北京航空航天大學

BEIHANG UNIVERSITY

实验三、时序逻辑设计——彩色灯控制

2018 年 11 月 27 日 （版本 v01）

花名册序号	组长标记	教学班	学号	姓名	签名
	*	162321	16231275	刘瀚骋	
提交日期	2018-12-28		签收：	评价日期	报告：
					现场：
					平均（百分制）

高等理工学院

目录

实验三、时序逻辑设计——彩色灯控制（实验指导书部分）

1 实验指导

设计一种通过时序控制彩色灯亮灭的逻辑功能，采用硬件描述语言描述同步时序逻辑电路的方法，体会状态转换和计数器定时，对状态机及顶层可综合模块进行功能仿真，并利用 EELAB-FPGACORE2 实验硬件实验平台（以下简称“实验板”）调试并实现，完成较为完整的 EDA 设计实现过程。

1.1 设计需求

若干个彩色灯泡从 1 依次编号，可以被控制或明或暗，亮暗规则是：初始状态，全部灯泡全暗，第一秒全部灯泡变亮，第二秒，凡编号为 2 的倍数（包括 2）的灯泡的亮暗情况翻转，第三秒，凡编号为 3 的倍数（包括 3）的灯泡的亮暗情况翻转，依次类推。

图 1 彩色灯示意

比如：当有 5 个灯时，其逻辑控制如下所示（0 表示灭，1 表示亮）：

表 1 5 个灯时的控制逻辑

时间(s)	1#灯	2#灯	3#灯	4#灯	5#灯
0	0	0	0	0	0
1	1	1	1	1	1
2	1	0	1	0	1

时间(s)	1#灯	2#灯	3#灯	4#灯	5#灯
3	1	0	0	0	1
4	1	0	0	1	1
5	1	0	0	1	0
0（6，全黑）	0	0	0	0	0
1（7，循环）	1	1	1	1	1

1.2 基本实验要求

基本实验要求 1 为：假设当前一共有 6 盏灯泡，并存在周期性的时钟 CP 输入，要求采用正边沿触发的 D 触发器和适当的门电路，对灯泡的控制逻辑进行实现，灯的控制信号输出依次设为 L_1、L_2 到 L_6。第 7 秒从新进入全黑状态，然后按照前述逻辑循环。

a) 请给出必要的设计说明和逻辑表达式，并绘制电路图；

b) 利用 Verilog，结合门级结构实现灯的控制逻辑（只是控制部分要求采用门级结构，其他部分比如数码管可以自行随意控制）；

c) 采用 Verilog 中的有限状态机（FSM），实现了该灯的逻辑控制；

d) 采用 ModelSim 功能仿真，验证两个模型中关键设计部分的正确性；

e) 要求具有暂停功能，当按下按键后（或者 On 状态），当前灯的控制逻辑停止，再按一下按键（或者 off 状态），从暂停的地方从新开始（灯控制逻辑的最开始状态可以是 off 状态）；

f) 可以切换时钟脉冲，正常情况下由分频的时钟信号进行驱动，在手动模式下，由人按下按键进行时序给定；

g) 采用 FPGA 开发工具实现相应的逻辑功能并加以演示；

h) 对比两种设计方法综合后的结构，并给出进一步的解释；

i) 在实验报告中，请绘制设计的框图（**此项为必选项**）；

j) 可以利用两个数码管中的一个数码管显示时间，另外一个数码管的不同段位显示不同的彩灯的亮灭状态。

基本实验要求 2 为：进一步，考虑 100 盏灯的控制情况，基本控制逻辑不变，解决如下问题：

k) 分析第 100 秒，当所有灯的亮暗控制结束后，继续点亮的灯有几盏（理论分析）；

l) 利用 Verilog，采用行为级方法进行逻辑设计，同样要求有暂停功能和手动功能。

m) 注意：便携式器件上没有那么多资源显示 100 盏灯，可以采用分段显示处理，比如：两个数目管共有 8 个分段（包括 DP），一次最多显示 16 盏，可以采用不同的策略。比如：只显示 1-16 盏灯，或者只显示 17-32 盏灯，用 LED 进行不同段位的灯的标示；或者先显示 1-16 盏灯，再显示 17-32 盏灯，……，最后显示 97-100 盏灯，同样用 LED 进行不同段位的灯的标示。显示方法不做任何限制。

本实验将采取**实验报告和现场汇报演示相结合**的形式，实验报告上交的具体**截止期限**请注意任课教师的通知，实验报告格式请参考本文档；**现场汇报演示**的时间和分组情况请注意任课老师的通知。特此广而告之。

1.3 扩展要求

扩展的实验要求（根据完成的工作量逐次提高评价的等级，但鼓励大家积极地去实验——即：只要至少完成其中一项，即可显著地获得好评，并可根据局部完成的质量酌情提高评价的等级。）

对于 100 盏灯的情况采用定制的手段，即对于 100 盏灯的设计采用基于 PROM 的设计方案，利用 PROM 存储 100 盏灯的亮灭情况，然后在时序的控制下提取存储的信息进行 100 盏灯的控制。其基本框架图如图 2 所示。

其中：

C_part 部分进行时钟信号和 100 盏灯选段信号（图中未给出）生成；

B_part 部分在调整后的时钟信号 clk_mod 的节拍下依次输出地址编码 D_0~D_9，范围为：0~699；

A_part 部分可编程 PROM 根据地址编码查询灯的亮暗控制逻辑，并实施灯亮暗的具体控制。

为了节省灯控制信号，对 100 盏灯采用分段控制的方式，结合灯选段信号，先对 1~16 号灯进行控制，然后对 17~32 号灯进行控制，以此类推，最后对 97~100 号灯进行控制。当一轮结束后，灯全部熄灭，重新开始。

