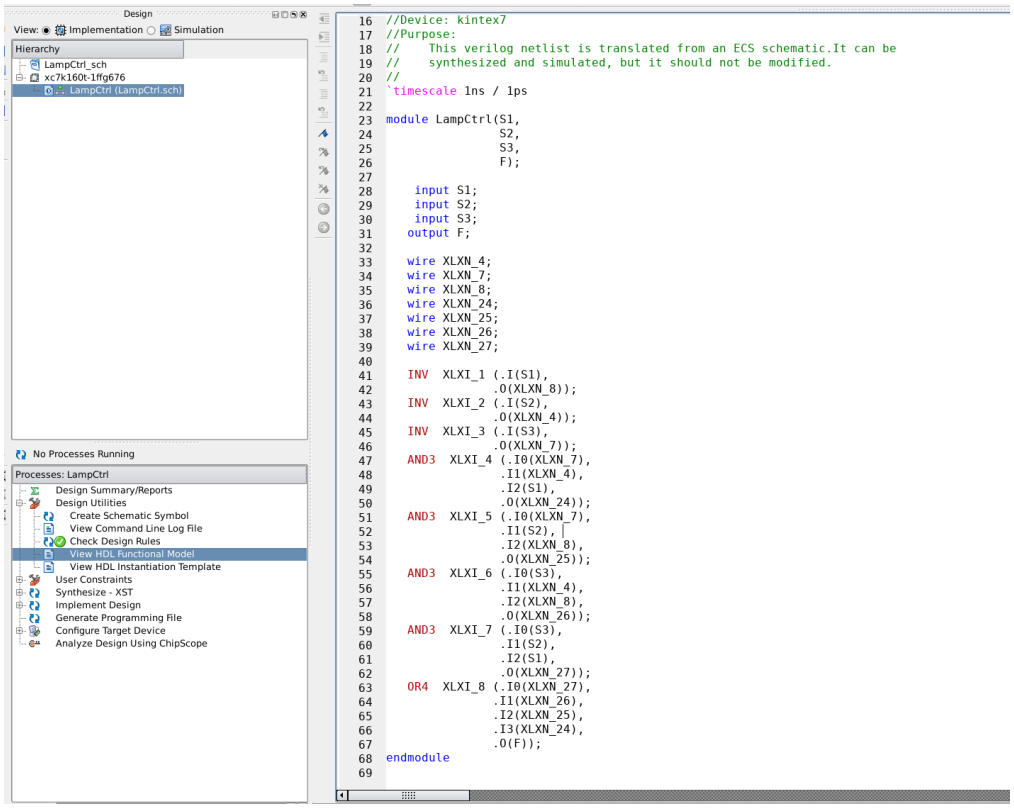
4.1.3 输入楼道灯控逻辑电路

在 Souces 窗口中选择 Symbols 选项卡，配合 Schematic Editor 工具条输入原理图，如图



4.1.4 查看输入电路的硬件描述代码

在 Sources 窗口中选择 Sources for: Synthesis / Implementation，选中 `LampCtrl.sch` 图标，在 Processes 窗口 Processes 选项卡中展开 Design Utilities 并双击 View HDL Functional Model ，如图

