廣東工業大學

实验报告

课程名称__EDA 技术_____

实验名称__基于 Verilog 的程序设计__

学生学院__集成电路学院 _____

专业班级__20 级集成电路 2 班_____

学　　号__3120002522_____

学生姓名____林铮_____

2023 年　6　月　22　日

EDA 课程设计

课程设计题目一：基于 Verilog 的跑马灯设计

设计目的：

设计一个能够有多种工作模式控制的 8 个灯亮灭的电路。

设计内容：

程序设计要求：

设计一个包含四个工作状态的状态机，每种工作模式重复两次，并在 2 分钟内完成所有工作模式。

工作模式以及内容如下：

工作模式 1：按照从左到右的方向，依次点亮每一盏灯，然后依次熄灭每一盏灯；

工作模式 2：分成两组灯，前四个灯为 1 组，后四个为 2 组，1 组灯按从左到右依次点亮，同时 2 组灯按从右到左依次点亮，然后两组灯按各自点亮的顺序依次熄灭；

工作模式 3：用 11110000 作为一组灯的序列，按照该顺序完成 8 盏灯亮灭：即首先灯 1 亮，然后灯 2 亮，然后灯 3 亮，然后灯 4 亮，然后灯 5 不亮，然后灯 6 不亮，然后灯 7 不亮，然后灯 8 不亮，然后八个灯同时变成亮，亮，亮，亮，不亮，不亮，不亮，不亮，并保持下去。

工作模式 4：与工作模式 1 相同，但初始序列列为 11111111。

设计源文件模块输入信号：

选择信号[1:0]S，时钟信号 clk，复位信号 rst

设计源文件模块输出信号：

跑马灯亮灭信号[7:0]Y

Verilog 源文件名：

Running_Light.v

Verilog Testbench 名：

Running_Light_top. v

仿真软件：

Vivado Modelsim

设计要求分析：

由数电知识可以知道 LED 即发光二极管，可以通过共接阴阳极的方式来实现高电平/低电平控制，本次实验要输出发光的模型是共阴极 LED 模型，即“0000 0000”为 LED 全暗，“1111 1111”为 LED 全亮。我们设从 LED 左到右的对应输出 Y 最高位到最低位。

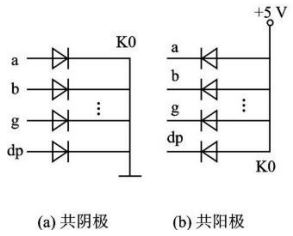

图 1 LED 两种共极连接方式的电路原理图

要实现四个工作模式的转变，首先在模块块内定义一个三位的寄存器[2:0]sel，一个作用是接收输入的 S，另一个是使用最高位来判别是否需要再循环一次。如果输入的 S 为 00，则在运行完一次工作状态 1 的块程序后，状态机会根据另一个寄存器(model_repeat)的值来判定是否需要重复。若 model_repeat 的值为 0，则给其+1 进位后状态跳转至暂存态(model_temp1, 3’b100)后再跳转回 Model_1，再一次运行完一次工作模式 1 的块程序后，跳转至工作模式 2。至完成工作模式 1-4 后各两次后，跳转回工作状态 1。

图 2 选择信号 S 状态机状态转移示意图

设计内容：

工作状态 1 分析：

首先，进入工作状态 1，输出初值为 8'b0000 0000,可以知道要实现 8 个灯点灭，要进行 16 次操作，前 8 次和后 8 次分别均为点灯和灭灯. 故可以在块外定义一个 8 位寄存器[7:0] state_exist，state_exist 由计数器块决定，当 state_exist 在 0-7 之间时，易知使用“Y_out[7-state_exist]<=1;”即可完成从左往右点灯操作，当 state_exist 在 8-15 之间时，需要灭灯只需将上式改成“Y_out[15-state_exist]<=0;”即可实现从左往右灭灯。

工作状态 2 分析：

从工作状态 1 进入工作状态 2，输出初值为 8'b0000 0000,可以知道要实现 8 个灯每次两个灯点灭，只需进行 8 次操作，前 4 次和后 4 次分别均为点灯和灭灯。故继续使用 state_exist。当 state_exist 在 0-3 之间时，易知使用“Y_out[state_exist] <= 1;”和“Y_out[7-state_exist]<= 1;”即可完成从两侧往中间点灯操作. 当 state_exist 在 4-7 之间时，需要灭灯只需改成“Y_out[state_exist-4]<=0 和“ Y_out[11-state_exist]<=0;”即可完成从两侧往中间灭灯操作。

工作状态 3 分析：

从工作状态 2 进入工作状态 3，输出初值为 8'b1111 0000，可以知道要实现 8

个灯点灭，要进行 8 次操作，前四次和后四次分别均为点灯和灭灯，故继续使用 state_exist 当 state_exist 在 0-3 之间时，易知使用“Y_out[7-state_exist]<=1;”即可完成从左往右点灯操作，当 state_exist 在 4-7 之间时，需要灭灯只需将上式改成“Y_out[15-state_exist]<=0;”即可实现从左往右灭灯。

工作状态 4 分析：

从工作状态 3 进入工作状态 4，输出初值为 8’1111 1111,可以知道要实现 8 个灯点灭，要进行 16 次操作，前 8 次和后 8 次分别均为灭灯和点灯，故继续使用 state_exist 当 state_exist 在 0-7 之间时，易知使用“Y_out[7-state_exist]<=0;”即可完成从左往右灭灯操作，当 state_exist 在 8-15 之间时，需要灭灯只需将上式改成“Y_out[15-state_exist]<= 1;”即可实现从左往右点灯。

计数器分析：

在主 always 块外，再另建一个 always 作为计数器。同主 always 块一样 clk 上升沿触发，rst 下降沿触发。当 rst 高电平时，开始计数，计数 Count_time 时间后给 state_exist 加 1 再将 Count_time 复位为 0。当主触发块不再需要 state_exist 再增加时，将 count_rst 置 1，即使计数器复位。