完成以下设计要求：

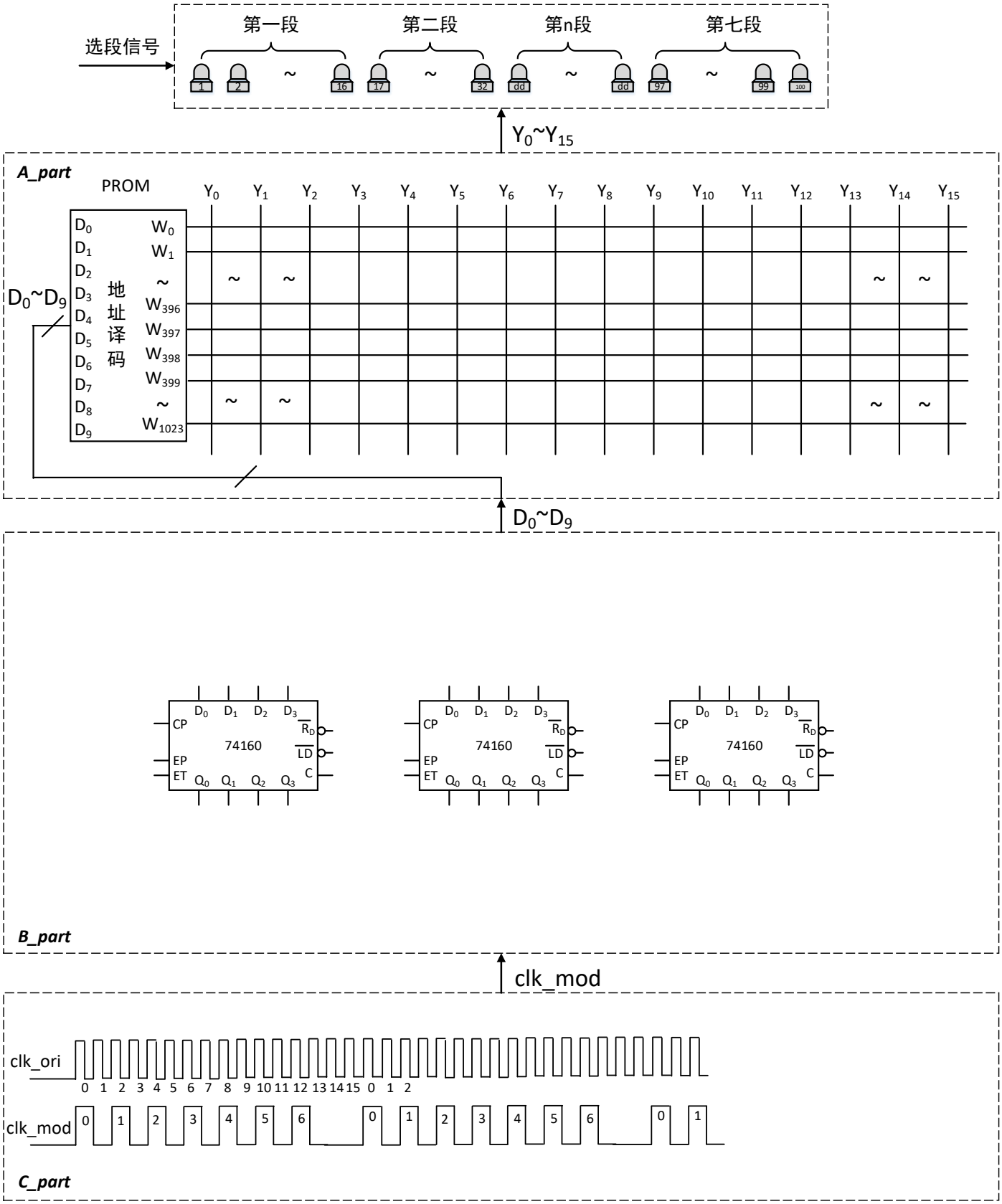

图 2 基于 PROM 的彩色灯控制

n) 结合 C_part 部分时钟信号 clk_ori 和 clk_mod 的调整关系，采用正边沿触发的 D 触发器和适当

的门电路，实现 clk_mod 的信号生成，绘制出电路图；

o) C_part 生成的时钟信号 clk_mod 接入到 B_part 部分进行时序控制，其中，芯片 74160 为同步十进制加法计数器，对其进行整体进位和地址输出设计；

p) A_part 部分 PROM 存储了灯的亮暗控制逻辑，表 2 给出了其存储矩阵的部分存储情况，“1”表示灯亮，“0”表示灯暗。每 7 个地址构成一组，对应 100 盏灯的逻辑控制情况。根据灯的控制逻辑，结合前述分析结果，在 A_part 部分的存储矩阵（即表 2）给出最后 100 秒时刻的 7 个地址对应的或逻辑阵列连接情况；

q) 利用 Verilog，采用自己熟悉方法设计 PROM 和 74160 单位电路功能；

r) 利用 Verilog，通过调用 PROM 和 74160 实现该 100 盏灯的逻辑控制；

s) 同样要求具有暂停功能和手动功能；

t) 增加灯亮灭时间控制功能，比如可以设定为 1s，2s，0.5s，或者任意时间。

表 2 PROM 存储数值

地址输入										数据输出															
D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8	Y_9	Y_{10}	Y_{11}	Y_{12}	Y_{13}	Y_{14}	Y_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	1	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	0	0	0	0	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	0	0	0	0	0	1	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	0	0	0	0	0	1	0	1	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	0	0	0	0	0	1	1	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	0	0	0	0	0	1	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~
~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~
~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~

地址输入										数据输出															
D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8	Y_9	Y_{10}	Y_{11}	Y_{12}	Y_{13}	Y_{14}	Y_{15}
~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~
~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~
~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~
~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~	~
1	0	1	0	1	1	0	1	0	1	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
1	0	1	0	1	1	0	1	1	0	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
1	0	1	0	1	1	0	1	1	1	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
1	0	1	0	1	1	1	0	0	0	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
1	0	1	0	1	1	1	0	0	1	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
1	0	1	0	1	1	1	0	1	0	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
1	0	1	0	1	1	1	0	1	1	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

1.4 其它提示

实验板的开关非常脆弱，如果操作不便，请用按键代替开关，但需要加入防抖电路，并适当修改开灯、关灯信号的逻辑（由电平信号变为脉冲信号）。

“实验板”的 I/O 接口和人机接口资源请参考《digiC2018 课程实验实验指导书(01)》，或参考实验一手册后面的附录。

实验三、时序逻辑设计——彩色灯控制（实验报告部分）

2 实验报告

2.1 需求分析

本次实验要求按照一定的规律控制 5 个或 100 个 LED 灯的亮灭情况。本次实验共完成四项任务

1. 使用门级描述，控制 5 盏灯
2. 使用 FSM，控制 5 盏灯
3. 使用门级描述，控制 100 盏灯
4. 使用规定的控制方式，控制 100 盏灯

2.2 系统设计

2.2.1 总体设计思路

系统顶层负责处理人机交互，响应用户的操作和将计算出的数据发出。

系统顶层会调用四个不同的模块

1. lamb5Control
2. lamb5FSMControl
3. lamb100Control
4. extraLampControl

这四个模块分别对应需求分析(见上方)中的四项实验需求。他们拥有完全相同的接口，即时钟输入，复位输入，状态计数输出及灯泡状态输出。灯牌状态输出将会被发送到显示模块进行显示。