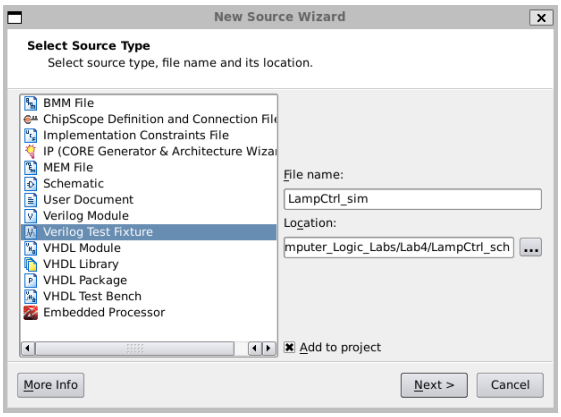


4.1.5 建立基准测试波形文件

1. 在 Sources 窗口空白处的右键菜单中选择 New Source

2. 在新建源文件向导中选择源类型为：Verilog Test Fixture，输入文件名 `LampCtrl_sim`，并勾选 Add to Project

3. 选择 LampCtrl 模块，点击 Next，在 Summary 窗口再点击 Finish，进入 `LampCtrl_sim.v` 编辑窗口

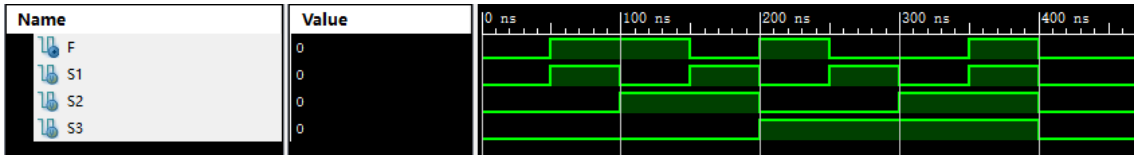


4.1.6 仿真激励输入

```
`timescale 1ns / 1ps
module LampCtrl_LampCtrl_sch_tb();

// Inputs
   reg S1;
   reg S2;
   reg S3;

// Output
   wire F;

// Bidirs

// Instantiate the UUT
   LampCtrl UUT (
        .S1(S1),
        .S2(S2),
        .S3(S3),
        .F(F)
   );
// Initialize Inputs
```

```
//    `ifdef auto_init
        initial begin
          S1 = 0;
          S2 = 0;
          S3 = 0;
      #50 S1 = 1;
      #50 S1 = 0;
          S2 = 1;
      #50 S1 = 1;
      #50 S1 = 0;
          S2 = 0;
          S3 = 1;
      #50 S1 = 1;
      #50 S1 = 0;
          S2 = 1;
      #50 S1 = 1;
      #50 S1 = 0;
          S2 = 0;
          S3 = 0;
      end
//    `endif
endmodule
```

View 选择 Simulation 视图，Hierarchy 窗口中选择 `LampCtrl_LampCtrl_sch_tb` ，Process 窗口中选择 Simulate Behavioral Model 。生成波形如图

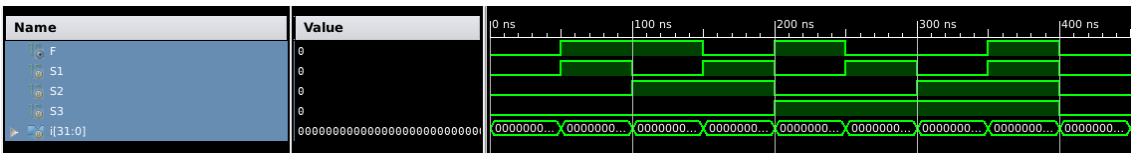


4.1.7 另一种仿真激励输入

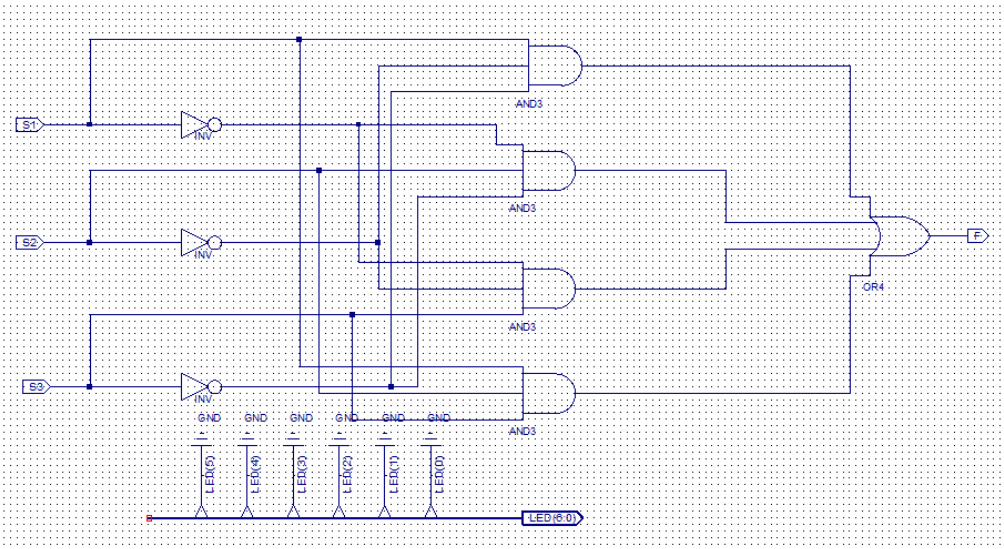
```
`timescale 1ns / 1ps

module LampCtrl_LampCtrl_sch_tb();

// Inputs
   reg S1;
```

```
    reg S2;
    reg S3;

// Output
    wire F;

// Bidirs

// Instantiate the UUT
    LampCtrl UUT (
          .S1(S1),
          .S2(S2),
          .S3(S3),
          .F(F)
    );
// Initialize Inputs
//    `ifdef auto_init
       integer i;
         initial begin
             for(i=0;i<=8;i=i+1) begin
                 {S3,S2,S1} <= i;
                 #50;
             end
         end
//    `endif
endmodule
```

得到波形如下：



4.1.8 最终电路



4.1.9 建立用户时序约束并为模块的端口指定引脚分配

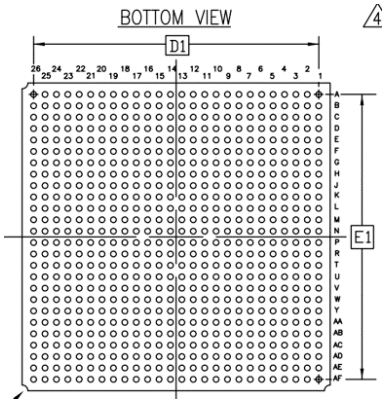
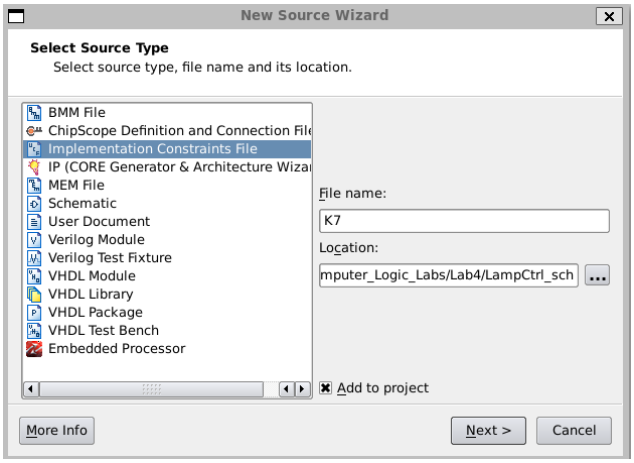


