计算机系统结构实验报告　实验 1

FPGA 基础实验: LED Flow Water Light

方泓杰 518030910150

2020 年 6 月 5 日

摘要

本实验实现了 FPGA 基础实验中的 LED 流水灯器件。该器件在每个周期的时钟上升沿时计数器加 1 ，当计数器达到最大值时，LED 灯左移一位点亮；该器件还支持接收 `reset` 信号对 LED 灯进行初始化与复位。本实验通过软件仿真的形式进行实验结果的验证。

目录

1 实验目的

本次实验有如下四个实验目的：

1. 熟悉 Xilinx 逻辑设计工具 Vivado 的基本操作；
2. 掌握使用 VerilogHDL 进行简单的逻辑设计；
3. 理解 LED 流水灯的工作原理；
4. 使用功能仿真验证功能实现的正确性。

2 原理分析

本次实验需要实现 LED 流水灯这一个简单的 FPGA 部件，其功能要求是每间隔一段时间点亮下一个 LED 灯并且熄灭当前的 LED 灯。我们可以利用一个计数器 `cnt_reg` 实现时钟周期数目的记录，当计数器达到我们设定的最大值时，我们进行 LED 灯的切换，同时将计数器重置为 0。由于我们使用 8 位 LED 灯，我们采取 8 位二进制编码 `light_reg` 来表示之；第 i 位为 0 说明第 i 个 LED 灯未被点亮；第 i 位为 1 说明第 i 个 LED 灯被点亮。于是我们可以使用左移操作进行 LED 灯的切换；需要注意的是，如果当前点亮的是最后一个 LED 灯，即 8 位二进制编码仅最高位为 1，我们需要特殊处理，将下一个状态的 8 位二进制编码设为仅最低一位为 1，从而实现 LED 灯的循环点亮。此外，我们还需要设计针对 `reset` 信号进行重置这一特殊机制。

3 功能实现

根据第2节中所阐释的原理，我们使用计数器 `cnt_reg` 实现时钟周期数目的记录，并且使用 8 位二进制编码 `light_reg` 来表示这个 8 位 LED 灯。

我们采取如下代码实现对于计数器 `cnt_reg` 的更新，并在其中加入关于重置模块的设计：当 `reset` 信号为 1 时，我们将计数器清零重置。

```
always @ (posedge clock)
    begin
        if (reset)
            cnt_reg <= 0;
        else
            cnt_reg <= cnt_reg + 1;
    end
```

我们采取如下代码实现表示 LED 灯的 8 位二进制编码 `light_reg` 的更新，并在其中加入关于重置模块的设计：当 `reset` 信号为 1 时，我们将 LED 灯状态设为初始状态（仅点亮第一个 LED 灯）。

```
always @ (posedge clock)
    begin
        if (reset)
            light_reg <= 8'h01;
        else if (cnt_reg == 24'hffffff)
```

```
        begin
            if (light_reg == 8'h80)
                light_reg <= 8'h01;
            else
                light_reg <= light_reg << 1;
        end
    end
```

在实际的仿真验证过程中，我们发现上述设定的计数器最大值过大，使得计数器达到最大值的次数过少，LED 状态改变数量过少，导致我们通过观测到 LED 的改变来验证程序的正确性，因此我们将计数器的最大值进行了修改，如以下代码所示。

```
always @ (posedge clock)
    begin
        if (reset)
            light_reg <= 8'h01;
        else if (cnt_reg == 2'b11)
            begin
                if (light_reg == 8'h80)
                    light_reg <= 8'h01;
                else
                    light_reg <= light_reg << 1;
            end
    end
```

完整的代码实现参见附录 A。

4 结果验证

我们使用 Verilog 编写激励文件，采用软件仿真的形式对于 LED 流水灯进行测试（代码实现参见附录 B）。初始的测试结果如图 1 所示（**注意：这个图对应的不是最终结果，最终结果请看下页**）：

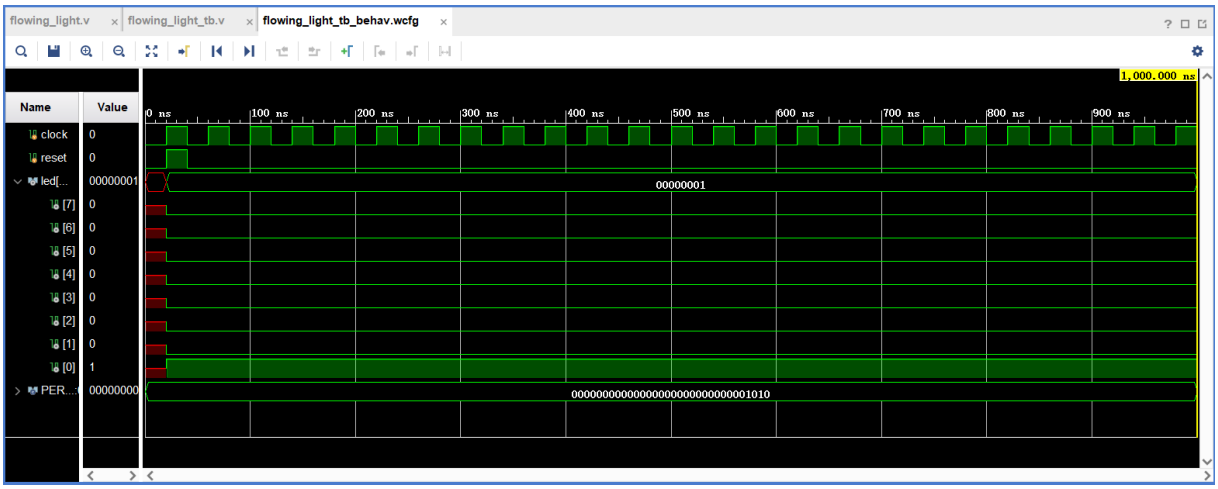

图 1: 初始测试结果

分析该结果出现的原因，我们发现，初始设定的计数器最大值过大，使得计数器达到最大值的次数过少，LED 状态改变数量过少，导致我们通过观测到 LED 的改变来验证程序的正确性。因此，我们对于计数器的最大值进行了一定的修改，便于我们在仿真的时候观测（这部分内容在第 4 节中有详细描述）。修改后的最终的测试结果如图 2 所示（**注意：这个图为最终结果**）：

