中国科学技术大学计算机学院

《数字电路实验》报告

实验题目：简单时序逻辑电路

学生姓名：傅申

学生学号：PB20000051

完成日期：2021 年 11 月 7 日

计算机实验教学中心制

2020 年 09 月

【实验题目】

简单时序逻辑电路

【实验目的】

- 掌握时序逻辑相关器件的原理及底层结构
- 能够用基本逻辑门搭建各类时序逻辑器件
- 能够使用 Verilog HDL 设计简单时序逻辑电路

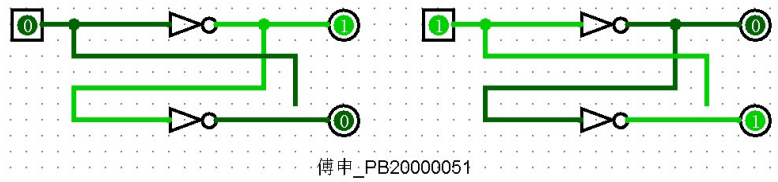
【实验环境】

- Windows PC 一台: CPU 为 Intel i5-1035G1
- Logisim 仿真工具
- Microsoft Visual Studio Code

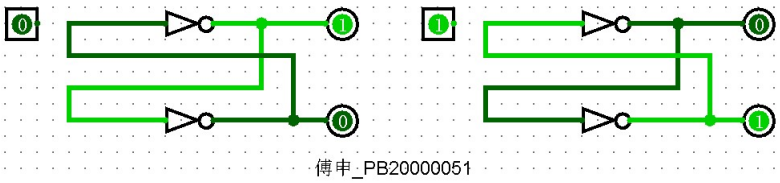
【实验过程】

Step 1: 搭建双稳态电路

首先搭建如下图 1(a) 的电路, 待输入信号将其状态初始到确定状态后再将右下的耦合线连上, 再断开输入引脚处的线, 如图 1(b).



(a) 第一步

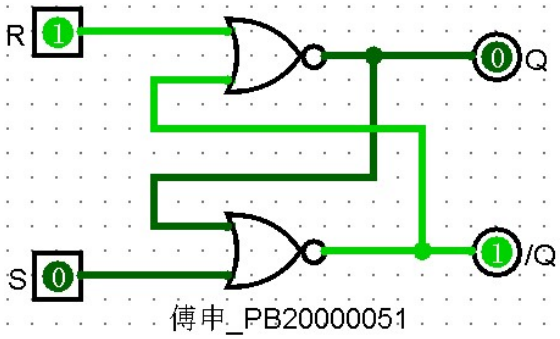


(b) 第二步

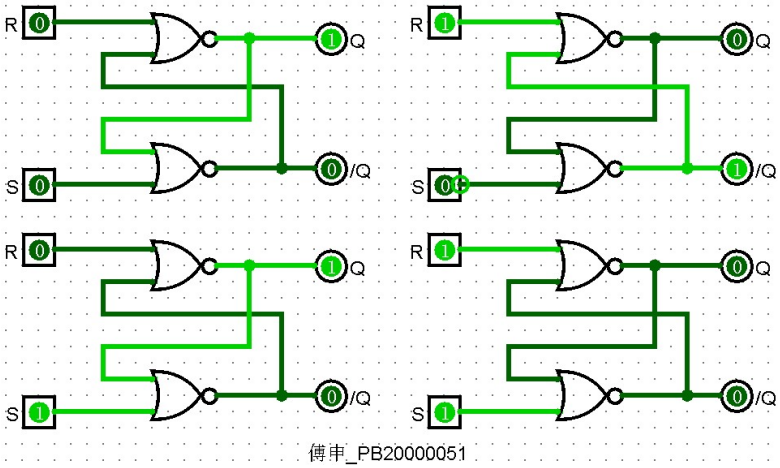
图 1: 双稳态电路

Step 2: 搭建 SR 锁存器

首先搭建如下图 2(a) 的电路, 这就是一个 SR 锁存器, 对于四种不同的输入状态, 它的输出状态也不同, 如图 2(b) 和表 1.