RTL 图：

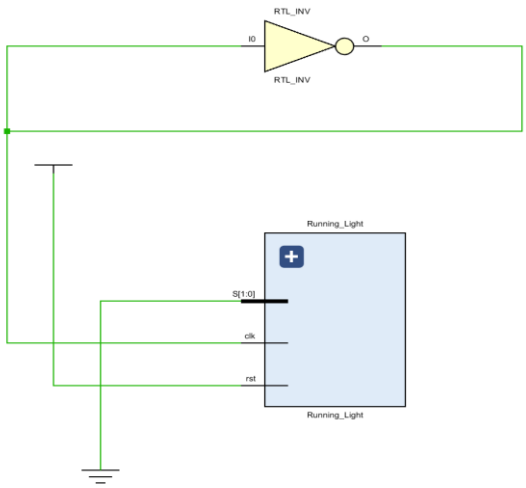

图 3 顶层模块 RTL 图

可以看到顶层模块的 RTL 图显示，Running_Light 输入为三个端口，即 clk，S[1:0],rst。时钟信号是由外界激励不断反向产生的结果。

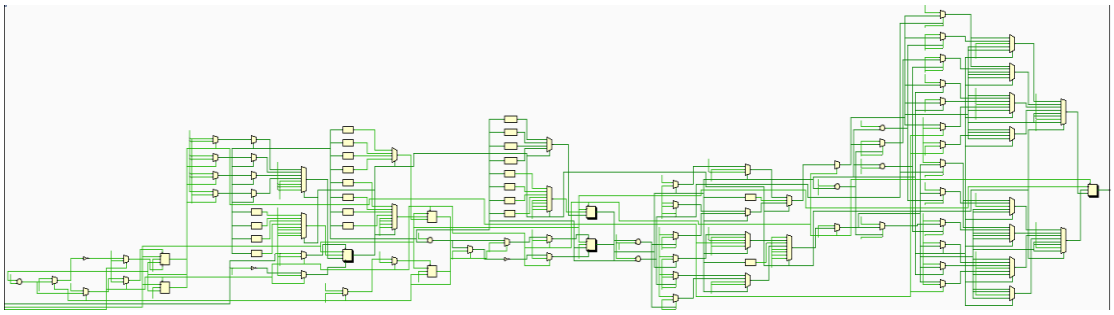

图 4 Running_Light 模块 RTL 图

可以看到 Running_Light 模块的 RTL 图，显示模块使用了较多的寄存器和多路选择器和触发器，可能存在较高的延迟。

设计结果分析

仿真整体结果分析：

Vivado 仿真：

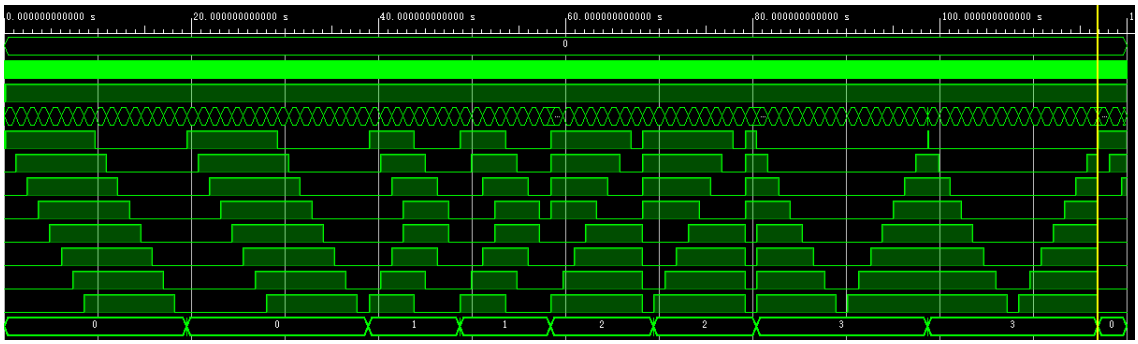

图 5 Vivado 仿真 四个工作模式波形图

可以看到 Vivado 仿真出的四个工作模式的波形图，不难看到完成四个工作模式所需时间约为 116 s，四个工作模式均重复两次。

Modelsim 仿真：

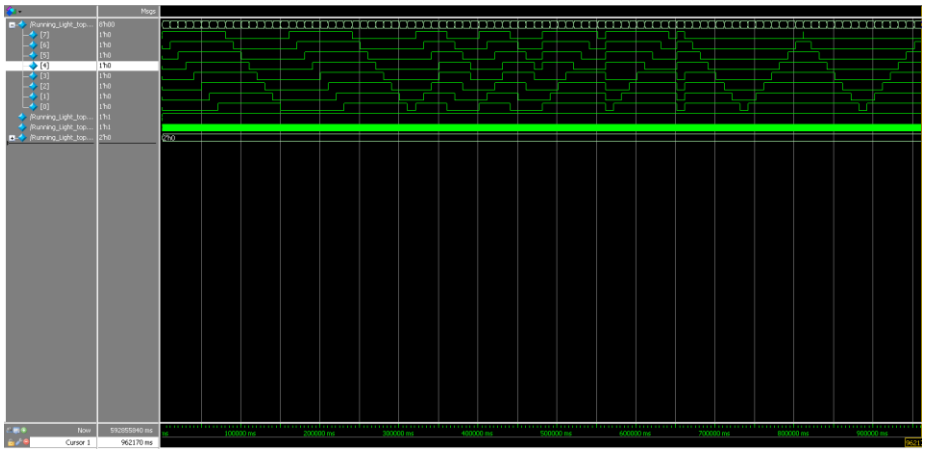

图 6 ModelSim 仿真 四个工作状态波形图

可以看到 ModelSim 仿真出的四个工作模式的波形图，不难看到完成四个工作模式所需时间约为 96.2 s。相较于 Vivado 仿真的 116s，推测可能是 Vivado 仿真使用的是虚拟 FPGA 资源，ModelSim 直接使用计算机资源。