四个模块的设计思路详见 2.2 节内的其他内容。

该部分代码参见 GoldenTop.v

2.2.2 接口设计

输入接口

使用实验板上的硬件实现各个输入接口。二段式拨码开关控制控制信号状态切换的间隔，包括 0.5s

1s 1.5s 的时钟与手动时钟给定。按键一在手动模式下用于控制灯控信号的步进；按键二负责切换暂停与开始。

响应输入信号的模块仅有 2.2.9 灯泡控制时钟生成模块。暂停,手动时钟给定及分频比变化功能亦由该模块提供。

输出接口

由于实验板上的资源不足，故将输出结果通过串口发送到 PC 机，于 PC 机上实现显示功能。

串口模块的实现在实验四中讨论，代码可以参见 Uart.v

PC 机显示模块通过 Django 和 React 实现，与本课程实验无关，在此不多赘述。具体代码可以参见[3]用于演示的 Django 和 React 项目

2.2.3 辅助工具集

为了便于系统的设计与验证，本项目在进行的过程中额外开发了若干辅助工具。

真值表生成器

该工具可用于用于根据实验要求中给定的控制规则及灯泡的数量，计算灯泡的状态并以真值表的形式输出。其中真值表的输出为 n 盏灯的亮灭状态，而输入为自然顺序下的二进制编码。

该工具的具体实现参见 lambTTGenerator.py

真值表化简器

该工具用于将真值表化简为最简与或表达式，并支持以前缀表达式或是 verilog 门级代码片段的形式输出化简结果。该工具可以读取由真值表生成器生成的真值表，并输出 verilog 门级实现代码片段。

该工具的具体实现参见 TT2Verilog.py

PROM 生成器

该工具用于将真值表生成器生成的真值表转换为 16 字的 Verilog 模拟 PROM 代码实现。生成的 Verilog 代码使用 Always 和 case 语句对 PROM 的行为进行模拟。

该工具仅用于完成本次实验的附加任务。

该工具的具体实现参 TT2Prom.py

2.2.4 n 进制计数器模块

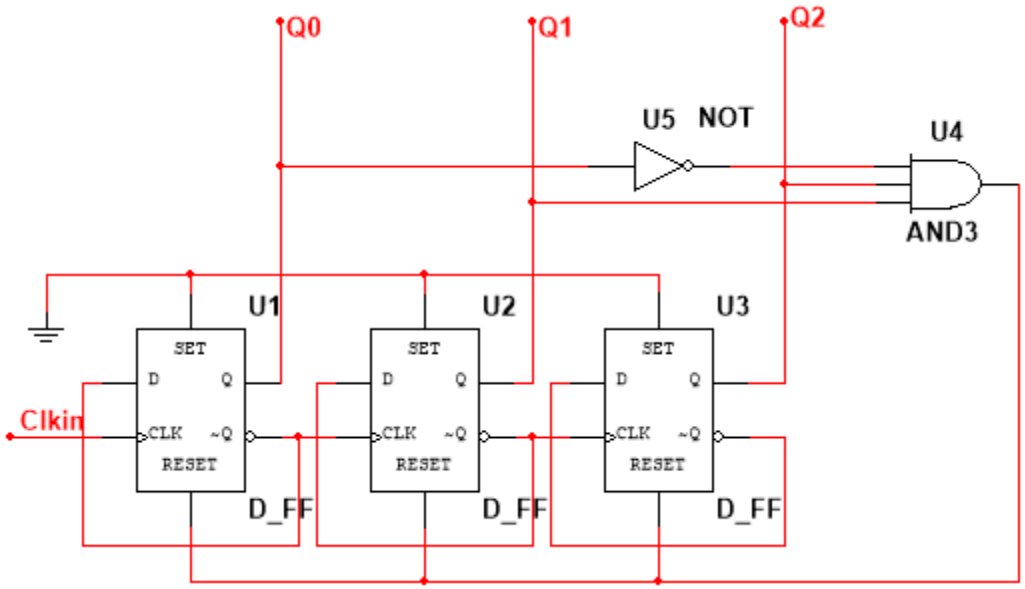

图 1 n 进制计数器电路图

如图 1 n 进制计数器电路图所示，由三个边沿驱动的带有 Reset 和 Set 端口的 D 触发器，采用异步清零法，实现了一个六进制计数器。该部分的代码实现详见 Carry6Counter.。

此外，驱动 100 个 LED 时同样使用了类似的方法，使用七个 D 触发器搭建了 101 进制的计数器，代码实现详见 Carry101Counter.。

2.2.5 基于 FSM 的灯控制模块

实现状态机，使用两个 Always 语句实现。其中一个 always 用于更新当前状态，另一个 always 用于产生驱动 LED 的输出。该状态机和上一个状态(即状态机的输出)有关而且当前状态机输入（高有效的异步复位信号）有关，属于 Melay Machine。