(a) SR 锁存器电路图



(b) SR 锁存器不同输出状态

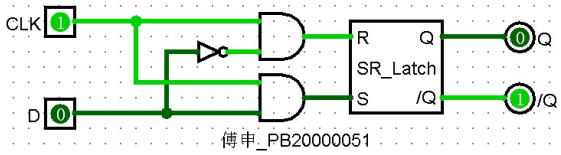
图 2: SR 锁存器

表 1: SR 锁存器真值表

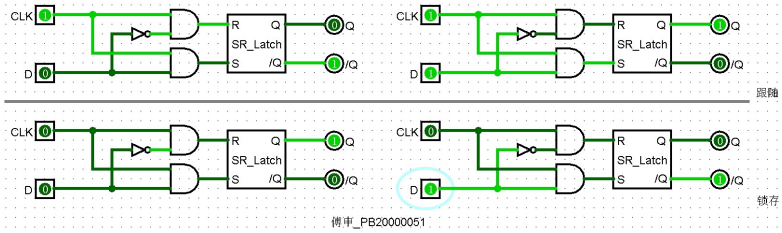
输入		输出		状态
S	R	Q	$\overline{Q}$	
0	0	不变	不变	锁存
0	1	0	1	置 0
1	0	1	0	置 1
1	1	0	0	未定义

Step 3: 搭建 D 锁存器

SR 锁存器两个输入都为 1 是一种未定义状态, 我们不希望这种状态出现, 为此我们在 SR 锁存器前面添加两个与门和一个非门, 如下图所示, 便构成了 D 锁存器, 如图 3(a). 分析 D 锁存器电路可以发现, 当 CLK 信号为高电平时, Q 信号将随着 D 端输入信号的变化而变化, 称之为 “跟随” 状态; 当 CLK 信号为低电平时, Q 信号将保持之前的值, 不会收到 D 信号变化的影响, 称之为 “锁存” 状态, 如图 3(b). D 锁存器是一种电平敏感的时序逻辑器件.



(a) D 锁存器电路图

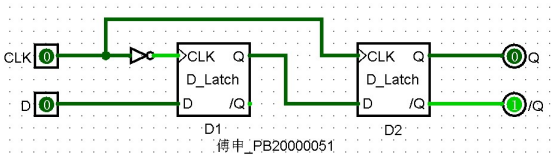


(b) D 锁存器不同输出状态

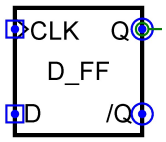
图 3: D 锁存器

Step 4: 搭建 D 触发器

将两个 D 锁存器串起来, 其控制信号有效值始终相反, 就构成了 D 触发器, 如下图 4(a) 所示, CLK 信号为低电平时, D 信号通过了 D1, 当 CLK 信号由低电平变为高电平时, D1 关闭, D2 打开, 信号到达 Q 端. 把 CLK 端口换成一个可自动变化的时钟信



(a) 电路图



(b) 封装

图 4: D 触发器

号. 在 Logisim 菜单栏中点击 “simulation” 选项, 将 “Tick Frequency” 设置为 “1Hz” , 然后使能仿真和触发功能, 在 “CLK” 信号以 1Hz 频率跳变过程中, 改变 D 信号的输入值, 观察 Q 信号的输出. 如下图 5.

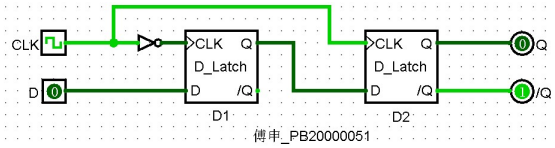


图 5: 观察 D 触发器输出

其 Verilog 代码为:

Verilog 代码 1: D 触发器

```
1 // D Flip-Flop
2 module d_ff (
3     input clk, d,
4     output reg q
```

```
);
    always@(posedge clk)  q <= d;
endmodule
```

我们还可以为触发器添加复位信号, 如下图 6 所示, 可以看出, 当复位信号有效 (低电平有效) 时, 输出信号 Q 始终为零. 复位信号只有在时钟信号的上升沿才起作用, 在非上升沿时刻, 复位信号不起作用. 这种复位方式称为同步复位. 其 Verilog 代码如下代码 2.

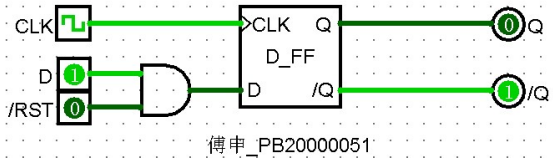


图 6: 同步复位 D 触发器

Verilog 代码 2: 同步复位 D 触发器

```
// D Flip-Flop with a synchronous reset
module d_ff_sr (
    input  clk, rst_n, d,
    output reg q
);
    always @(posedge clk) begin
        if (rst_n == 0) q <= 1'b0;
        else q <= d;
    end
endmodule
```

同样地, 我们可以写出异步复位 D 触发器的 Verilog 代码 3.