工作模式分析

各工作模式形图从上到下分别为时钟信号 clk，复位信号 rst，输出信号 Y[7:0](从上到下分别是 7-0)和选择信号[2:0]sel。

工作模式 1 工作结果分析：

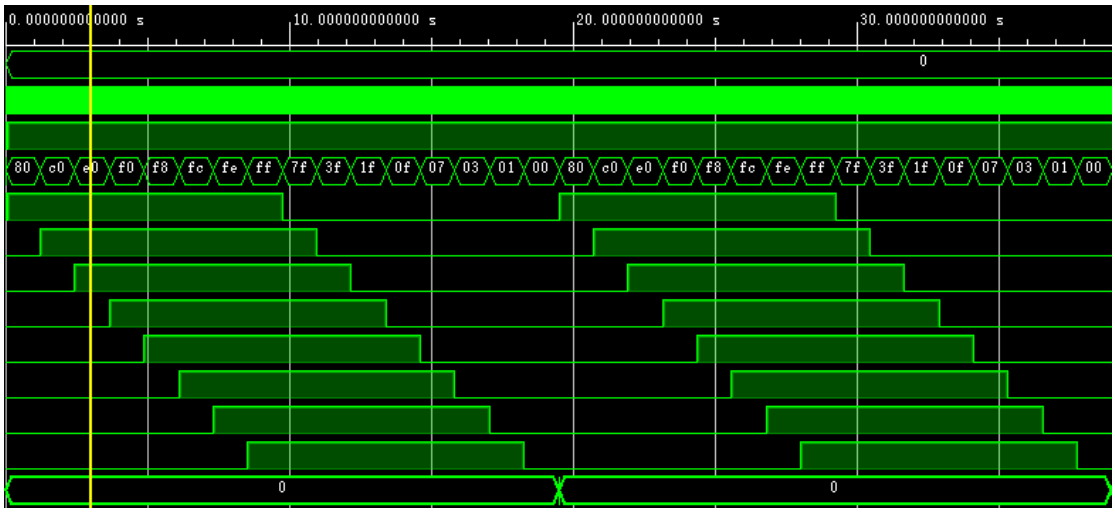

图 7 工作状态 1 波形图

工作模式 1 最初输出为 8’d0000_0000，然后是变成 8’d1000_0000，实现了最高位灯亮，即最左边灯亮。再然后是变成 8’d1100_0000，即次最左灯亮。以此类推，到最右侧灯亮，即输出变为 8’d1111_1111 时，全部灯都被点亮。完成点灯后是灭灯操作，首先是 Y_out[7]转变成 0，即最左侧灯灭。也可以以此类推，到最右侧灯灭，输出变为 8’d0000_0000，即全部灯灭。在完成一次工作模式 1 后，选择信号跳变到 3’b100(即暂存态 Model_temp1)后经过一个时钟周期，在变成工作模式 1，再进行一次工作模式 1 的内容。

工作模式 2 工作结果分析：

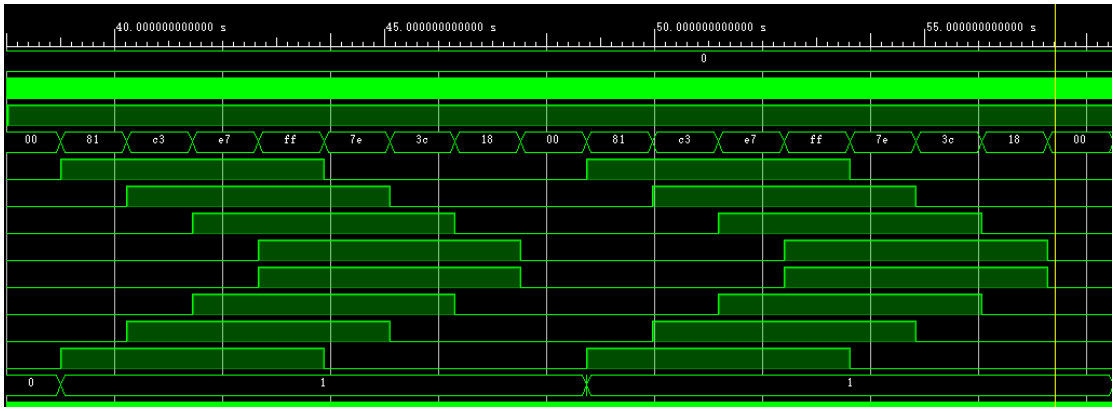

图 8 工作状态 2 波形图

工作模式 2 最初输出也为 8’d0000_0000，因为只需进行 8 次点灭，所以与工作模式 1 有所不同。首先第一次点灯是变成 8’d1000_0001，实现了最高位和最低位灯亮，即最左边和最右边灯亮。再然后是变成 8’d1100_0011，即次最左灯和次最右灯亮。以此类推，到中间两灯亮，即输出变为 8’d1111_1111 时，全部灯都被点亮。完成点灯后是灭灯操作，首先是 Y_out[7]和 Y_out[0]转变成 0，即最左边和最右边灭。也可以以此类推，到最中间两灯灭，输出变为 8’d0000_0000，即全部灯灭。在完成一次工作模式 2 后，选择信号跳变到 3’b101(即暂存态 Model_temp2)后经过一个时钟周期，再变成工作模式 2，再进行一次工作模式 2 的内容。