控制模块的状态转换图如图 2 有限状态机实现的状态转换图所示。实现代码参见 Lamb5FSMControl.v

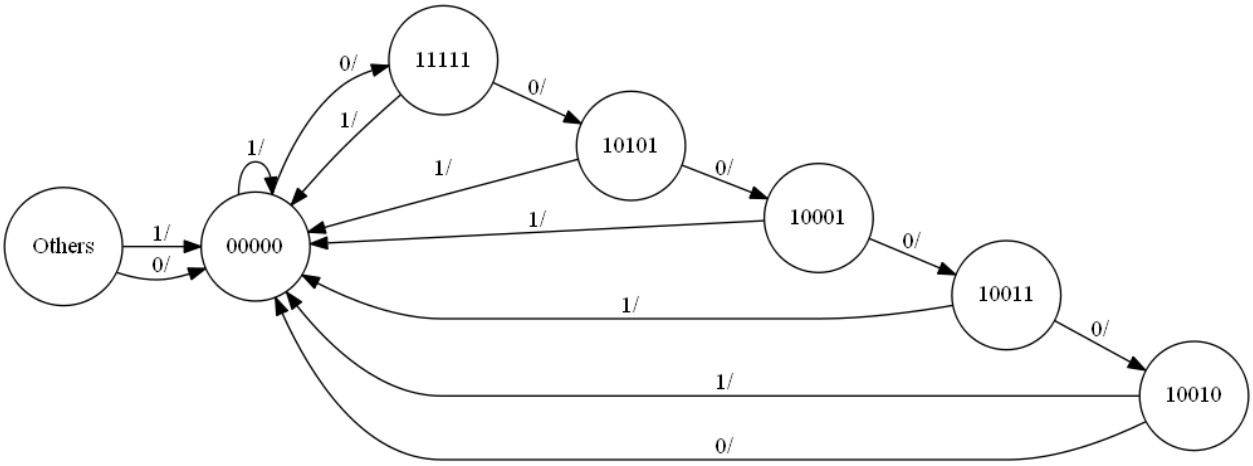

图 2 有限状态机实现的状态转换图

2.2.6 基于门级的灯控制模块

该节阐述基本实验要求 1 中的第一部分，即使用门级建模控制部分电路的设计思路。使用门级实现时，设计可分为时序发生部分和组合逻辑部分。

时序发生部分

对于时序发生部分，从给定的控制规律中可以推断出，对于存在 n 盏灯的情况，总共可能有 n+1 种状态，因此可以使用上升沿触发的 D 触发器构成一个计数器，并结合相应的门电路将计数器设计为异步清零的 n+1 进制计数器，以此产生 n+1 种状态的编码。

在控制模块内部实例化一个 2.2.4 n 进制计数器模块用于提供时序。

组合逻辑部分

当控制 n 盏灯时，存在 n+1 种状态。因此，n 盏灯控制电路组合逻辑部分可以由一个y_1输入，n 输出的真值表完全表示。其中，y_1由下式计算(ceil 表示向上取整)

$$y1 = \text{ceil}[\log_2 n + 1]$$

使用 2.2.3 辅助工具集编写的工具首先生成灯控制信号的真值表，其次用其中的工具化简真值表得到门级表示的 Verilog 代码片段。引出门级实现的输出作为灯控信号。

控制 5 盏灯时的代码参见 Lamb5Control.v

控制 100 盏灯时的代码参见 Lamb100Control.v

2.2.7 附加实验任务模块

C Part

C Part 时钟生成模块可以由 D 触发器构成 Ripple Counter，然后结合一定的组合逻辑输出要求的波形。

Ripple Counter 复用 2.2.4 n 进制计数器模块。额外的门电路由 2.2.3 辅助工具集中的工具进行设计。

电路图见图 3 C Part 电路图

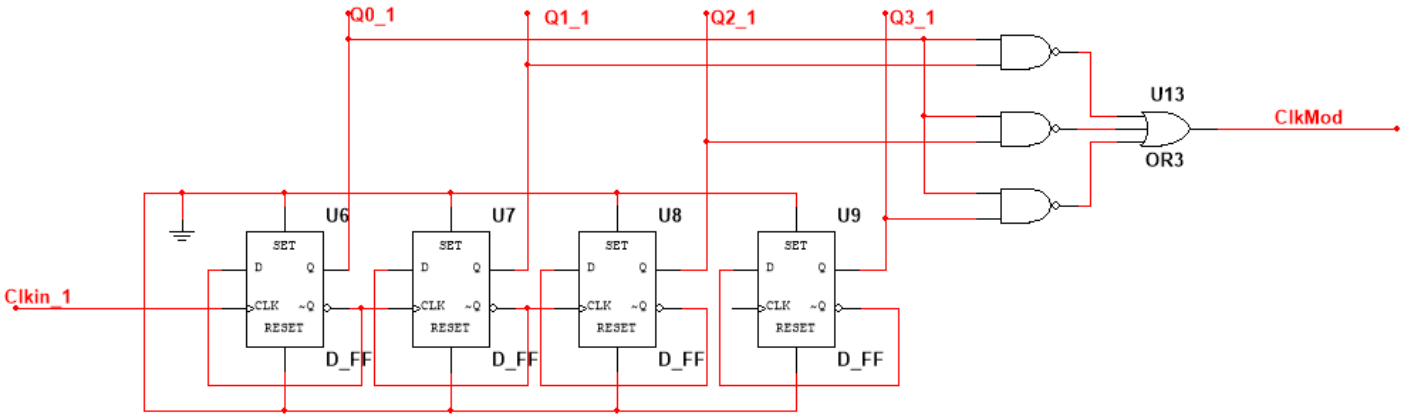

图 3 C Part 电路图

代码实现参见 CPartClockGenerator.v

B Part

B Part 首先使用门级描述建模 74HC160，然后将三个 74HC160 串接，使用异步清零法，配合相应的组合逻辑构成 700 进制计数器。

由于 74HC160 输出的为二进制 BCD 码，故还需将 BCD 码转换成 10 位的二进制地址。使用纯组合逻辑实现这部分的转换。BCD 转换部分和计数器部分在同一个 module 中实现。