Verilog 代码 3: 异步复位 D 触发器

```
// D Flip-Flop with a synchronous reset
module d_ff_ar (
    input  clk, rst_n, d,
    output reg q
);
    always @(posedge clk or negedge rst_n) begin
        if (rst_n == 0) q <= 1'b0;
        else q <= d;
    end
endmodule
```

Step 5: 搭建寄存器

寄存器的本质就是 D 触发器, 如下图 7所示, 我们可以用 4 个 D 触发器构成一个能够存储 4bit 数据的寄存器, 同时带有低电平有效的复位信号. 在时钟上升沿, 如果复位信号无效, 寄存器的输入信号就会被存储在寄存器中. 它的 Verilog 代码如代码 4.

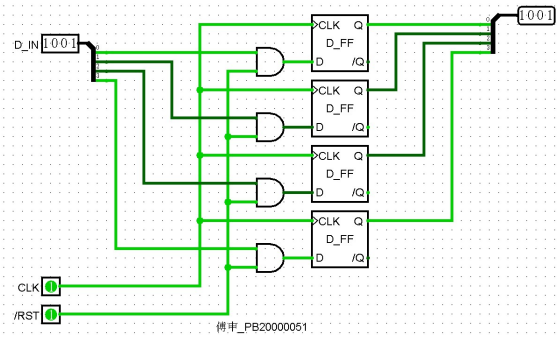


图 7: 4bit 寄存器

Verilog 代码 4: 4bit 寄存器

```
1  // 4-bit register
2  module reg_4bit (
3      input  clk, rst_n,
4      input  [3:0] D_in,
5      output reg [3:0] D_out
6  );
7      always @(posedge clk) begin
8          if (rst_n == 0) D_out <= 4'b0; // D_out <= 4'b0011;
9          else D_out <= D_in;
10     end
11 endmodule
```

Step 6: 搭建简单时序逻辑电路

利用 4bit 寄存器, 搭建一个 4bit 的计数器, 该计数器在 0 ∼ 15 之间循环计数, 复位时输出值为 0, 电路图如下图 8 所示.

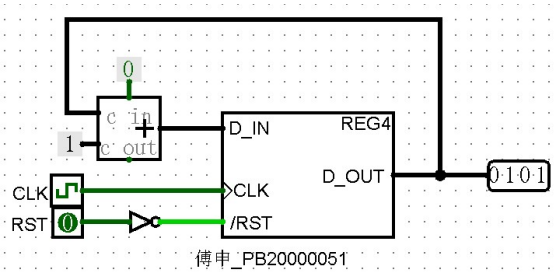


图 8: 4bit 计数器

其 Verilog 代码如下:

Verilog 代码 5: 4bit 计数器

```
// 4-bit counter MOD16
module counter_4bit (
    input  clk, rst_n,
    output reg [3:0] cnt
);
    always @(posedge clk ) begin
        if (rst_n == 0) cnt <= 4'b0;
        else cnt <= cnt + 4'b1;
    end
endmodule
```

【实验练习】

题目 1: 用与非门搭建的 SR 锁存器如下图 9 所示, 电路在不同输入时的状态如下图 10 和表 2 所示.