工作模式 3 工作结果分析：

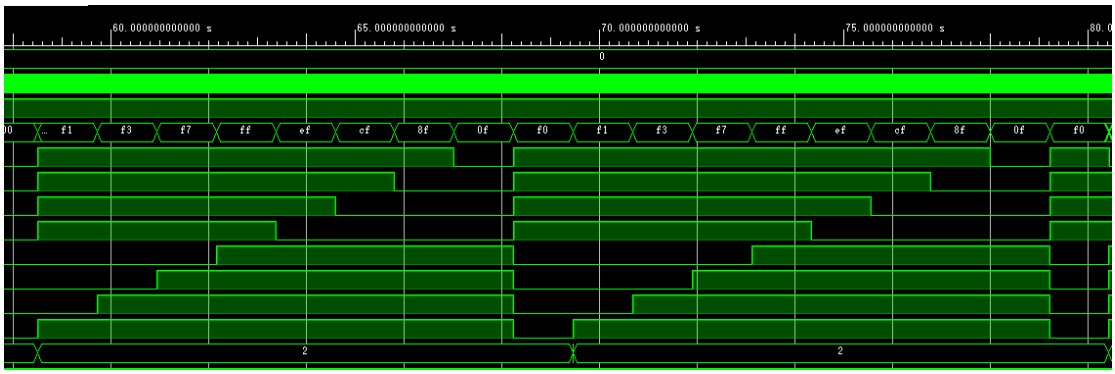

图 9 工作状态 3 波形图

工作模式 1 最初输出为 8’d1111_0000，即左边四个灯全亮，右侧四个灯全灭，然后是变成 8’d1111_0001，实现了最低位灯亮，即最右边灯亮。再然后是变成 8’d1111_0011，即次最右灯也点亮。以此类推，到中心靠右侧灯亮，即输出变为 8’d1111_1111 时，全部灯都被点亮。完成点灯后是灭灯操作，首先是 Y_out[4]转变成 0，即 8’d1110_1111，也就是中心靠左侧灯灭。也可以以此类推，到最左侧灯灭，输出变为 8’d0000_1111，即左边四个灯全灭，右侧四个灯全亮，然后瞬间变回 8’d1111_0000，即左边四个灯全亮，右侧四个灯全灭。在完成一次工作模式 3 后，选择信号跳变到 3’b110(即暂存态 Model_temp3)后经过一个时钟周期，再变成工作模式 3，再进行一次工作模式 3 的内容。

工作模式 4 工作结果分析：

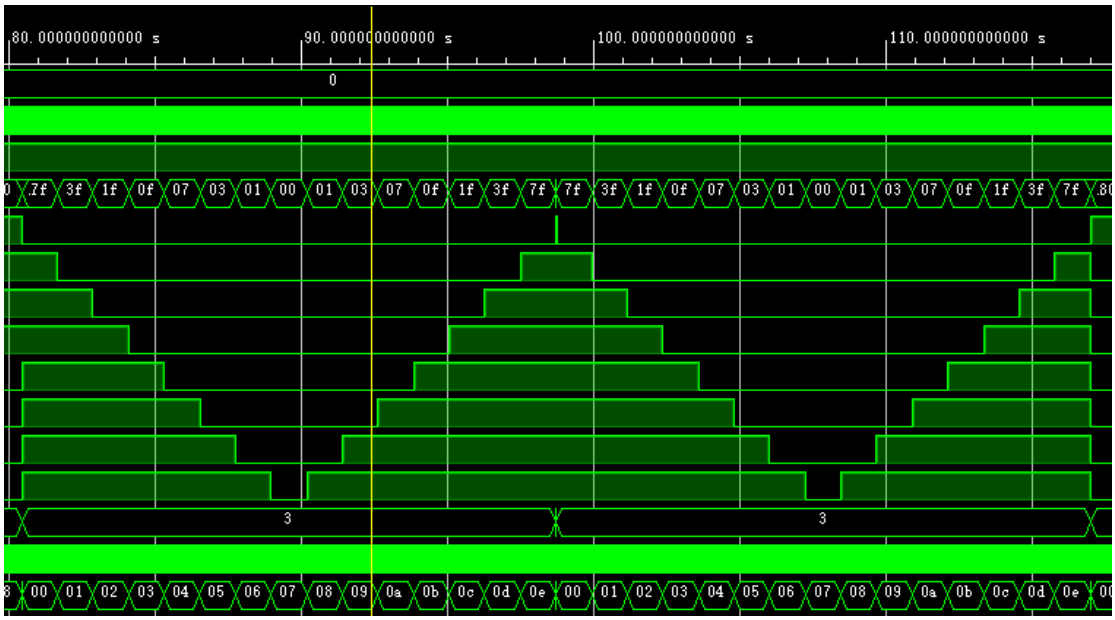

图 10 工作状态 4 波形图

工作模式 4 最初输出为 8’d1111_1111，然后是变成 8’d0111_1111，实现了最高位灯灭，即最左边灯灭。再然后是变成 8’d0011_1111，即次最左灯灭。以此类推，到最右侧灯灭，即输出变为 8’d0000_0000 时，全部灯都被熄灭。完成灭灯后是点灯操作，首先是 Y_out[7]转变成 1，即最左侧灯亮。也可以以此类推，到最右侧灯亮，输出变为 8’d1111_1111，即全部灯亮。在完成一次工作模式 4 后，选择信号跳变到 3’b111（即暂存态 Model_temp4）后经过一个时钟周期，在变成工作模式 4，再进行一次工作模式 4 的内容。