图 2: XC7K160T 芯片的引脚图

1. 在 Sources 窗口空白处的右键菜单中选择 New Source
2. 在新建源文件向导中选择源类型为：Implementation Constraints File，输入文件名 K7，并勾选 Add to Project
3. 点击 Finish 进入 `K7.ucf` 编辑窗口，输入代码

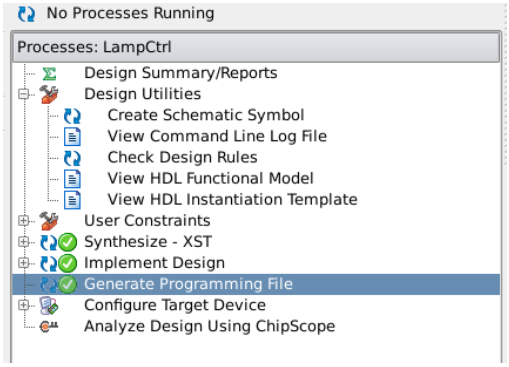


在 K7.ucf 中输入以下代码：

```
NET "S1" LOC = AA10 | IOSTANDARD = LVCMOS15;
NET "S2" LOC = AB10 | IOSTANDARD = LVCMOS15;
NET "S3" LOC = AA13 | IOSTANDARD = LVCMOS15;
NET "F" LOC = AF24 | IOSTANDARD = LVCMOS33;
NET "LED[0]" LOC = W23 | IOSTANDARD = LVCMOS33;
NET "LED[1]" LOC = AB26 | IOSTANDARD = LVCMOS33;
NET "LED[2]" LOC = Y25 | IOSTANDARD = LVCMOS33;
NET "LED[3]" LOC = AA23 | IOSTANDARD = LVCMOS33;
NET "LED[4]" LOC = Y23 | IOSTANDARD = LVCMOS33;
NET "LED[5]" LOC = Y22 | IOSTANDARD = LVCMOS33;
NET "LED[6]" LOC = AE21 | IOSTANDARD = LVCMOS33;
```

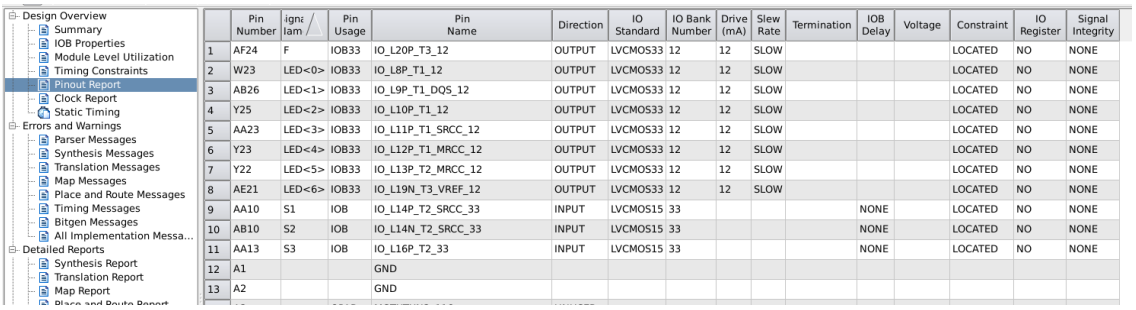
4.1.10 设计实现

在 Degign 窗口中选择 Implementation，选中 LampCtrl 顶层模块；在 Processes 窗口下右键点击 Generate Programming File，选择 Run，进行物理转换、平面布图、映射、物理布线等 FPGA 综合操作，实现目标文件生成。



4.1.11 检查约束结果

最后在 Design Summary 设计摘要文档中有如下结果：



Design Overview
- Summary
- IOB Properties
- Module Level Utilization
- Timing Constraints
- Pinout Report
- Clock Report
- Static Timing

