点亮数字人生 实验报告

2018011365 张鹤潇

2020-03-22

目录

1 实验目的

- 通过数码管点亮程序，熟悉 VHDL 语言，了解硬件程序的编写规范。
- 掌握 EDA 软件的使用方法和工作流程。
- 进一步理解可编程芯片的工作原理。

2 实验任务

使用至多两个带译码的数管和和至少一个不带译码的数码管，有规律地显示奇数列、偶数列、自然数列。

3 实验代码和分析

3.1 实验源码

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_ARITH.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;

entity digitalLife is
    port(
        light_even_4: out std_logic_vector(3 downto 0); --带译码的偶数列
        light_odd_4: out std_logic_vector(3 downto 0); --带译码的奇数列
        light_natural_7: out std_logic_vector(6 downto 0); --不带译码的自然数列
        clk, rst: in std_logic --clk对应clk接口; rst作为复位器, 对应rst接口
    );
end digitalLife;

architecture lighting of digitalLife is
    signal signal_natural_7: std_logic_vector(3 downto 0):="0000";
    signal signal_odd_4: std_logic_vector(3 downto 0):="0001";
    signal signal_even_4: std_logic_vector(3 downto 0):="0000";
    signal cnt: integer:=0; --计数器
begin
    process(clk, rst) --clk/rst变化时触发, 只改变signal
```

```
begin
    if(clk'event and clk='1') then --clk被按下，计数器加1
        if (cnt < 1000000) then
            cnt <= cnt + 1;
        else
            cnt <= 0;
            if (signal_even_4 ="1000") then --偶数列 8->0
                signal_even_4 <= "0000";
            else
                signal_even_4 <= signal_even_4 + 2;
            end if;
            if (signal_odd_4="1001") then -- 奇数列 9->1
                signal_odd_4 <= "0001";
            else
                signal_odd_4 <= signal_odd_4 + 2;
            end if;
            if (signal_natural_7="1001") then --只显示自然数列0-9
                signal_natural_7 <= "0000";
            else
                signal_natural_7 <= signal_natural_7 + '1';
            end if;
        end if;
    end if;
    if (rst='1') then --复原计数器
        signal_natural_7 <= "0000";
        signal_odd_4 <= "0001";
        signal_even_4 <= "0000";
    end if;
end process;

process(signal_even_4) --改变偶数列的输出
begin
    light_even_4 <= signal_even_4;
    end process;

process(signal_odd_4) --改变奇数列的输出
begin
    light_odd_4 <= signal_odd_4;
end process;

process(signal_natural_7) --改变自然数列的输出
begin
    case( signal_natural_7 ) is
        when "0000" => light_natural_7<="1111110";
        when "0001" => light_natural_7<="0110000";
```