课程设计题目二：基于 Verilog 的十字路口交通灯控制电路设计

设计目的：

设计并实现一个简单的十字路口交通灯控制电路。

设计内容：

程序设计要求：

以 4 个红色指示灯、4 个绿色指示灯和 4 个黄色指示灯模拟路口东西南北 4 个方向的红绿黄交通灯。控制这些灯，使它们安下列规律亮灭。

工作模式以及内容如下：

1、东西方向绿灯亮，南北方向红灯亮。东西方向通车，时间 30 秒；

2、东西方向黄灯闪烁，南北方向红灯亮，时间 2 秒。

3、东西方向红灯亮，南北方向绿灯亮。南北方向通车，时间 30 秒；

4、东西方向红灯亮，南北方向黄灯闪烁，时间 2 秒。

5、返回 1，继续运行。

设计源文件模块输入信号：

选择信号[1:0]S，时钟信号 clk，复位信号 rst

设计源文件模块输出信号：

跑马灯亮灭信号[7:0]Y

Verilog 源文件名：

Traffic_lights. v

Verilog Testbench 名：

Traffic_lights_top. v

仿真软件：

Vivado ModelSim

设计要求分析：

图 1 交通灯的物理模型图

由日常知识可以知道，交通灯是由三个 LED 模块组合而成的，我们可以这样认为每一个方向的交通灯都可以用一个三位寄存器表示，每一个颜色的灯都可以认为是寄存器的其中的一位的高电平表示。可以做出如下定义，红灯对应的是 3’b001, 黄灯对应的是 3’b010, 绿灯对应的是 3’b100, 不工作状态对应的是 Idle=3’b000。要限制四个方向灯的亮灭，我们需要另外定义对应四个方向的四个三位寄存器，分别是面北的[2:0]North_Lights，面南的[2:0]South_Lights, 面西的[2:0]West_Lights, 面东的[2:0]East_Lights，以及通过计数器分频控制状态持续时间。

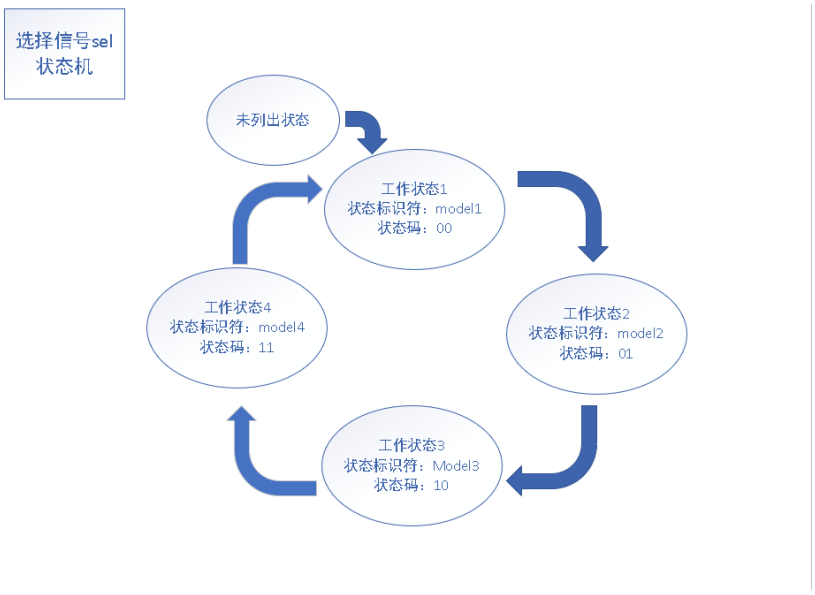

图 2 选择信号 sel 状态机状态转移示意图

要实现五个工作内容的转变，只需实现一个四个状态的状态机。即，首先在模块块内定义一个两位的寄存器[2:0]sel, 作用是接收输入的 S 和更变状态。如果输入的 S 为 00，则在运行完一次工作状态 1 的块程序后，跳转至工作模式 2，执行工作模式 2 后，跳转到至工作模式 3，执行工作模式 4 后，跳转至工作模式 4，执行工作模式 4 后跳转回工作模式 1。

设计内容：

工作状态 1 分析：

首先，进入工作状态 1，由不工作状态变成为面东的灯变绿（East_Lights<=green），面西的灯变绿（West_Lights<=green），面北的灯变红（North_Lights<=red），面南的灯变红（South_Lights<=red）。此过程持续 30s，完成后，跳转至工作状态 2，面东，面西灯进入不工作状态。

工作状态 2 分析：

进入工作状态 2，由工作状态 1 变成为面东的灯变黄且闪烁 East_Lights[1]<=East_Lights[1]^1 ；），面西的灯也变黄且闪烁即有（ West_Lights[1]<=West_Lights[1]^1; ），面北的灯一直为保持为红灯（North_Lights<=red），面南的灯也一直为保持为红灯（South_Lights<=red）。此过程持续 2s，完成后，跳转至工作状态 3。

工作状态 3 分析：

进入工作状态 3，由工作状态 2 变成为面东的灯变红（East_Lights<= red），面西的灯变红（West_Lights<=red），面北的灯变绿（North_Lights<= green），面南的灯变绿（South_Lights<= green）。此过程持续 30s，完成后，跳转至工作状态 4，面东，面西灯进入不工作状态。

工作状态 4 分析：

进入工作状态 4，由工作状态 3 变成为面东的灯一直保持为红灯

（East_Lights<= red），面西的灯也一直保持为红灯即有（West_Lights<=red;），面北的灯变黄且闪烁（North_Lights[1]<=North_Lights[1]^1），面南的灯也变黄且闪烁（South_Lights[1]<=South_Lights[1]^1）。此过程持续 2s，完成后，跳转回工作状态 1。

计数器分析：

在主 always 块外，再另建一个 always 作为计数器。同主 always 块一样 clk 上升沿触发，rst 下降沿触发。当 rst 高电平时，开始计数，计数 Count_time 时间后给 state_exist 加 1 再将 Count_time 复位为 0。当主触发块不再需要 state_exist 再增加时，将 count_rst 置 1，即使计数器复位。