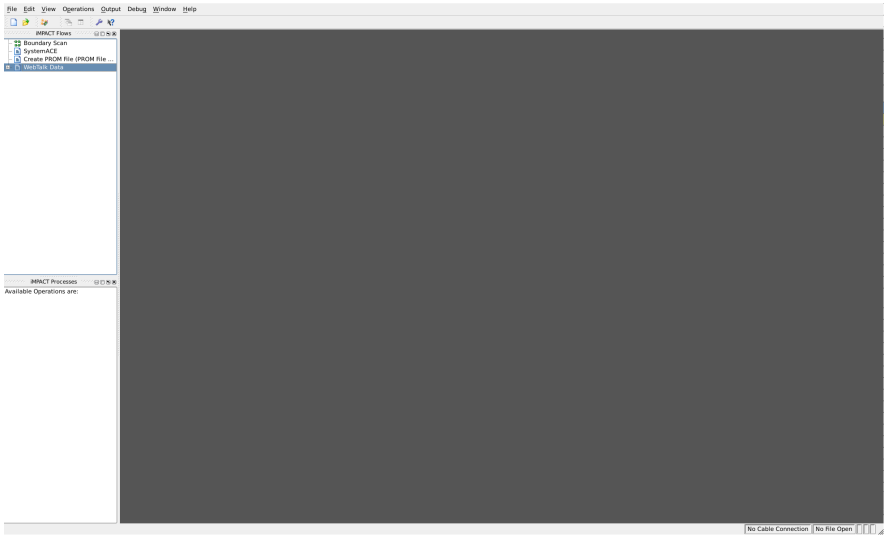
Errors and Warnings
- Parser Messages
- Synthesis Messages
- Translation Messages
- Map Messages
- Place and Route Messages
- Timing Messages
- Bitgen Messages
- All Implementation Messa...

Detailed Reports
- Synthesis Report
- Translation Report
- Map Report
- Place and Route Report

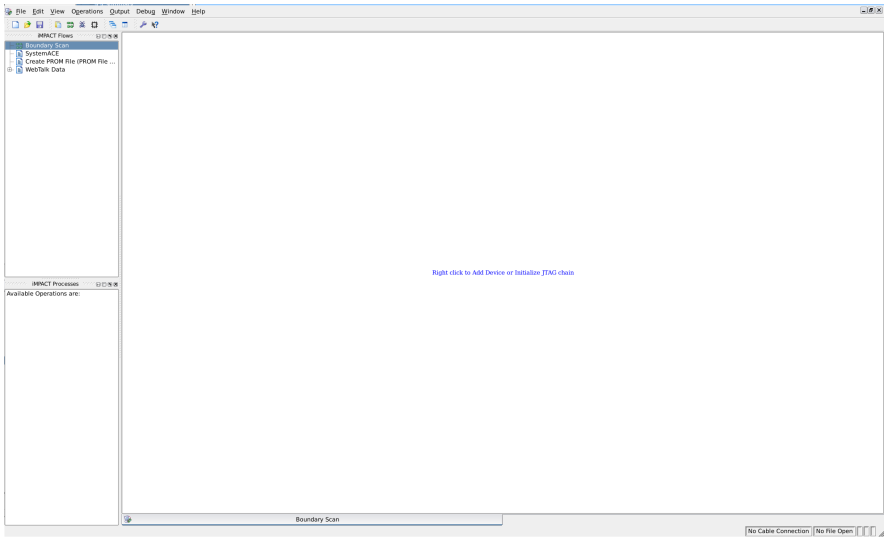
	Pin Number	Signal Name	Pin Usage	Pin Name	Direction	IO Standard	IO Bank Number	Drive (mA)	Slew Rate	Termination	IOB Delay	Voltage	Constraint	IO Register	Signal Integrity
1	AF24	F	IOB33	IO_L20P_T3_12	OUTPUT	LVCMOS33	12	12	SLOW				LOCATED	NO	NONE
2	W23	LED<0>	IOB33	IO_L8P_T1_12	OUTPUT	LVCMOS33	12	12	SLOW				LOCATED	NO	NONE
3	AB26	LED<1>	IOB33	IO_L9P_T1_DQS_12	OUTPUT	LVCMOS33	12	12	SLOW				LOCATED	NO	NONE
4	Y25	LED<2>	IOB33	IO_L10P_T1_12	OUTPUT	LVCMOS33	12	12	SLOW				LOCATED	NO	NONE
5	AA23	LED<3>	IOB33	IO_L11P_T1_SRCC_12	OUTPUT	LVCMOS33	12	12	SLOW				LOCATED	NO	NONE
6	Y23	LED<4>	IOB33	IO_L12P_T1_MRCC_12	OUTPUT	LVCMOS33	12	12	SLOW				LOCATED	NO	NONE
7	Y22	LED<5>	IOB33	IO_L13P_T2_MRCC_12	OUTPUT	LVCMOS33	12	12	SLOW				LOCATED	NO	NONE
8	AE21	LED<6>	IOB33	IO_L19N_T3_VREF_12	OUTPUT	LVCMOS33	12	12	SLOW				LOCATED	NO	NONE
9	AA10	S1	IOB	IO_L14P_T2_SRCC_33	INPUT	LVCMOS15	33				NONE		LOCATED	NO	NONE
10	AB10	S2	IOB	IO_L14N_T2_SRCC_33	INPUT	LVCMOS15	33				NONE		LOCATED	NO	NONE
11	AA13	S3	IOB	IO_L16P_T2_33	INPUT	LVCMOS15	33				NONE		LOCATED	NO	NONE
12	A1			GND											
13	A2			GND											

接下来将文件传送至实验板中检验结果

1. 在 Design 的 View 窗口中选择 Implementation
2. 在 Sources 窗口中选择 `LampCtrl.sch`；在 Processes 窗口中，用鼠标点开 Config Target Device，双击 Manage Configuration Project(iMPACT) 选项，出现 IMPACT 窗口