BPart 的电路图如图 4 74HC160 700 进制计数器所示，实现代码参见 BPartCounter.v

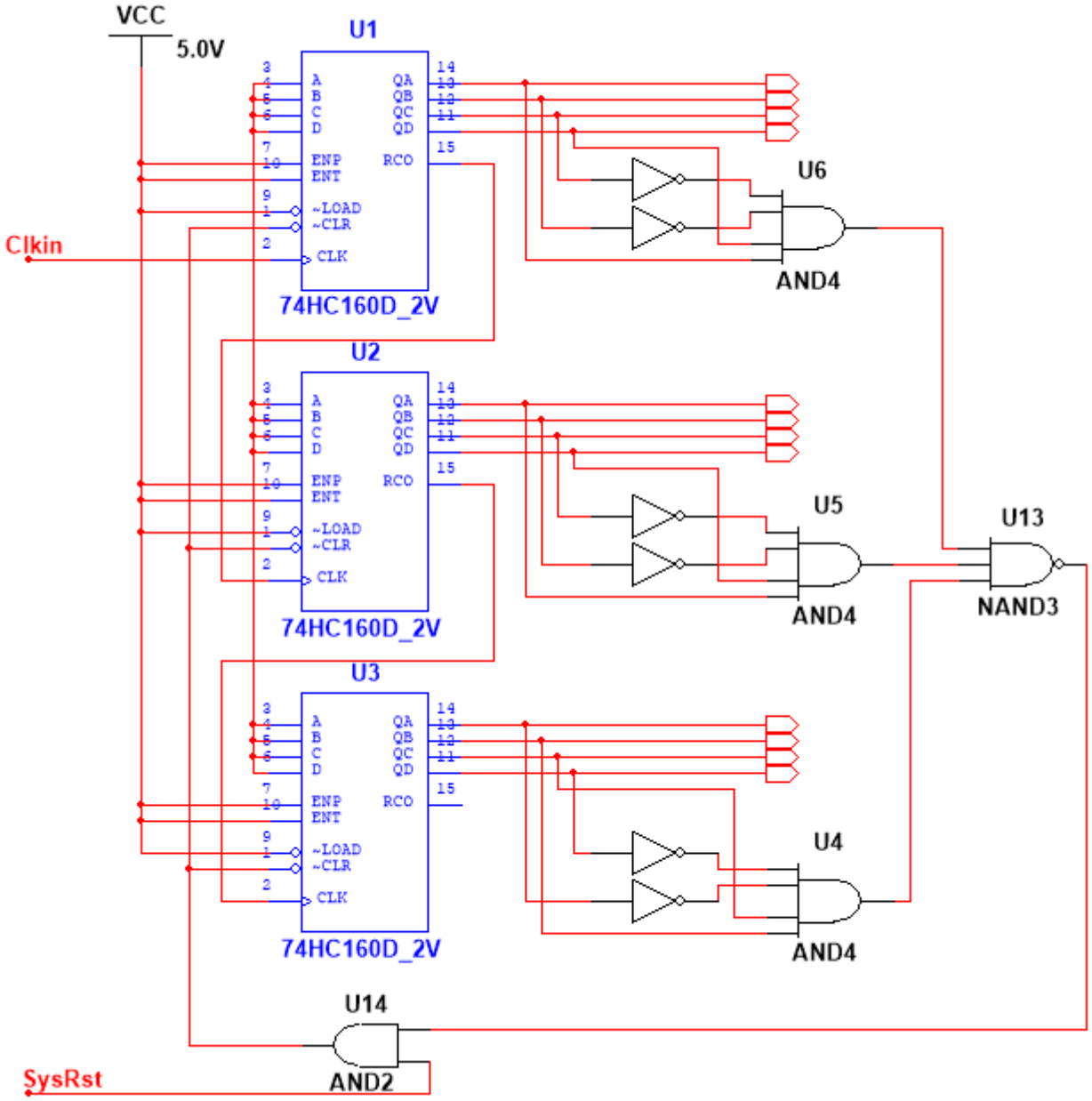

图 4 74HC160 700 进制计数器

A Part

A Part 使用 verilog 中的 always 和 case 语句模拟 PROM 的行为。设计了一个单周期的 ROM，地址线为 9 位 bit,16bit。其中，前 700 个字分别存储 1-7 段灯在第 0-99 个时间单位时的亮灭情况。

使用了 2.2.3 辅助工具集中的工具生成了符合应用需求的 PROM 代码，具体实现参见 Lamp100Prom.v

2.2.8 基于 74HC160 和 PROM 的灯控制模块

该模块用于将 2.2.7 附加实验任务模块的三个模块互联，并提供和 2.2.4 n 进制计数器模块 2.2.5 基于 FSM 的灯控制模块相同的接口方便顶层模块调用。

具体实现代码参见 extraLampControl.v

2.2.9 灯泡控制时钟生成模块

灯泡时钟生成模块内部读取输入的两位信号（由开发板上的 switch）指定，当输入分别是 2'b00 2'b01 2'b10 时将时钟输入连接到系统时钟，并分别设置了不同的分频比，产生周期为 0.5s 1s 1.5s 的时钟信号；当输入为 2'b11 时，将时钟输入连接到按键一，并设置分频比为 0 实现手动控制。

该模块实现了暂停/时钟分频比可调/可以手动控制时钟三项功能。

此外，该模块还监测按键二的输入。当按键二信号出现上升沿（被按下）时，翻转内部的一个用于标记暂停状态的寄存器。当该寄存器被置位时认为系统处在暂停状态，将停止时钟输出。

这部分的代码参见 clockGenerator.v

2.2.10 串口通信模块

串口通信模块实现了满足部分 Avalon Streaming 总线协议的内部接口。具体实现在实验四中阐述。

代码可参见 Uart.v

2.3 功能仿真测试

按照实验要求，本节中的所有仿真均是编写基于 Verilog 的 Testcase，然后在 Modelsim 中进行手动仿真。注意在 ISE 开发环境中，首次调用 Modelsim 仿真前，需要在 ISE Commander 中执行以此 compxlib 对 Xilinx 器件库进行编译。

2.3.1 Ripper Counter 仿真

2.3.2 n 进制技术器仿真

在对 Ripper Counter 进行仿真时，请注意建模用于产生异步清零信号的门电路时，需要为门电路声明额外的传播时延。否则 Modelsim 会将该门电路视为理想的无传输时延的门，进而导致产生一个仿真器无法仿真的脉冲信号，使计数器无法正确清零。