RTL 图：

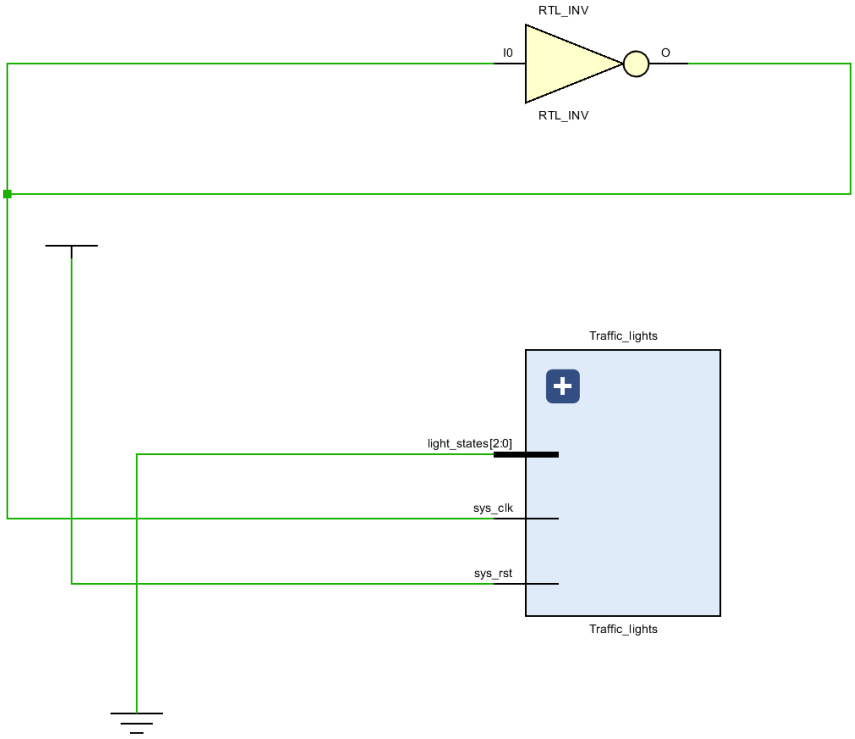

图 3 顶层模块 RTL 图

可以看到顶层模块的 RTL 图显示，Traffic_lights 一共有三个端口，即 sys_clk，light_states[2:0]，sys_rst。时钟信号是由外界激励不断反向产生的结果。

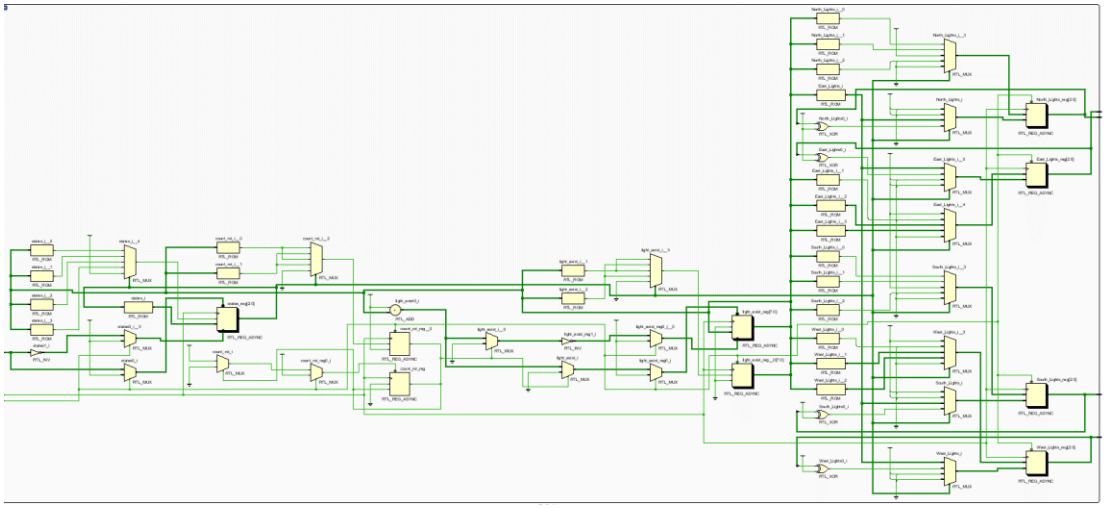

图 4 Running_Light 模块 RTL 图

可以看到 Traffic_lights 模块的 RTL 图，显示模块使用了较多的寄存器和多路选择器和触发器，可能存在较高的延迟，但相较于题目一器件使用的数量更少。

设计结果分析

仿真整体结果分析：

Vivado 仿真：

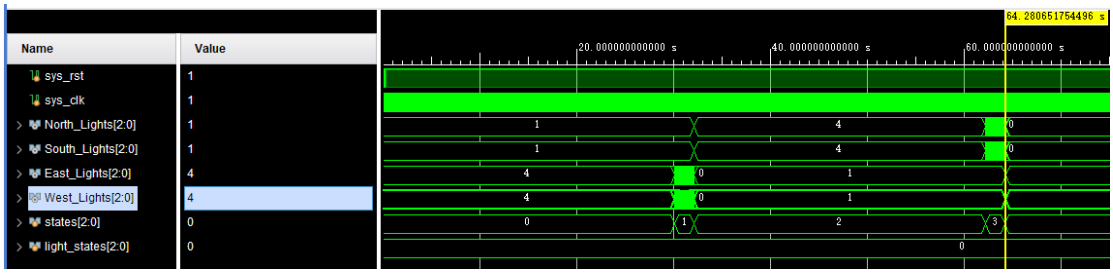

图 5 Vivado 仿真 四个工作模式波形图

可以看到 Vivado 仿真出的四个工作模式的波形图，不难看到完成四个工作模式所需时间约为 64 s，符合题目要求的（30s+2s+30s+2s=64s）红绿灯灯工作时间。