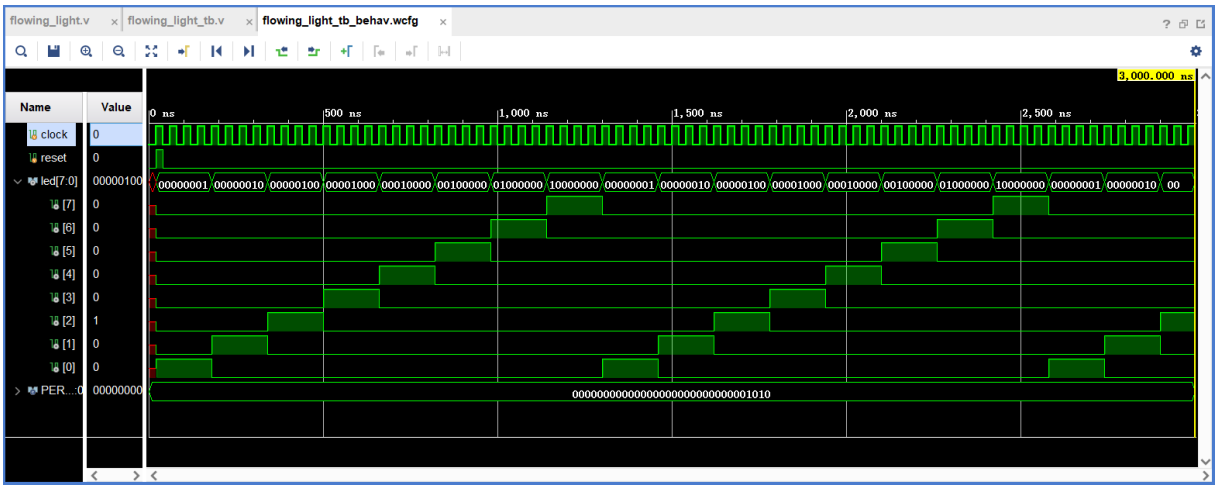

图 2: 最终测试结果

从图 2 中可以看出，我们完成了 LED 流水灯的功能实现，并且仿真结果正确。

5 总结与反思

本实验实现了 FPGA 实验中 LED 流水灯这一基础部件的设计与仿真。这一部件内部的逻辑较为简单，易于初学 Verilog 语言的编程者上手。通过这次试验，我对于 Xillinx 逻辑设计工具 Vivado 有了初步的认识，基本掌握了使用 Verilog 语言进行逻辑设计的方法。可以发现，Verilog 语言与 C 语言类似，我们可以通过类比来快速掌握 Verilog 语言的一些基本设计方法。同时，我还学习了仿真模拟验证的具体方法。这些都为后面几次实验的复杂逻辑设计与仿真奠定了基础。观察到实验中使用了 `<=` 的赋值方法，我上网搜索了相关资料，发现这是时序逻辑的设计方法，而普通的赋值 `=` 使用的是组合逻辑的设计方法，这个认识对于后面的大型试验有着非常重要的帮助。总之，在这次实验中我收获颇丰。

6 致谢

感谢本次实验中指导老师在课程微信群里为同学们答疑解惑；

感谢上海交通大学网络信息中心提供的远程桌面资源；

感谢计算机科学与工程系相关老师对于课程指导书的编写以及对于课程的设计，让我们可以更快更好地学习相关知识，掌握相关技能；

感谢电子信息与电气工程学院提供的优秀的课程资源。

附录 A　设计文件完整代码实现

```
1  `timescale 1ns / 1ps
2
3  module flowing_light(
4      input clock,
5      input reset,
6      output [7 : 0] led
7      );
8
9      reg [1 : 0] cnt_reg;
10     reg [7 : 0] light_reg;
11
12     always @ (posedge clock)
13         begin
14             if (reset)
15                 cnt_reg <= 0;
16             else
17                 cnt_reg <= cnt_reg + 1;
18         end
19
20     always @ (posedge clock)
21         begin
22             if (reset)
23                 light_reg <= 8'h01;
24             else if (cnt_reg == 2'b11)
25                 begin
26                     if (light_reg == 8'h80)
27                         light_reg <= 8'h01;
28                     else
29                         light_reg <= light_reg << 1;
30                 end
31         end
32
33     assign led = light_reg;
34 endmodule
```

附录 B　激励文件完整代码实现

```
1  `timescale 1ns / 1ps
2
3  module flowing_light_tb(
4      );
5      reg clock;
6      reg reset;
7      wire [7 : 0] led;
8
9      flowing_light u0(
10         .clock(clock),
11         .reset(reset),
12         .led(led));
13
14     parameter PERIOD = 10;
15
16     always #(PERIOD * 2) clock = !clock;
17
18     initial begin
19         clock = 1'b0;
20         reset = 1'b0;
21         #(PERIOD * 2) reset = 1'b1;
22         #(PERIOD * 2) reset = 1'b0;
23
24         // #580; reset = 1'b1
25      end
26 endmodule
```