线上自学复习实验 02_简单时序逻辑

简介

通过系列实验，希望能够帮助数字电路基础较为薄弱的同学快速掌握完成计算机组成原理实验所需的逻辑电路设计基础。

本次实验中，我们将学习简单时序逻辑电路的设计方法。

实验目的

了解在线实验平台结构

学习使用远程在线实验平台

掌握简单时序逻辑的设计方法

实验环境

PC 一台（网络流畅）

远程桌面环境（vlab）

FPGA 远程实验平台（FPGAOL）

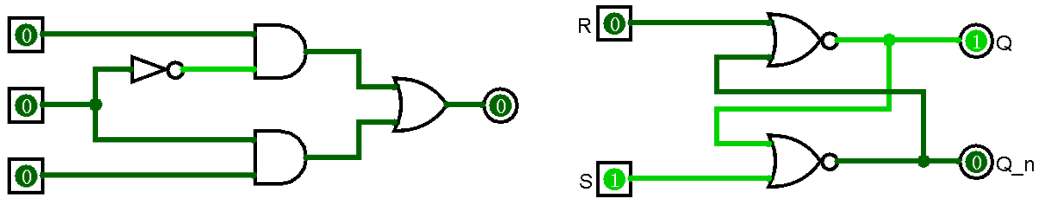
实验步骤

Step1： SR 锁存器——组合逻辑到时序逻辑的过渡

时序逻辑和组合逻辑电路的本质区别在于电路是否具有“记忆”功能，即当前时刻的电路状态（尤其是输出结果）是否与之前的状态有关，如当前状态与之前的状态完全无关，则为组合逻辑，否则为时序逻辑电路。

在电路结构上进行对比可以发现，组合逻辑电路中的数据流向是单方向的，信号从电路模块的输入端口进入，从输出端口出去，这个过程中不会重复经过任意部件，或者说组合逻辑电路中没有反馈机制。

而时序逻辑电路则恰恰相反，其“记忆”特性正是引入反馈机制的结果，如下图所示

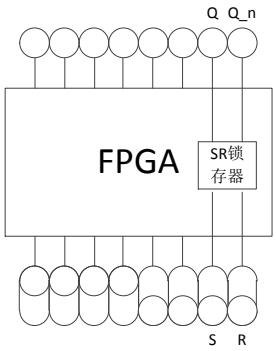


a.组合逻辑电路　　b.时序逻辑电路

SR 锁存器是构成时序电路的基本器件，下面我们尝试在 FPGA 上实现一个 SR 锁存器，其 Verilog 代码如下所示：

```
module step1_sr(
input   S,R,
output  Q,Q_n);
assign  Q   = ~(R | Q_n);
assign  Q_n = ~(S | Q);
endmodule
```

我们将最右侧的两个开关作为 S、R 输入，最右侧的两个 LED 作为 Q、Q_n 输出，如下图所示：



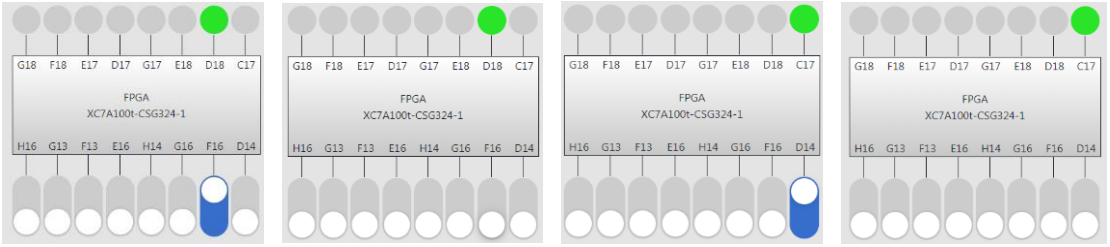
其管脚约束文件为：

```
#Switch
set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 } [get_ports { S }];
set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 } [get_ports { R }];
#Led
set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 } [get_ports { Q }];
set_property -dict { PACKAGE_PIN C17   IOSTANDARD LVCMOS33 } [get_ports { Q_n }];
set_property CFGBVS VCCO [current_design];
```

```
set_property CONFIG_VOLTAGE 3.3 [current_design];
set_property SEVERITY {Warning} [get_drc_checks LUTLP-1];
```

说明：因为 FPGA 内部不推荐出现锁存器，因此需要在约束文件中加入最后三条命令，用户可尝试将这三条命令删除，观察 Vivado 编译过程中的提示信息，实际上这三条约束都是根据错误提示加入的。