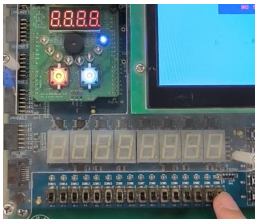
3. 双击 Boundary Scan 弹出下载编辑窗口
4. 鼠标右键菜单里选择 Initialize Chain，系统自动查找已连接在电脑上的开发平台 JTAG 下载链



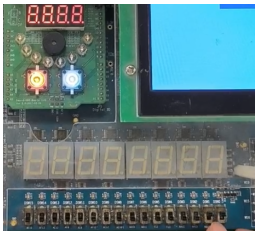
5. 接下来出现 Assign Configuration Files 对话框。这时从文件列表中选择 `lampctrl.bit` 文件，将会为 JTAG chain 上的 xc7k160t 设备指定配置文件；在弹出的 Attach SPI or PRI PROM 对话框弹出，点击 NO 按钮；在弹出的 Device Programming Property 对话框，选择 OK 按钮即可。

6. 右键点击 xc7k160t 设备图标，选择菜单项 Program 后即可对硬件设备进行下载编程

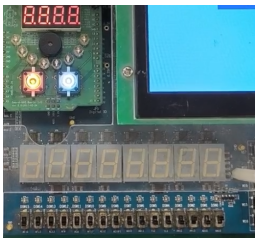
7. 下载后，验证是否满足设计要求



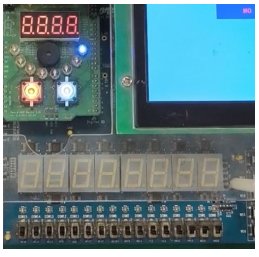
(a) index = 001



(b) index = 011



(c) index = 101

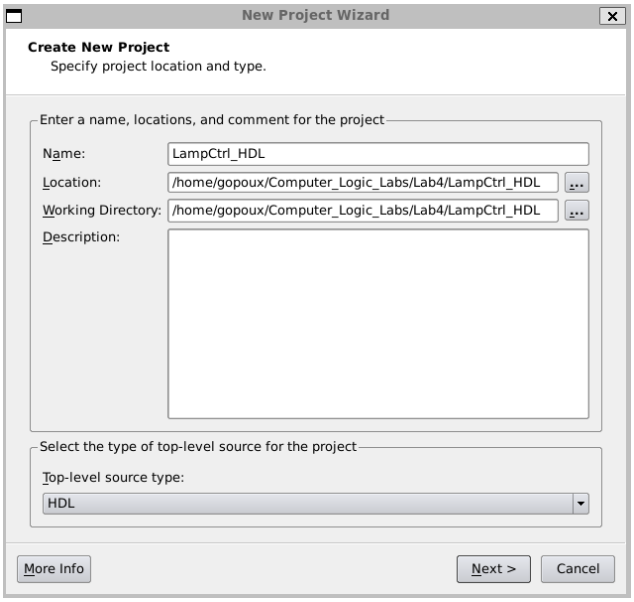


(d) index = 111

4.2 Verilog 代码输入逻辑功能描述

4.2.1 建立楼道控制的工程

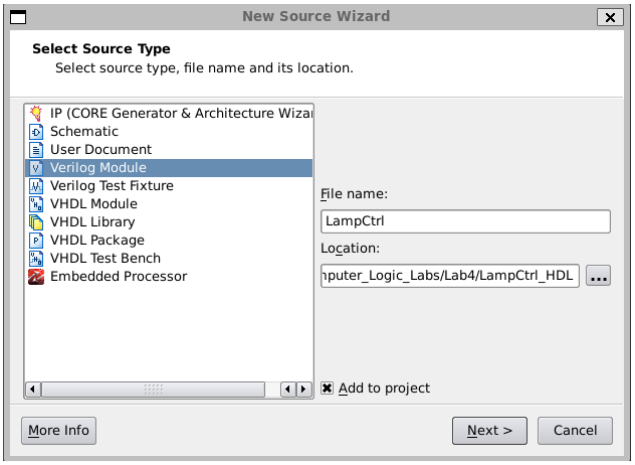
1. 依次点击菜单 File → New Project.
2. 在对话框中设置：
   - Project Name: `LampCtrl_HDL`
   - Top-Level Source Type: HDL



3. 确认后，点击 Next 到设备属性页，设置：
   - Family: Kintex7
   - Device: XC7K160T
   - Package: FFG676
   - Speed: -1
4. 确认后，一直点击 Next 直到创建工程结束。

4.2.2 创建 Verilog 输入源文件

1. 在 Sources 窗口 Sources 选项卡设备型号名处右键菜单选择 New Source.
2. 在新建源文件向导中选择源类型为 Verilog Module，输入文件名`LampCtrl`，勾选 Add to Project