```
            when "0010" => light_natural_7<="1101101";
            when "0011" => light_natural_7<="1111001";
            when "0100" => light_natural_7<="0110011";
            when "0101" => light_natural_7<="1011011";
            when "0110" => light_natural_7<="1011111";
            when "0111" => light_natural_7<="1110000";
            when "1000" => light_natural_7<="1111111";
            when "1001" => light_natural_7<="1111011";
            when "1010" => light_natural_7<="1110111";
            when "1011" => light_natural_7<="0011111";
            when "1100" => light_natural_7<="1001110";
            when "1101" => light_natural_7<="0111101";
            when "1110" => light_natural_7<="1001111";
            when "1111" => light_natural_7<="1000111";
            when others => light_natural_7<="0000000";
        end case;
    end process;
end lighting;
```

3.2 工作原理

输出端包括 light_even_4，light_odd_4、light_natural_7 、clk 和 rst.

在 architecture 中，设置了四个 signal，其中 cnt 作为计数器，另三个 signal 负责向输出端传递信号。

clk 每按下一次，cnt 就加 1，当 $cnt \geq 1M$ 时，三个数码管就显示数列的下一个数字，同时 cnt 归零。rst 作为复位器，按下后所有数码管保持初始值。

process(clk, rst) 在 clk 或 rst 被按下时触发，根据相应的 event 改变记录信号的三个 signal 的状态。

process(signal_even_4), process(signal_odd_4), process(signal_natural_7) 在相应的信号发生变化时触发，负责改变数码管的输出。

3.3 测试效果

3.3.1 JieLabs 平台在线测试

在 JieLabs 实验平台上进行测试。两个带译码的数码管分别显示奇数列和偶数列，不带译码的数码管显示自然数列，当输入时钟信号为 2MHz 时，三个数列的数字每 0.5 秒跳动一次。

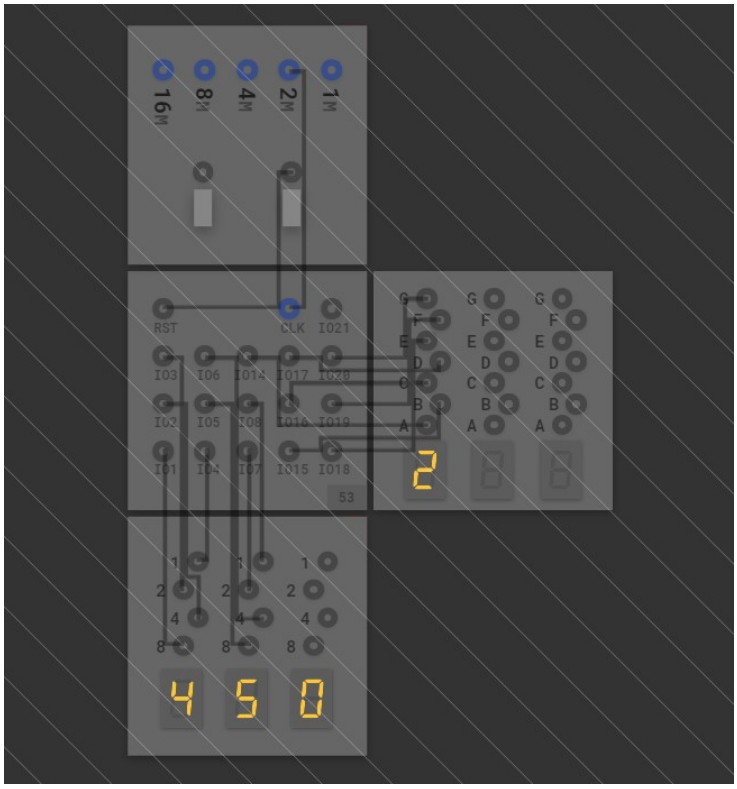

图 1: JieLabs 平台运行效果

3.3.2 Modelsim-Alter 仿真

用 Modelsim-Alter 进行仿真测试。对代码略作修改。当 clk 被按下时，各数码管直接显示下一个数字。testbench 关键代码如下。