Modelsim 仿真：

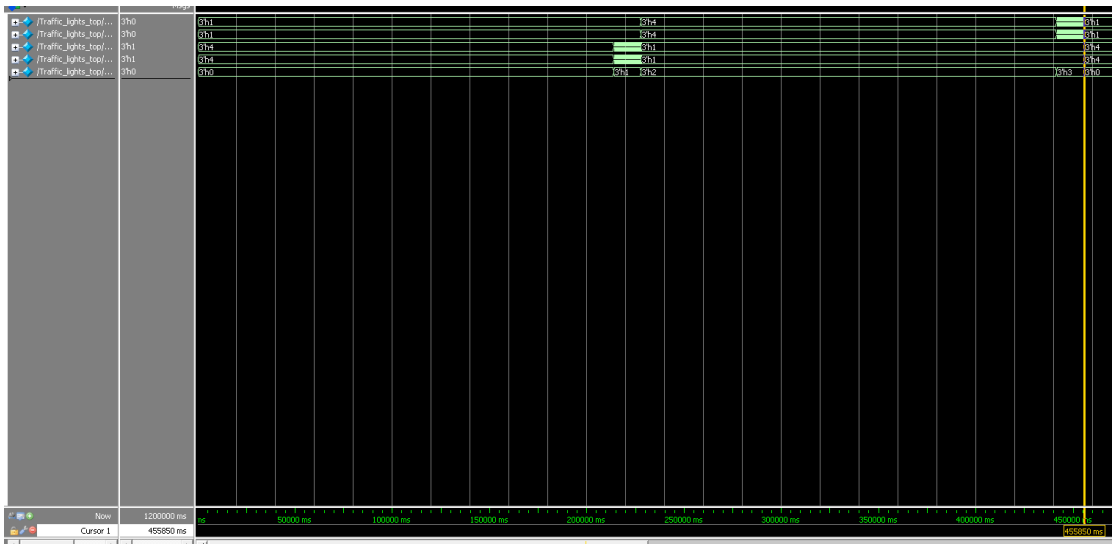

图 6 ModelSim 仿真 四个工作状态波形图

可以看到 ModelSim 仿真出的四个工作模式的波形图，不难看到完成四个工作模式所需时间约为 45. 5 s。相较于 Vivado 仿真的 64 s，推测也应该是 Vivado 仿真使用的是虚拟 FPGA 资源，ModelSim 直接使用计算机资源。

工作模式分析

各工作模式形图从上到下分别为复位信号 sys_rst，时钟信号 sys_clk，面北面灯[2:0]North_lights, 面南面灯[2:0]South_lights, 面东面灯[2:0]East_lights, 面西面灯[2:0]West_lights 和状态信号[2:0]states。

工作模式 1 工作结果分析：

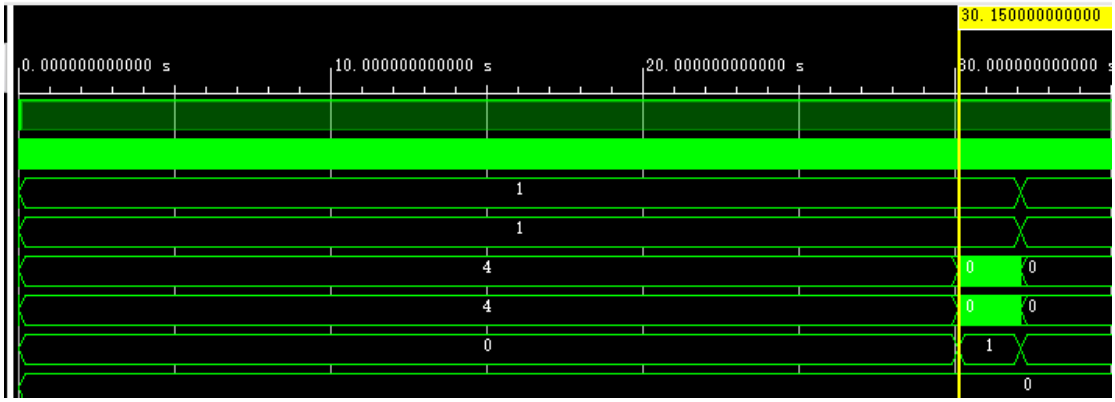

图 7 工作状态 1 波形图

在该工作状态，下面北面灯和面南面灯始终为 3’b001（即红灯），面东面灯和面西面灯始终为 3’b100（即绿灯），且从竖线可以看到工作状态 1 结束的时间为 30. 15s 符合题目要求的持续时间为 30s。

工作模式 2 工作结果分析：

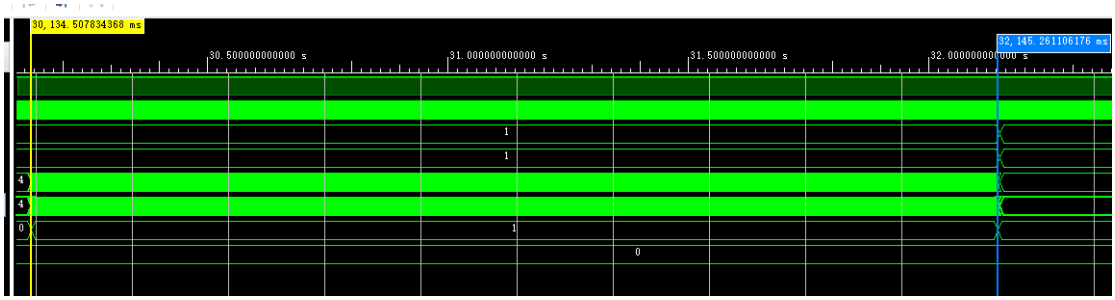

图 8 工作状态 2 波形图

在该工作状态下，面北面灯和面南面灯始终保持为 3’b001（即红灯），面东面灯和面西面灯突变为 3’b010（即黄灯）且开始不断闪烁，且从竖线可以看到工作状态 2 结束的时间约为 2. 011s 符合题目要求的持续时间为 2s。