3. 连续点击 Next，最后点击 Finish

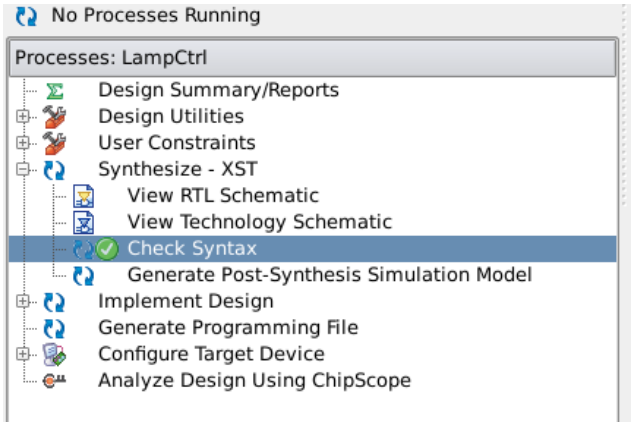
4.2.3 输入楼道灯控逻辑电路 Verilog HDL 代码

1. 在源代码编辑器，输入代码

```
    `timescale 1ns / 1ps
module LampCtrl(
     input wire clk,
    input wire S1,
    input wire S2,
    input wire S3,
    output wire F
    );

    parameter C_NUM = 8;
    parameter C_MAX = 8'hFF;

    reg [C_NUM-1:0] count;
    wire [C_NUM-1:0] c_next;
    wire w;

    initial begin
       count = C_MAX;
    end

    //button pressed
    assign w=S1||S2||S3;
```

```
23
24      //lamp logic
25      assign F = (count < C_MAX) ? 1'b1 : 1'b0;
26
27      //count logic
28      always@(posedge clk)
29      begin
30          if(w == 1'b1)
31              count = 0;
32          else if(count < C_MAX)
33              count = c_next;
34      end
35      //next logic
36      assign c_next = count + 8'b1;
37  endmodule
```

2. 检查输入代码的语法规则，并排除输入错误



3. 延时时间修改

- 仿真时 $\Delta t = (2^8 - 1) \times 20\text{ns} = 5100\text{ns}$

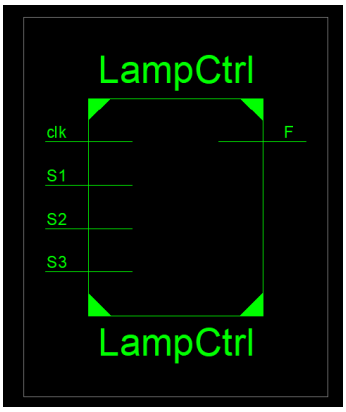
```
1  parameter C_NUM = 8;
2  parameter C_MAX = 8'hFF;
```

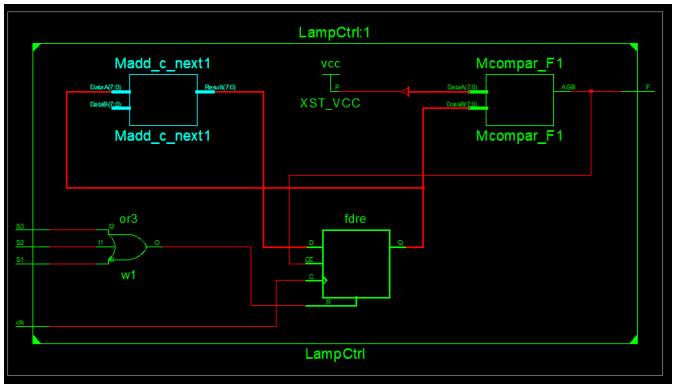
- 下载运行时 $\Delta t = (2^8 - 1) \times \frac{1}{100\text{MHz}} \approx 2.68\text{s}$

```
1  parameter C_NUM = 28;
2  parameter C_MAX = 28'hFFFFFFF;
```

4.2.4 楼道控制电路代码的综合

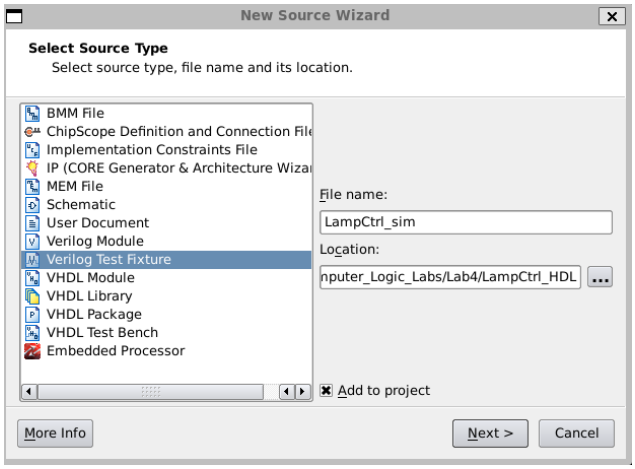
1. 在 Sources 窗口选中文件 `LampCtrl.v`
2. 在 Processes 窗口运行 Synthesis XST → View RTL Schematic
3. 观察综合的电路结构，尝试理解是否与设计目标一致。





4.2.5 建立基准测试波形文件

1. 在 Sources 窗口空白处的右键菜单中选择 New Source
2. 在新建源文件向导中选择源类型为：Verilog Test Fixture，输入文件名 `LampCtrl_sim`，并勾选 Add to Project