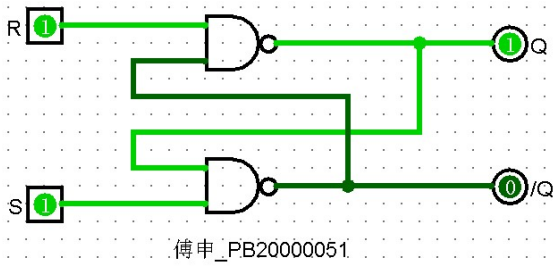


图 9: SR 锁存器

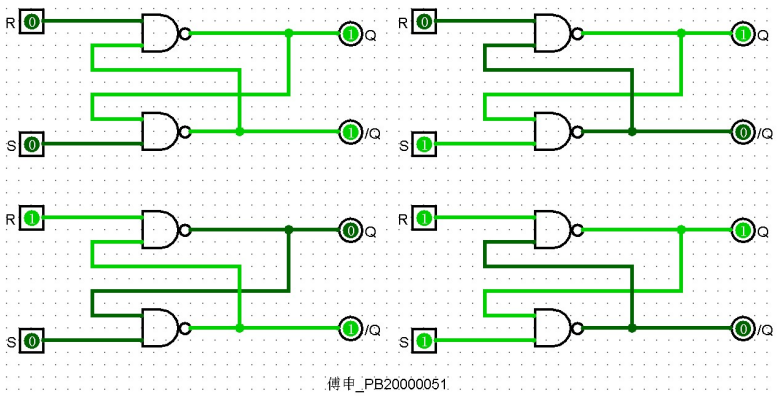
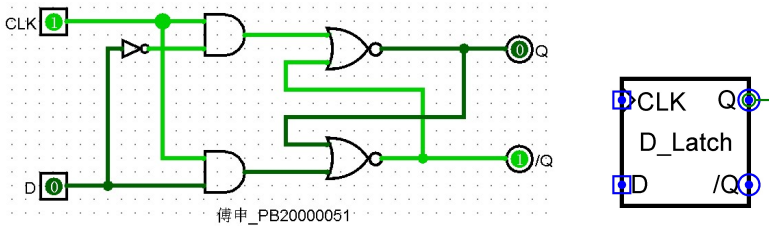


图 10: SR 锁存器状态

表 2: SR 锁存器真值表

输入		输出		状态
S	R	Q	$\overline{Q}$	
0	0	1	1	未定义
0	1	0	1	置 0
1	0	1	0	置 1
1	1	不变	不变	锁存

题目 2: 首先搭建 D 锁存器如下图 11. 仿照同步复位 D 触发器的搭建方法, 将置位信号取



(a) 电路图　　(b) 封装

图 11: D 锁存器

反后与 D 信号求或作为 D 锁存器的输入, 这样, 当置位信号为低电平时, D 锁存器的输出为 D 信号, 当置位信号为高电平时, D 锁存器的输入就是 1, 在时钟上升沿, 触发器被置位 1. 如下图 12 所示.

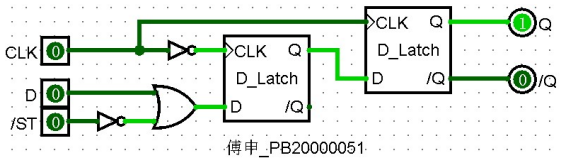


图 12: 同步置位 D 触发器

其 Verilog 代码如下:

Verilog 代码 6: 同步置位 D 触发器

```
1  // D Flip-Flop with a synchronous set
2  module d_ff_ss (
3      input  clk, st_n, d,
4      output reg q
5  );
6      always @(posedge clk) begin
7          if (st_n == 0) q <= 1'b1;
8          else q <= d;
9      end
10 endmodule
```

题目 3: 首先如图 2(a) 搭建 SR 锁存器. 然后搭建基本框架如下图 13 所示.

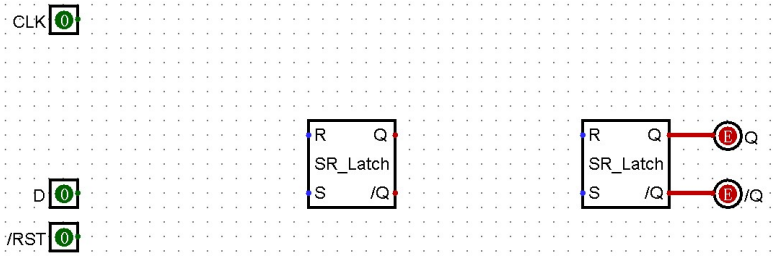


图 13: 基本框架

要求异步复位, 则不论时钟信号与 D 信号的状态如何, SR 锁存器的输出都是 0. 因此, 复位信号为低电平时, 两个 SR 锁存器的 R 输入端应该输入 1, 为了防止 SR 锁存器的两个输入端均为 1, 此时 S 输入端应该输入 0. 所以, 在两个 SR 锁存器前, 复位信号取反后接一个或门接入 R 输入端, 复位信号同时接一个与门然后接入 S 输入端. 如下图 14所示.