```
for i in 0 to 2 loop
    clk <= '1';
    wait for clock_period;
    clk <= '0';
    wait for clock_period;
end loop;
rst <= '1';
wait for clock_period;
rst <= '0';
wait for clock_period;
for i in 0 to 4 loop
    clk <= '1';
    wait for clock_period;
    clk <= '0';
    wait for clock_period;
end loop;
```

先按下 3 次 clk，然后再按下 1 次 rst，之后按下 5 次 clk，每次按键高电平

10ns，低电平 10ns. 仿真结果符合预期。

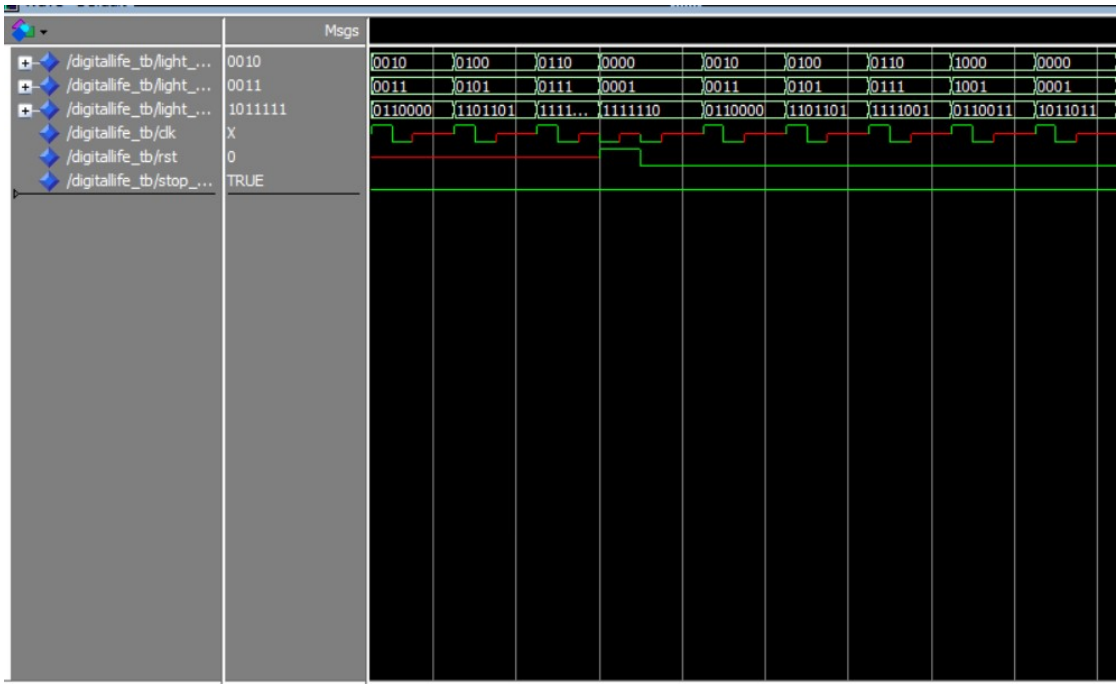

图 2: Modelsim-Alter 仿真效果

4 问题及其解决

VHDL 作为一门硬件编程语言，具有许多独特的特性。在编写程序时，我踩了很多坑。

4.1 signal 的延迟赋值

在 VHDL 中，signal 使用 <= 赋值，在进程结束时赋值；variable 使用:= 赋值，没有延迟，立刻赋值。因此，如果在某一进程中执行 a<=b,c<=a，则执行结束后，a,c 储存的是 b,a 的原值。

这也是代码中单独设计了三个 process 用于给数列赋值的原因。

4.2 process 中的赋值

在 VHDL 中，不同 process 不能给同一 signal/variable 赋值。这应该与 VHDL 的并行性有关——这与多线程编程中，多个线程同时写入同一内存变量可能引发错误是一样的道理。

另外，port 属于特殊的 signal，所以下列做法会导致数列 vector 在两个进程中被改变，是不可行的：

```
process(rst)
... -- do some changes
process(clk)
... -- do some changes
```

4.3 event 在 process 中的唯一性

又是一个非常奇怪的特性。举例而言，如果在某个 process 中:

```
if (clk'event and clk='1') then --clk被按下，各计数器加1
    ...
if (rst'event and rst='1') then --复原计数器
    ...
```

就会引发编译错误。这可能是因为 event 的时间很短，不存在两个 event 同时发生的情况。

4.4 event 的调用位置

```
if (rst='1') then --复原计数器
    ...
if (clk'event and clk='1') then --clk被按下，各计数器加1
    ...
```

交换一下这两行语句的位置，就会引发错误。这可能是因为 event 的时间很短，应当立即被处理。对于这个特性，我感到有些困惑。

5 总结反思

VHDL 的特性与我之前用过的编程语言，如 python，C/C++ 在逻辑上有很大的不同。为了理解这些特性，需要从硬件运行的角度进行思考。

JieLabs 实验平台 UI 设计简洁又美观，我在该网站上多次测试的过程中没有遇到 bug；该平台的上线让我们足不出户就能收获看到自己的数码管被点亮的成就感。