3. 点击 Finish 进入 `LampCtrl_sim.v` 编辑窗口

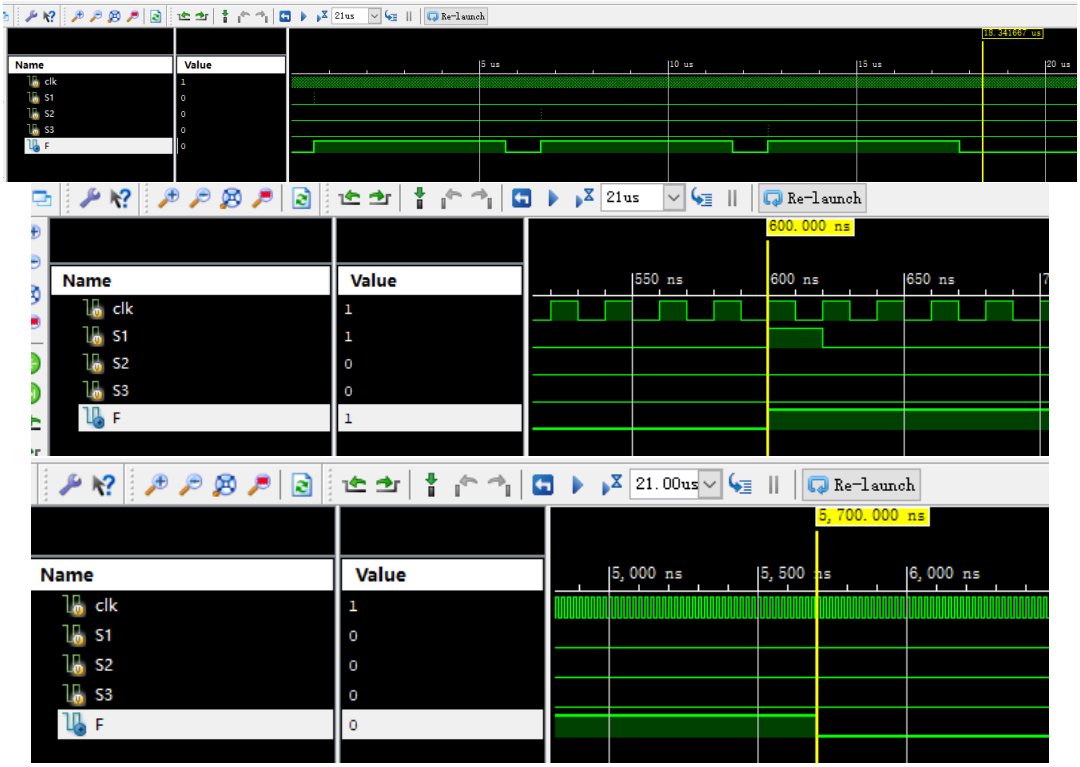
4.2.6 仿真激励输入波形

为便于仿真，`LampCtrl.v` 代码中 `parameter C_MAX = 8'hFF` 。仿真代码如下：

```
`timescale 1ns / 1ps
module LampCtrl_sim;

    // Inputs
    reg clk;
    reg S1;
    reg S2;
    reg S3;

    // Outputs
    wire F;

    // Instantiate the Unit Under Test (UUT)
    LampCtrl uut (
        .clk(clk),
        .S1(S1),
        .S2(S2),
        .S3(S3),
        .F(F)
    );

    initial begin
        // Initialize Inputs
        clk = 0;
        S1 = 0;S2 = 0;S3 = 0;

        #600 S1 = 1;^^I
        #20 S1 = 0;
        #6000 S2 = 1;
        #20 S2 = 0;
        #6000 S3 = 1;
        #20 S3 = 0;
    end

    always begin
        #10 clk = 0;
        #10 clk = 1;
```

```
    end
endmodule
```

仿真波形如下：



4.2.7 建立用户时序约束并为模块的端口指定引脚分配

建立引脚约束文件 k7.ucf，输入代码如下

```
NET "clk" LOC = AC18 | IOSTANDARD = LVCMOS18;
NET "S1" LOC = AA10 | IOSTANDARD = LVCMOS15;
NET "S2" LOC = AB10 | IOSTANDARD = LVCMOS15;
NET "S3" LOC = AA13 | IOSTANDARD = LVCMOS15;
NET "F" LOC = AF24 | IOSTANDARD = LVCMOS33;
NET "LED[0]" LOC = W23 | IOSTANDARD = LVCMOS33;
NET "LED[1]" LOC = AB26 | IOSTANDARD = LVCMOS33;
NET "LED[2]" LOC = Y25 | IOSTANDARD = LVCMOS33;
NET "LED[3]" LOC = AA23 | IOSTANDARD = LVCMOS33;
NET "LED[4]" LOC = Y23 | IOSTANDARD = LVCMOS33;
NET "LED[5]" LOC = Y22 | IOSTANDARD = LVCMOS33;
NET "LED[6]" LOC = AE21 | IOSTANDARD = LVCMOS33;
```