工作模式 3 工作结果分析：

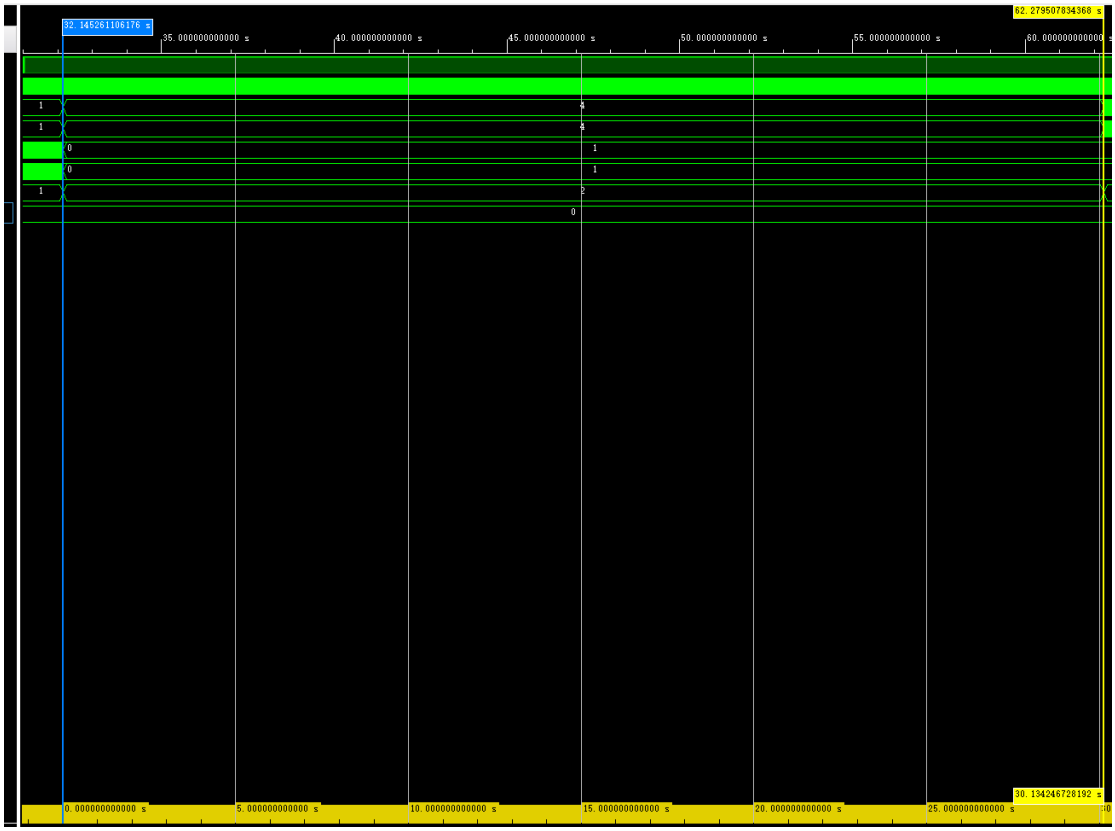

图 9 工作状态 3 波形图

在该工作状态下，面北面灯和面南面灯由 3’b001（即红灯）变为 3’b100（即绿灯），面东面灯和面西面灯由闪烁的黄灯变为 3’b001（即红灯），且从竖线可以看到工作状态 1 结束的时间为 30. 134s 符合题目要求的持续时间为 30s。

工作模式 4 工作结果分析：

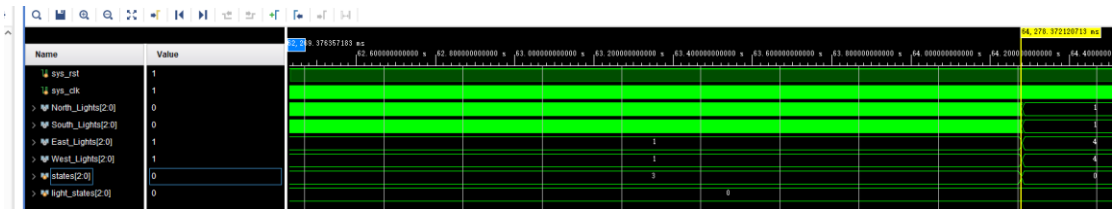

图 10 工作状态 4 波形图

在该工作状态下，面北面灯和面南面灯突变为 3’b010（即黄灯）且开始不断闪烁，面东面灯和面西面灯始终保持为 3’b001（即红灯），且从竖线可以看

到工作状态 2 结束的时间约为 2. 008s 符合题目要求的持续时间为 2s。

设计心得体会：

本次课程设计，使我对 Verilog 这门语言有了更加深入的了解。在本次设计中，我遇到的最大问题有两个，一个是题目一的重复操作和两个题目的分频操作。问题一的重复操作主要是要考虑到非阻塞赋值的特点，即等号右边的值都先计算，然后再到下一个时钟的上升沿处，再统一赋值给等号左边。我的问题是在状态转换时，会出现一些参数还没复位但状态已经转过去，然后直接略过转过去的状态跳到下下个状态，解决方法主要是再另写一个去处理复位而不去对 LED 操作的状态。问题二，主要还是在题目一怎么分频上卡住了，一开始的想法是写自建任务，但是无论怎么样都无法分频第二次，发现是要写 automatic（动态）去修饰多次调用的任务，来给数据划分多个空间，再然后发现分频的计数参数只会变化一次，参考 C 语言写了个嵌套，但一开始仿真就会卡在 13ns，把电路综合出来发现是出现了多个锁存器。于是我放弃了 task 的写法，把任务删了，然后写第二个 always 语句用来分频。在仿真阶段，我发现写 always 语句要比 task 更好处理本题的变化，于是题目一和题目二都改用为 always。究其原因，还是把 HDL 语言当作 C 语言来写，写 Verilog 程序应该是带着会综合成什么硬件的视角来看待这个问题。

参考文献

［1］夏宇闻. Verilog 数字系统设计教程. 北京：北京航空航天大学出版社. 2017

［2］刘福奇. Verilog HDL 设计与实战. 北京：北京航空航天大学出版社. 2012