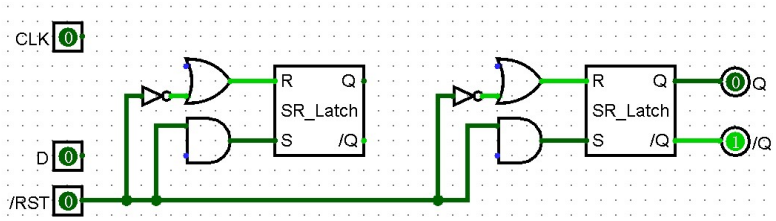
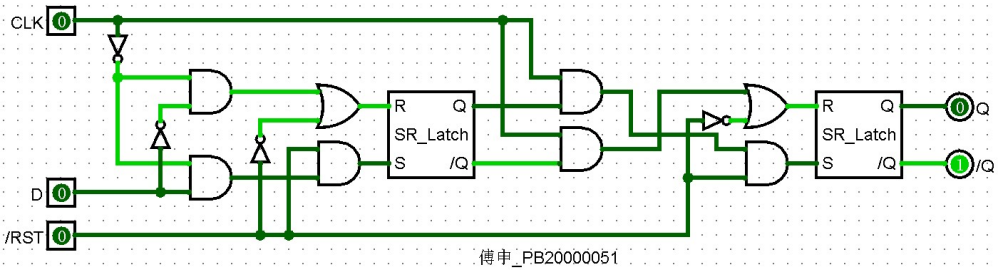
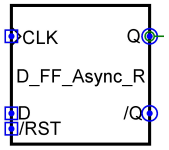


图 14: 处理复位信号

这时, 可以将四个门剩下的输入端当作对应 SR 锁存器的输入, 按照 D 触发器的搭建方法, 相应地连线, 可以得到异步复位 D 触发器如下图 15 所示.



(a) 电路图



(b) 封装

图 15: 异步复位 D 触发器

进一步调用该触发器, 使用 Logisim 中的加法器模块, 设计出一个从 0~15 循环计数的

4bit 计数器, 如下图 16 所示. 其 Verilog 代码如下:

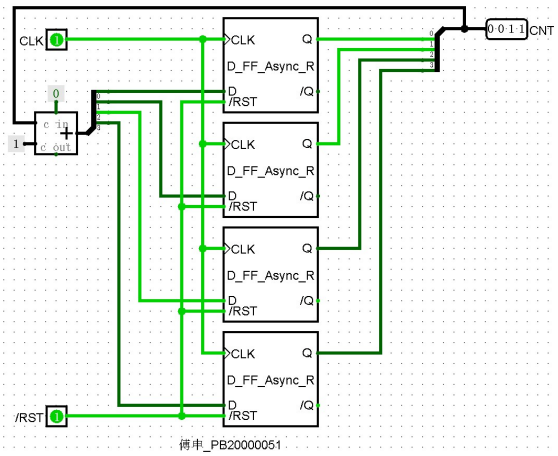
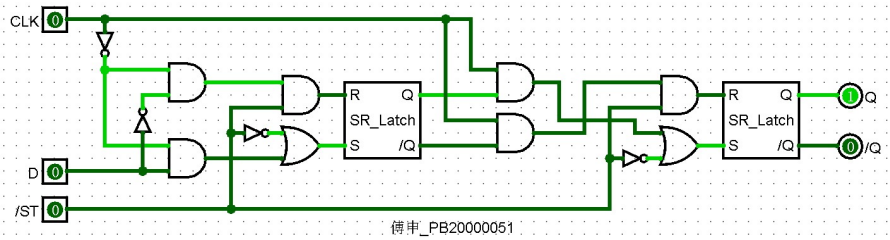


图 16: 0∼15 循环计数 4bit 计数器

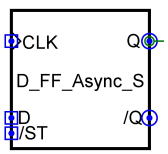
Verilog 代码 7: 0∼15 循环计数 4bit 计数器

```
1  // 4-bit up counter MOD 16
2  module up_counter_mod16 (
3      input clk, rst_n,
4      output reg [3:0] cnt
5  );
6      always @(posedge clk or negedge rst_n) begin
7          if (rst_n == 0) cnt <= 4'b0;
8          else cnt <= cnt + 4'b1;
9      end
10 endmodule
```

题目 4: 仿照图 15 中的异步复位 D 触发器, 构造一个异步置位 D 触发器如下图 17 所示.