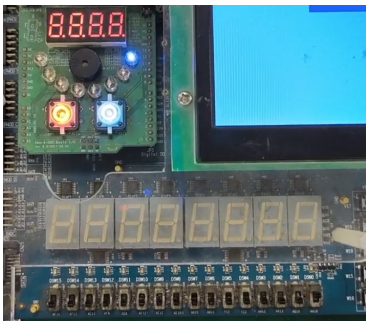
4.2.8 在实验板上运行

将 LampCtrl.v 代码中计数器位数改成 28 位，修改后代码如下：

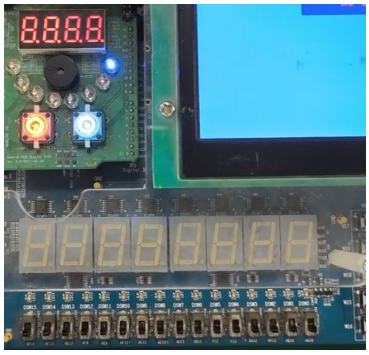
```
`timescale 1ns / 1ps
module LampCtrl(
     input wire clk,
    input wire S1,
    input wire S2,
    input wire S3,
    output wire [6:0] LED,
    output wire F
    );

    parameter C_NUM = 28;
    parameter C_MAX = 28'hFFFFFFF;

    reg [C_NUM-1:0] count;
    wire [C_NUM-1:0] c_next;
    wire w;

    initial begin
       count = C_MAX;
    end

    //button pressed
    assign w=S1||S2||S3;

    //lamp logic
    assign F = (count < C_MAX) ? 1'b1 : 1'b0;

    //count logic
    always@(posedge clk)
    begin
        if(w == 1'b1)
            count = 0;
        else if(count < C_MAX)
            count = c_next;
    end
```

```
36      //next logic
37      assign c_next = count + 8'b1;
38      assign LED = 7'b0000000;
39  endmodule
```

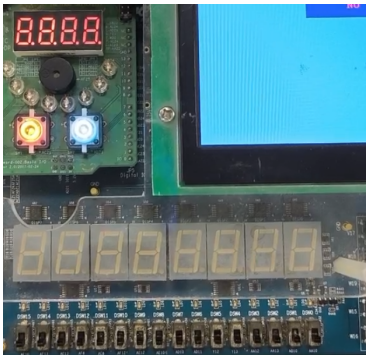
将生成 Bit 文件下载到 SWORD 实验板上物理运行，验证设计是否成功。



(a) 打开开关灯亮起



(b) 关闭开关后灯仍亮起



(c) 一段时间后灯熄灭

5 实验结果分析

5.1 硬件描述代码

查看输入电路的硬件描述代码时,可以发现可以里面的语句和电路图都是相对应的。比如 OR4(四输入或门), AND3(三输入与门)，INV(inverte r，反相器) 还有 VCC(模拟信号源)。这些语句后面的相应代码表示各个门的输入输出。因此生成的 Verilog 代码与预期相符合。

5.2 仿真结果

两个仿真激励输入的代码都是让 (S1, S2, S3) 遍历真值表。可以通过依次表达所有的情况，或者用循环来实现。两种方法生成的仿真波形图一致，与预期相同。

5.3 组合逻辑和时序逻辑

根据逻辑电路的不同特点，数字电路可以分为：组合逻辑和时序逻辑。本次实验分别实现了这两种逻辑电路。

1. 组合逻辑

   组合逻辑的特点是任意时刻的输出仅仅取决于该时刻的输入，与电路原本的状态无关，逻辑中不牵涉跳变沿信号的处理，组合逻辑的 verilog 描述方式有两种：

   (a) `always @(电平敏感信号列表)` 或者 `always @ *` always 模块的敏感列表为所有判断条件信号和输入信号，但一定要注意敏感列表的完整性。在 always 模块中可以使用 if、case

和 for 等各种 RTL 关键字结构。由于赋值语句有阻塞赋值和非阻塞赋值两类，使用阻塞赋值语句 = 更好。always 模块中的信号必须定义为 reg 型。

(b) assign 描述的赋值语句信号只能被定义为 wire 型。

2. 时序逻辑

时序逻辑的特点为任意时刻的输出不仅取决于该时刻的输入，而且还和电路原来的状态有关。电路里面有存储元件（各类触发器，在 FPGA 芯片结构中只有 D 触发器）用于记忆信息，从电路行为上讲，不管输入如何变化，仅当时钟的沿（上升沿或下降沿）到达时，才有可能使输出发生变化；与组合逻辑不同的是：

(a) 在描述时序电路的 always 块中的 reg 型信号都会被综合成寄存器。

(b) 时序逻辑中推荐使用非阻塞赋值 <=

(c) 时序逻辑的敏感信号列表只需要加入所用的时钟触发沿即可，其余所有的输入和条件判断信号都不用加入，这是因为时序逻辑是通过时钟信号的跳变沿来控制的。

6 讨论与心得

本次实验是第一次上板实验。实验之前我就已经进行了预习，但因为 ISE 软件版本太老，在目前大部分系统中都无法顺利运行，尝试了 Windows 10 22H2 、Ubuntu 22.04 LTS 两个系统都无法运行，因此等到实验的时候才用实验室电脑从头开始做，花费了很多时间。实验之后，我尝试使用更老的系统 Ubuntu 18.04 LTS ，才能够在自己的电脑上运行 ISE。希望以后的实验能够顺利一些。