在门级建模时，描述额外时延的文法如下例所示

```
and #(1) U1(foo,bar1,bar2);
```

该文法将会创建一个传播时延为 1 个时间单位的与门。

6 进制计数器仿真

6 进制计数器的仿真结果如图 5 6 进制计数器仿真结果所示

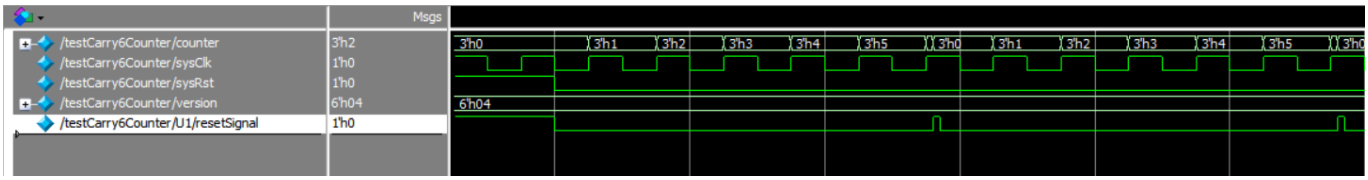

图 5 6 进制计数器仿真结果

异步信号进行清零操作的细节如图 6 6 进制计数器置位细节所示

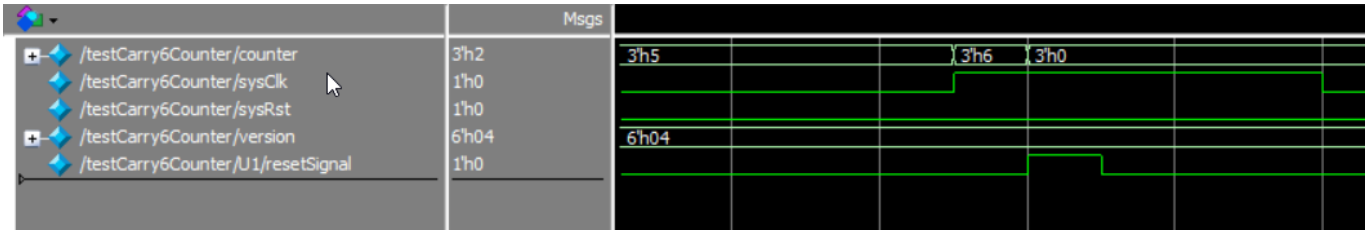

图 6 6 进制计数器置位细节

由上两图，可以看出，当计数器计数到 6 时，额外的组合逻辑产生一个清零信号将计数器清零。

101 进制计数器仿真

101 计数器的仿真结果如图 7 101 进制计数器仿真结果所示

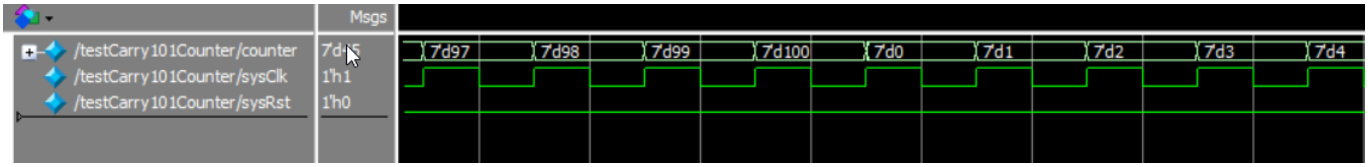

图 7 101 进制计数器仿真结果

2.3.3 灯泡控制逻辑仿真

基于门级实现和 FSM 实现的控制模块接口相同，二者使用相同的 TestCase 进行测试。

基于门级实现的控制逻辑

n=5 时，基于门级实现的仿真结果如图 8 n=5 门级实现仿真结果所示

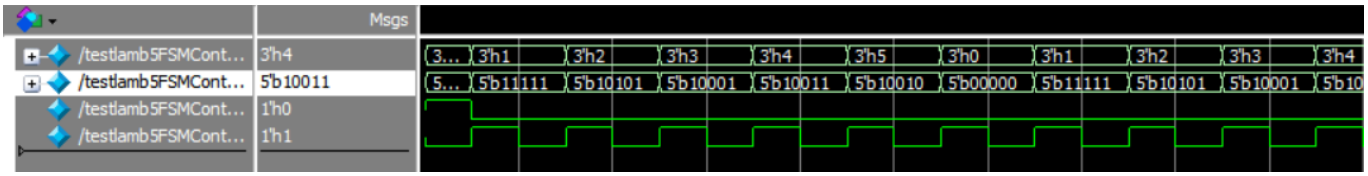

图 8 n=5 门级实现仿真结果

n=100 时，基于门级实现的仿真结果如图 9 n=100 循环起始状态，图 10 n=100 仿真循环结束状态所示

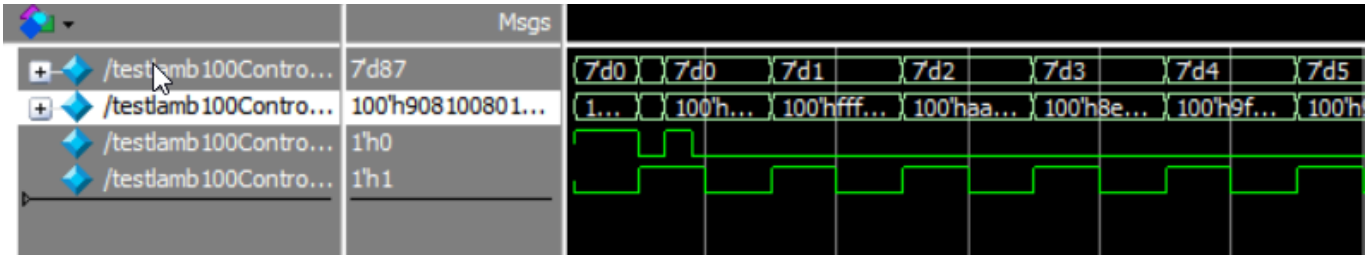

图 9 n=100 循环起始状态

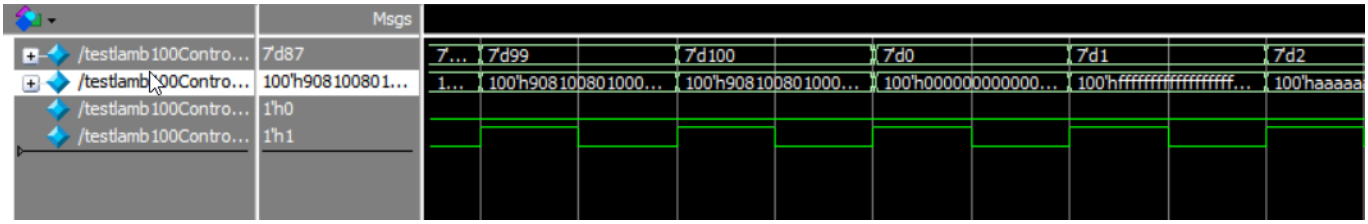