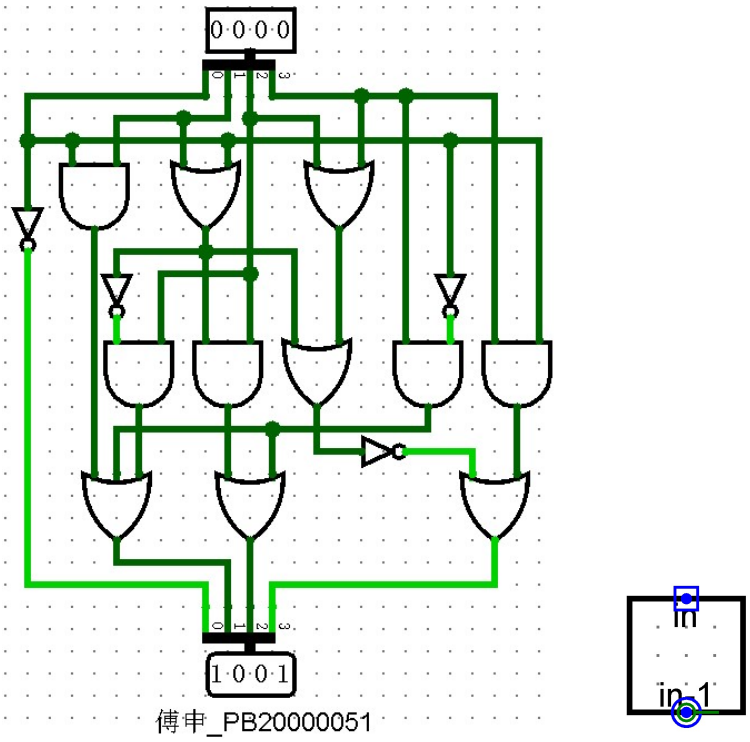
(a) 电路图



(b) 封装

图 17: 异步置位 D 触发器

构造一个 4bit 9～ 0 自减器如下图 18(a), 其真值表如表 3 所示.



(a) 电路图　　(b) 封装

图 18: 4bit 9～ 0 自减器

表 3: 自减器真值表

输入	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
输出	1001	0000	0001	0010	0011	0100	0101	0110	0111	1000
输入	1010	1011	1100	1101	1110	1111				
输出	0111	1010	0111	1100	0111	1110				

最后设计出的 4bit 9～ 0 自减器电路图如下图 19 所示, 其 Verilog 代码如下代码 8 所示.

Verilog 代码 8: 4bit 9～ 0 自减器

```
1 module down_counter_mod10(
2     input clk, rst_n,
3     output reg [3:0] cnt
4 );
5     always @(posedge clk or negedge rst_n) begin
6         if (rst_n == 0) cnt <= 1'b0;
7         else begin
8             if (cnt == 4'b0) cnt <= 4'b1001;
9             else cnt <= cnt - 1;
```

```
        end
    end
endmodule
```

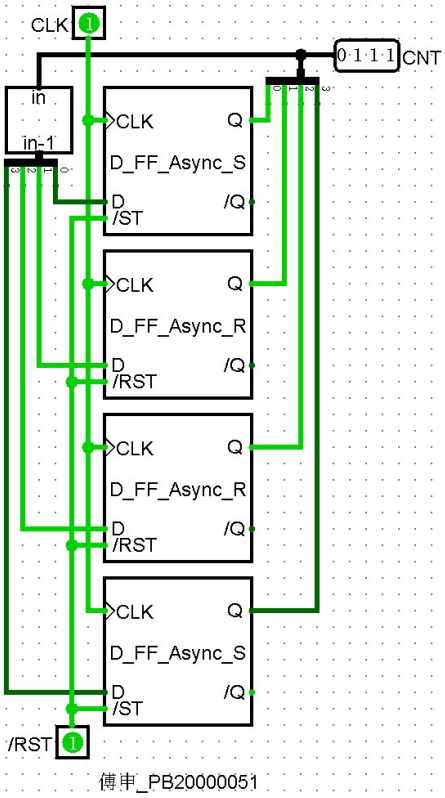


图 19: 4bit 9∼ 0 自减器

题目 5: 要使复位信号高电平有效, 最简单的方法是对各个示例电路的 /RST 输入取反, 或者在内部门电路进行调整. 对应的 Verilog 代码结构调整如下:

Verilog 代码 9: 对应的 Verilog 代码结构调整

```
module example(
    input clk, rst,
    output reg out
);
    always @(posedge clk or posedge rst) begin
        if (rst == 1) out <= 1'b0;
        else out <= out; // 这里可以是任意的操作
    end
endmodule
```

【总结与思考】

收获 了解了如何在 Logisim 中搭建简单的时序逻辑电路, 并且编写相应的 Verilog 代码. 学习了在 Logisim 模拟中出现故障如何排查 Bug 并重启模拟.

难易程度与任务量 中等.

建议 暂无.