将生成的 bit 文件烧写到 FPGAOL 在线实验平台，其运行结果如下图所示：

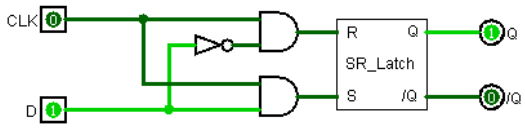


（S=1, R=0）　（S=0, R=0）　（S=0, R=1）　（S=0, R=0）

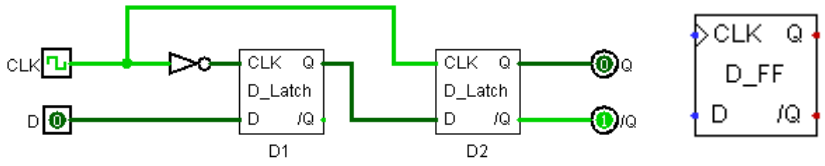
可以发现，该电路运行结果与 SR 锁存器行为特性一致，尤其是 S=0、R=0 时，其输出结果与前一状态有关。

Step2： D 触发器——同步时序逻辑电路的核心

以 SR 锁存器为核心，加上一个非门和两个与门，便构成了 D 锁存器，如下图所示：

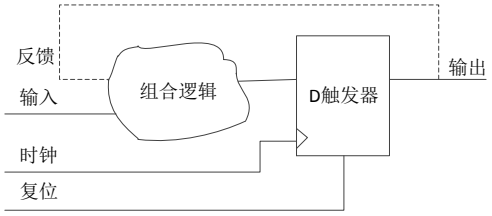


将两个 D 锁存器如下图所示的方式进行串联（需外加一个非门），便构成了 D 触发器，其符号如下方右图所示：

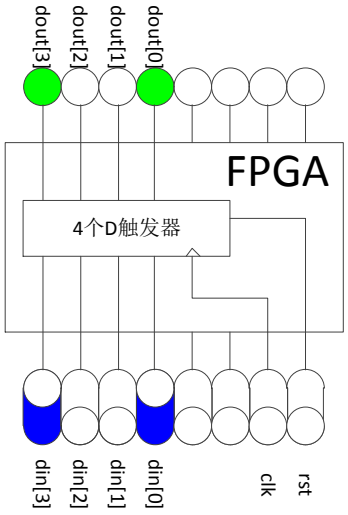


D 触发器是同步时序逻辑电路最为核心的基础器件，可以说同步

时序逻辑电路是由组合逻辑电路加上 D 触发器构成的，其结构一般如下图所示：



现在，我们设计一个简单的同步时序逻辑电路，一个 4bit 的输入数据(用 4 个开关输入)，经 D 触发器缓存后输出(用 4 个 LED 表示)，复位和时钟信号也由开关模拟，如下图所示：



其 Verilog 代码为：

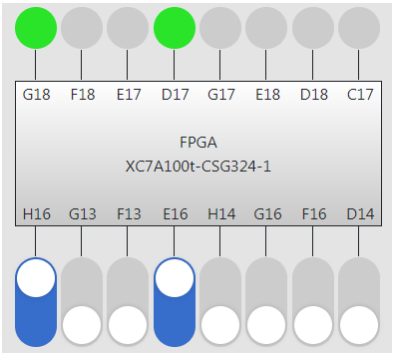
```
module step2_dff(
input                clk,rst,
input        [3:0]   din,
output  reg  [3:0]   dout);
always@(posedge clk or posedge rst)
begin
    if(rst)
        dout <= 4'b0;
    else
        dout <= din;
end
endmodule
```

其管脚约束文件为：

```
#Switch
set_property -dict { PACKAGE_PIN H16    IOSTANDARD LVCMOS33 } [get_ports { din[3] }];
set_property -dict { PACKAGE_PIN G13    IOSTANDARD LVCMOS33 } [get_ports { din[2] }];
set_property -dict { PACKAGE_PIN F13    IOSTANDARD LVCMOS33 } [get_ports { din[1] }];
set_property -dict { PACKAGE_PIN E16    IOSTANDARD LVCMOS33 } [get_ports { din[0] }];
set_property -dict { PACKAGE_PIN F16    IOSTANDARD LVCMOS33 } [get_ports { clk }];
set_property -dict { PACKAGE_PIN D14    IOSTANDARD LVCMOS33 } [get_ports { rst }];
#Led
set_property -dict { PACKAGE_PIN G18    IOSTANDARD LVCMOS33 } [get_ports { dout[3] }];
set_property -dict { PACKAGE_PIN F18    IOSTANDARD LVCMOS33 } [get_ports { dout[2] }];
set_property -dict { PACKAGE_PIN E17    IOSTANDARD LVCMOS33 } [get_ports { dout[1] }];
set_property -dict { PACKAGE_PIN D17    IOSTANDARD LVCMOS33 } [get_ports { dout[0] }];
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets {clk_IBUF}];
```