图 10 n=100 仿真循环结束状态

基于有限状态机实现的控制逻辑

n=5 时，基于 FSM 的仿真结果如图 11 n=5FSM 实现仿真结果所示

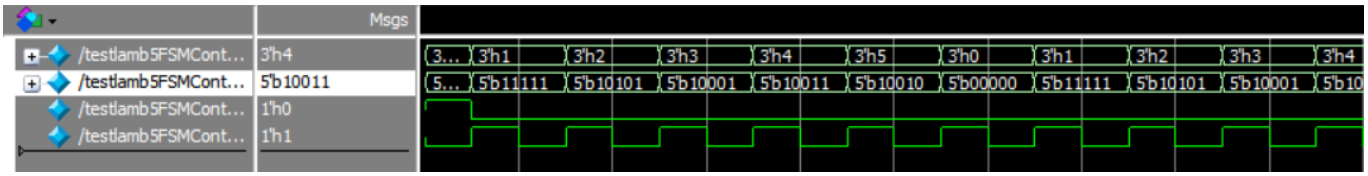

图 11 n=5FSM 实现仿真结果

附加任务部分仿真

Bpart 仿真

74160 模块仿真

74160 模块的仿真结果如图 12 74HC160 功能仿真结果所示

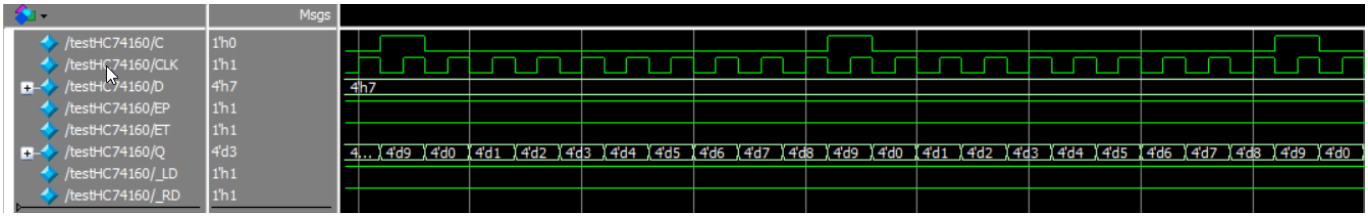

图 12 74HC160 功能仿真结果

此外，还测试了 74160 的预置数功能，如图 13 74HC160 预置数功能仿真所示 ，将初始值置为 7.

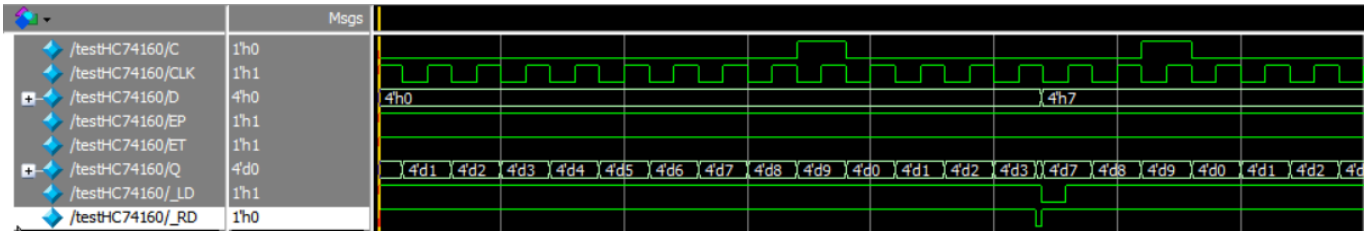

图 13 74HC160 预置数功能仿真

整体仿真

Bpart 整体仿真结果如图 14 B Part 整体仿真所示。可见计数器可被正常复位，且 BCD 到 2 进制转换器正常工作。

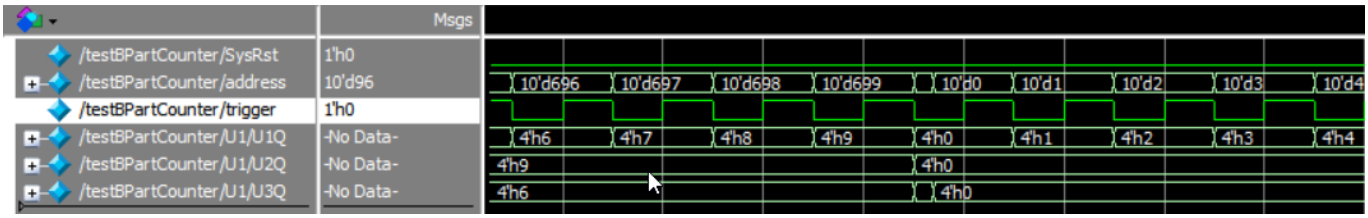

图 14 B Part 整体仿真

Cpart 仿真

Cpart 整体仿真结果如图 15 C Part 仿真结果所示，可见输出了题目要求的时钟信号。

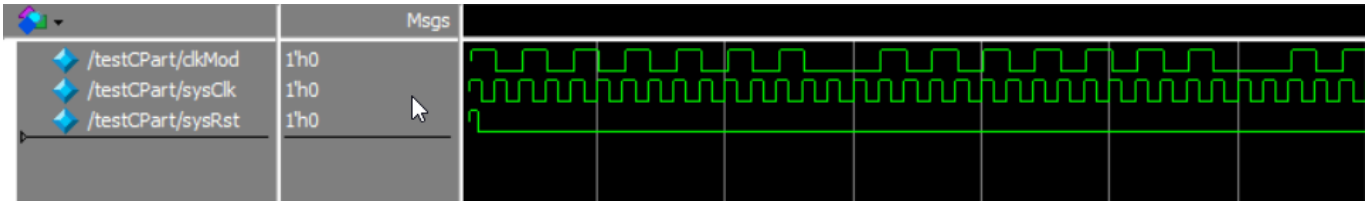

图 15 C Part 仿真结果

2.3.4 报告问题回答

问: 分析第 100 秒，当所有灯的亮暗控制结束后，继续点亮的灯有几盏（理论分析）;

答: 10 盏

问: 请给出 PROM 最后几位存储的数据

答:

10'h2af:data=16'b0000000010000000;//STATE:99 SEG:2 COUNTER:687

10'h2b0:data=16'b0001000000000000;//STATE:99 SEG:3 COUNTER:688

10'h2b1:data=16'b1000000000000001;//STATE:99 SEG:4 COUNTER:689

10'h2b2:data=16'b0000000000000000;//STATE:99 SEG:5 COUNTER:690

10'h2b3:data=16'b1000000000000000;//STATE:99 SEG:6 COUNTER:691

10'h2b4:data=16'b0000000000000000;//STATE:99 SEG:7 COUNTER:692

10'h2b5:data=16'b1001000010000001;//STATE:100 SEG:1 COUNTER:693

10'h2b6:data=16'b0000000010000000;//STATE:100 SEG:2 COUNTER:694

10'h2b7:data=16'b0001000000000000;//STATE:100 SEG:3 COUNTER:695

10'h2b8:data=16'b1000000000000001;//STATE:100 SEG:4 COUNTER:696

10'h2b9:data=16'b0000000000000000;//STATE:100 SEG:5 COUNTER:697

10'h2ba:data=16'b1000000000000000;//STATE:100 SEG:6 COUNTER:698

10'h2bb:data=16'b0001000000000000;//STATE:100 SEG:7 COUNTER:699

2.4 设计实现

2.4.1 综合和下载过程

点击 ISE 开发环境中的综合按钮进行综合。

使用 IMPACT 软件进行下载。

2.4.2 实验关键结果及其解释

实验结果采用现场答辩的方式进行演示，此处列出一些 UI 的照片。

任务一及任务二的测试最终结果如图 16 任务一/二 实验结果所示，展示了五盏灯在 Counter=2 时，偶数灯熄灭时的情景

图 16 任务一/二 实验结果

任务三的测试最终结果如图 17 任务三实验结果所示，展示了 Counter=2 时偶数灯泡被熄灭时的情景，见

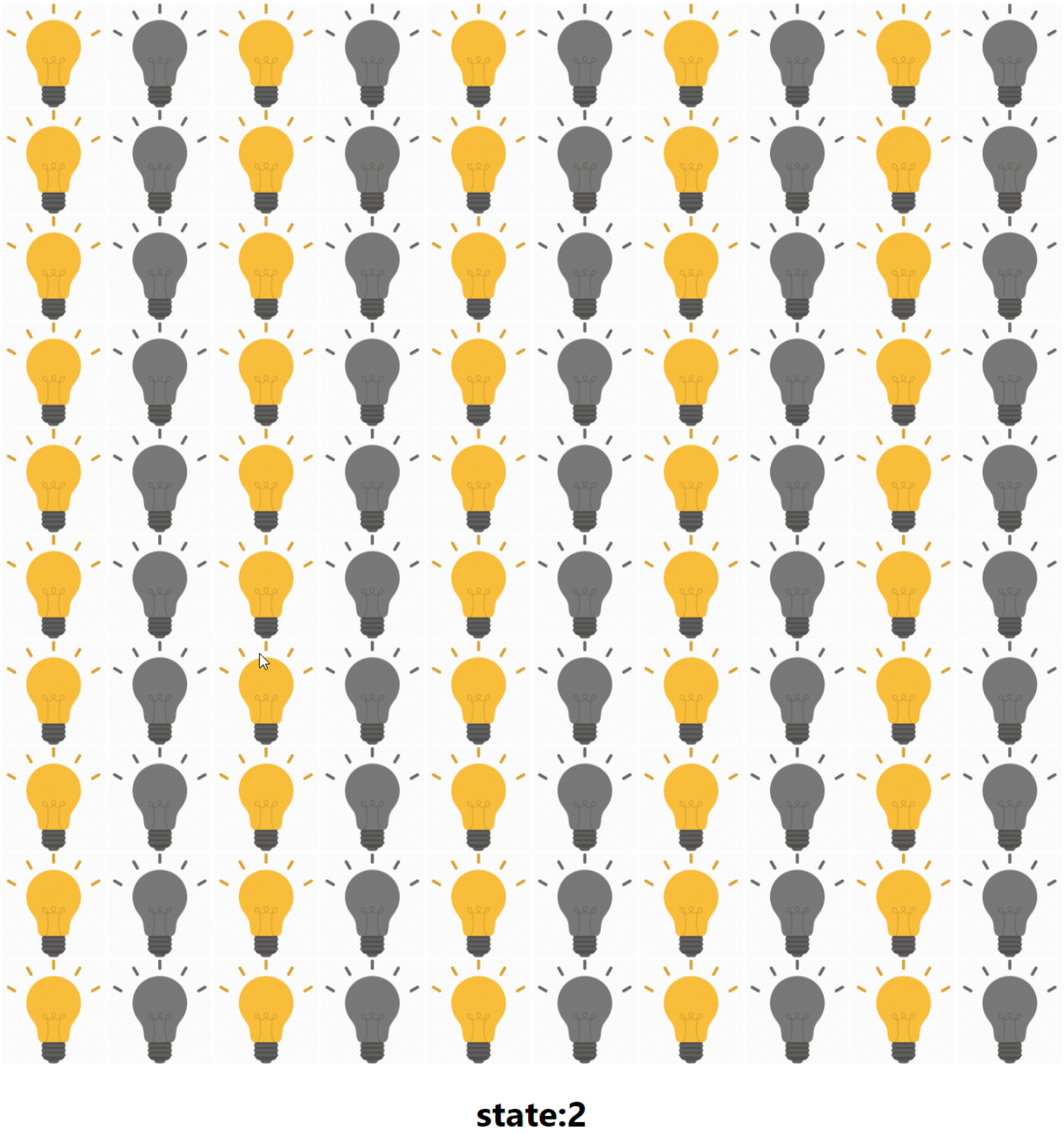

图 17 任务三实验结果

任务四的测试最终结果如图 18 附加实验结果所示，展示了 Counter=1 时第 5 段偶数灯泡被熄灭时的情景

图 18 附加实验结果

综合结果对比

按照实验要求，对比使用门级实现和使用 FSM 实现所消耗的系统资源数对比。

使用门级实现时综合结果见图 19 门级实现综合结果

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	4,434	9,312	47%	
Number of 4 input LUTs	3,117	9,312	33%	
Number of occupied Slices	3,698	4,656	79%	
Number of Slices containing only related logic	3,698	3,698	100%	
Number of Slices containing unrelated logic	0	3,698	0%	
Total Number of 4 input LUTs	3,251	9,312	34%	
Number used as logic	3,117			
Number used as a route-thru	134			
Number of bonded IOBs	22	66	33%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	3.51			

图 19 门级实现综合结果

使用 FSM 实现时综合结果见图 20 FSM 实现综合结果

对比图 19 门级实现综合结果，图 20 FSM 实现综合结果，可见两种实现方式下的资源占用并无区别。可能是 ISE 自带的综合算法对于两种不同的文法优化后，综合出了相同的结果。

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Total Number Slice Registers	4,442	9,312	47%	
Number used as Flip Flops	4,437			
Number used as Latches	5			
Number of 4 input LUTs	3,121	9,312	33%	
Number of occupied Slices	3,703	4,656	79%	
Number of Slices containing only related logic	3,703	3,703	100%	
Number of Slices containing unrelated logic	0	3,703	0%	
Total Number of 4 input LUTs	3,259	9,312	34%	
Number used as logic	3,121			
Number used as a route-thru	138			
Number of bonded IOBs	22	66	33%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	3.51			

图 20 FSM 实现综合结果

2.4 小结

本次实验，加深了我对时序电路的理解，提高了我的 Verilog 的编码水平。

参考文献

参考代码列表

[1] GoldenTop.v

[2] Uart.v

[3] 用于演示的 Django 和 React 项目

[4] TT2Prom.py

[5] lambTTGenerator.py

[6] TT2Verilog.py

[7] Carry101Counter.v

[8] Carry6Counter.v

[9] Lamb5FSMControl.v

[10] Lamb5Control.v

[11] Lamb100Control.v

[12] CPartClockGenerator.v

[13] BPartCounter.v

[14]Lamp100Prom.v

[15]extraLampControl.v

[16]clockGenerator.v