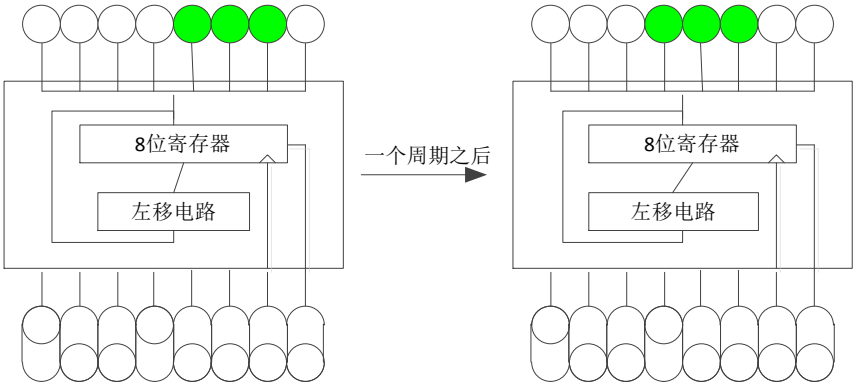
说明：在同步时序逻辑电路的设计中，时钟是一种非常特殊的信号，因此一般会通过专用的管脚输入，或者由 FPGA 芯片内部的专门电路生成，因此当使用普通管脚或普通信号当做时钟输入时，Vivado 会弹出警告甚至报错，因此我们在约束文件最后加入约束命令，使其能够完成综合。虽然这样做最后能生成正确的电路文件，但并不是推荐的做法，此例中的做法只是单纯为了演示 D 触发器的行为特性。

下图为实际运行结果，与预期结果一致：



Step3： 跑马灯

此设计中，我们仍然采用拨动开关来模拟时钟信号，来实现一个跑马灯电路的设计，8 个 LED 灯中有 3 个点亮，5 个暗，每个时钟周期亮暗状态左移一位，如下图所示：



其 Verilog 代码为：

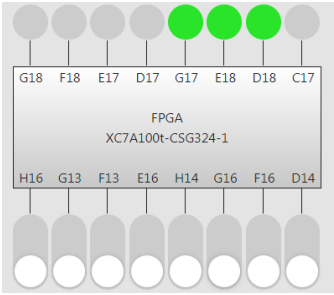
```
module step3_loop(
input                clk,rst,
output reg  [7:0]    led);
always@(posedge clk or posedge rst)
begin
    if(rst)
        led <= 8'b0000_0111;
    else
        led <= {led[6:0],led[7]};
end
endmodule
```

其管脚约束文件为：

```
#Switch
set_property -dict { PACKAGE_PIN F16    IOSTANDARD LVCMOS33 } [get_ports { clk }];
set_property -dict { PACKAGE_PIN D14    IOSTANDARD LVCMOS33 } [get_ports { rst }];
#Led
set_property -dict { PACKAGE_PIN G18    IOSTANDARD LVCMOS33 } [get_ports { led[7] }];
set_property -dict { PACKAGE_PIN F18    IOSTANDARD LVCMOS33 } [get_ports { led[6] }];
set_property -dict { PACKAGE_PIN E17    IOSTANDARD LVCMOS33 } [get_ports { led[5] }];
set_property -dict { PACKAGE_PIN D17    IOSTANDARD LVCMOS33 } [get_ports { led[4] }];
set_property -dict { PACKAGE_PIN G17    IOSTANDARD LVCMOS33 } [get_ports { led[3] }];
set_property -dict { PACKAGE_PIN E18    IOSTANDARD LVCMOS33 } [get_ports { led[2] }];
set_property -dict { PACKAGE_PIN D18    IOSTANDARD LVCMOS33 } [get_ports { led[1] }];
set_property -dict { PACKAGE_PIN C17    IOSTANDARD LVCMOS33 } [get_ports { led[0] }];
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets {clk_IBUF}];
```

将生成的 bit 文件烧写到 FPGAOL 平台，观察运行结果，可以发现，

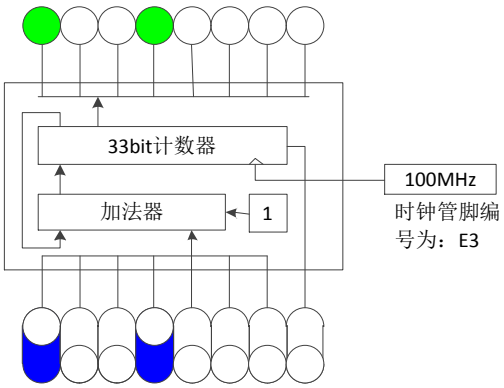
每个时钟周期 LED 灯都会左移一位。如下图所示：



用户可在此设计基础上进行修改以增加功能，如通过开关控制跑马灯移动方向、通过开关控制跑马灯的初始状态等。

Step4：计数器

下面我们将完成一个计数器电路，采用板载的 100MHz 时钟作为计数时钟，最右侧的开关作为复位，左侧的 7 个开关对应的数字作为计数步长 cnt_step，每个时钟周期，计数器数值增加（cnt_step+1），计数器的高 8 位输出显示到 8 个 LED 灯上。其电路结果如下图所示：



其 Verilog 代码为：

```
module step4_cnt(
input           clk,rst,
input   [6:0]   step,
output  [7:0]   led);
reg [32:0]  cnt;
always@(posedge clk or posedge rst)
begin
    if(rst)
        cnt <= 33'h1_5555_FFFF;
    else
```

```
        cnt <= cnt + step +1'b1;
end
assign  led = cnt[32:25];
endmodule
```

其管脚约束文件为：

```
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }];
#Switch
set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 } [get_ports { cnt_step[6] }];
set_property -dict { PACKAGE_PIN G13   IOSTANDARD LVCMOS33 } [get_ports { cnt_step[5] }];
set_property -dict { PACKAGE_PIN F13   IOSTANDARD LVCMOS33 } [get_ports { cnt_step[4] }];
set_property -dict { PACKAGE_PIN E16   IOSTANDARD LVCMOS33 } [get_ports { cnt_step[3] }];
set_property -dict { PACKAGE_PIN H14   IOSTANDARD LVCMOS33 } [get_ports { cnt_step[2] }];
set_property -dict { PACKAGE_PIN G16   IOSTANDARD LVCMOS33 } [get_ports { cnt_step[1] }];
set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 } [get_ports { cnt_step[0] }];
set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 } [get_ports { rst }];
#Led
set_property -dict { PACKAGE_PIN G18   IOSTANDARD LVCMOS33 } [get_ports { led[7] }];
set_property -dict { PACKAGE_PIN F18   IOSTANDARD LVCMOS33 } [get_ports { led[6] }];
set_property -dict { PACKAGE_PIN E17   IOSTANDARD LVCMOS33 } [get_ports { led[5] }];
set_property -dict { PACKAGE_PIN D17   IOSTANDARD LVCMOS33 } [get_ports { led[4] }];
set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports { led[3] }];
set_property -dict { PACKAGE_PIN E18   IOSTANDARD LVCMOS33 } [get_ports { led[2] }];
set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 } [get_ports { led[1] }];
set_property -dict { PACKAGE_PIN C17   IOSTANDARD LVCMOS33 } [get_ports { led[0] }];
```

最后,将生成的bit文件烧写到FPGAOL实验平台,观察运行结果。特别提醒一下,由于在线实验平台是按一定频率对输出结果进行采样并显示在浏览器中，因此当信号变化频率较高时，可能会由于采样率不够高，导致显示效果与实际运行结果不一致，请同学们注意。

有兴趣的同学可以试着将计数器计数频率增加，观察运行结果（实验平台可能会崩溃或者报错）。

Step5：呼吸灯设计

请同学们根据前面的设计经验，自行设计一个呼吸灯程序，能够正确的运行在FPGAOL在线实验平台，且程序最好能具备以下功能：

a,可以设置同时亮灯的数量，如4个或5个亮灯的呼吸灯

b,呼吸灯变化方式可设置，如渐明渐暗、忽明渐暗等

c,呼吸灯呼吸频率可调

d,其它可设置的参数

注意：本步骤没有亲测，因此在线实验平台上的实现效果未知，有待感兴趣的同学亲自验证。

总结与思考

1.请总结本次实验的收获

2.请评价本次实验的难易程度

3.请评价本次实验的任务量

4.请为本次实验提供改进建议