西安电子科技大学

电子线路实验(II) 课程实验报告

实验名称 大规模集成数字电路设计

机电工程 学院 2004031 班

小组成员：

姓名 刘杰 学号 20049200159

姓名 侯钧宇 学号 20049200209

姓名 杨英豪 学号 20049200375

姓名 孟庆豪 学号 20049200181

姓名 张子谦 学号 20049200079

成 绩

实验日期 2021 年 11 月 20 日

指导教师评语：
指导教师： 年 月 日

实验报告内容基本要求及参考格式

一、实验目的

二、实验所用仪器（或实验环境）

三、实验基本原理及步骤（或方案设计及理论计算）

四、实验数据记录（或仿真及软件设计）

五、实验结果分析及回答问题（或测试环境及测试结果）

状态机数码控制系统

一、状态机的一般组成

用 VHDL 设计有限状态机方法有多种，但最一般和最常用的状态机设计通常包括说明部分，主控时序部分，主控组合部分和辅助进程部分。

1.说明部分

说明部分中使用 TYPE 语句定义新的数据类型，此数据类型为枚举型，一其元素通常都用状态机的状态名来定义。状态变量定义为信号，便于信息传递，并将状态变量的数据类型定义为含有既定状态元素的新定义的数据类型。说明部分一般放在结构体的 ARCHITECTURE 和 BEGIN 之间。

2.主控时序进程

是指负责状态机运转和在时钟驱动正负现状态机转换的进程。状态机随外部时钟信号以同方式工作，当时钟的有效跳变到来时，时序进程将代表次态的信号 next_state 中的容送入现态信号 current_state 中，而 next_state 中的容完全由其他进程根据实际情况而定，此进程中往往也包括一些清零或置位的控制信号。

3.主控组合进程

根据外部输入的控制信号（包括来自外部的和状态机容的非主控进程的信号）或（和)当前状态值确定下一状态 next_state 的取值容，以及对外或对部其他进程输出控制信号的容。

4.辅助进程

用于配合状态机工作的组合、时序进程或配合状态机工作的其他

时序进程。在一般状态机的设计过程中，为了能获得可综合的，高效的 VHDL 状态机播述，建议使用枚举类数据类型来定义状态机的状态，一并使用多进程方式来描述状态机的部逻辑。一例如可使用两个进程来描述，一个进程描述时序逻辑，包括状态寄存器的工作和寄存器状态的输出，另一个进程描述组合逻辑，一包括进程间状态值的传递逻辑以及状态转换值的输出。必要时还可以引入第三个进程完成其它的逻辑功能。

下例描述的状态机由两个主控进程构成，其中进程 IG 为主控时序进程，RNG 为主控组
合进程，EDG 为辅助进程。

二、崭新点

状态机的输出不仅与当前的状态有关，还与当前的输入有关，具有时序功能、记忆功能。同时可利用主控组合进程和辅助进程分别完成不同的逻辑功能，我们可以通过修改辅助进程的一些重要参数，从而轻松达到想要实现的目标。

三、研制成本

1.时间成本

状态机数码系统的时间成本较为适中，主要用于开发运营新功能，以及协调主要控制状态和辅助控制状态，其中最为关键的是运用状态机的状态变换特性来实现被控制对象的激励变化特性属性。

2.经济成本

状态机数码系统的经济研发成本较为低廉，可用于大规模的系统研发和生产。

四、性价比分析

状态机以较低的经济成本，完成并可以成功控制较大规模的系统控制，性价比较高。

实现状态机数码控制系统：

步骤如下：

1. 编写 VHDL 程序：

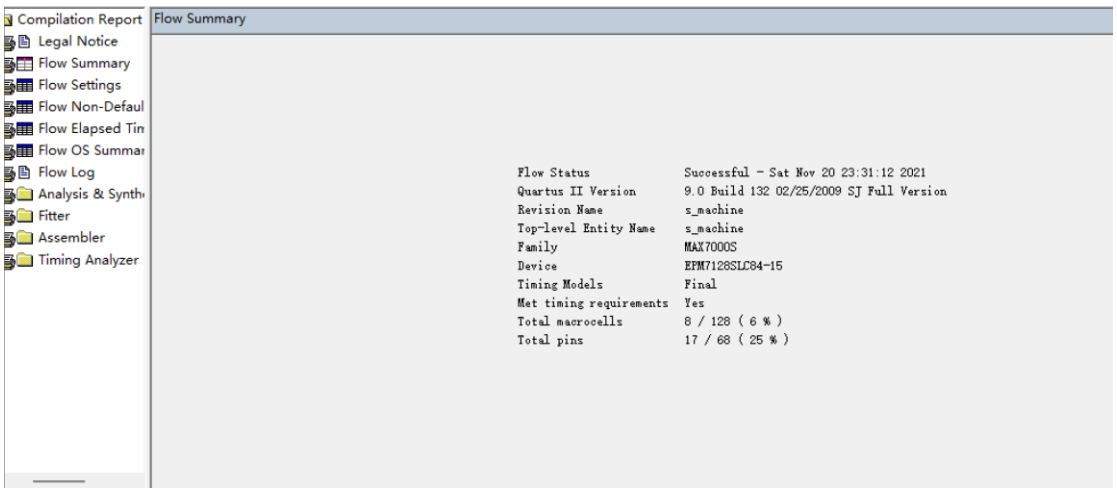
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY s_machine IS
PORT ( clk,reset    : IN STD_LOGIC;
```

```
        state_inputs : IN STD_LOGIC;
         sel:in std_LOGIC_VECTOR(1 DOWNTO 0);
         cout:out std_LOGIC_VECTOR(0 to 2);
         comb_outputs : OUT INTEGER RANGE 0 TO 31);

END s_machine;

ARCHITECTURE LPL OF s_machine IS
  TYPE FSM_ST IS (s0, s1, s2, s3, s4, s5, s6, s7);
  SIGNAL current_state, next_state: FSM_ST;
BEGIN
IG: PROCESS (reset,clk)
  BEGIN
    IF reset = '1' THEN   current_state <= s0;
    ELSIF clk='0' AND clk'EVENT THEN
      current_state <= next_state;
    END IF;
  END PROCESS;
RNG:PROCESS(current_state, state_Inputs)
  BEGIN
    CASE current_state IS
      WHEN s0 =>cout<="000" ;
        IF state_inputs ='0'   THEN  next_state<=s0;
        ELSE  next_state<=s1;
        END IF;
      WHEN s1 =>cout<="001" ;
        IF state_inputs = '0'   THEN  next_state<=s1;
        ELSE  next_state<=s2;
        END IF;
      WHEN s2 =>cout<="010" ;
        IF state_inputs = '0'  THEN  next_state <= s2;
        ELSE  next_state<=s3;
        END IF;
      WHEN s3 =>cout<="011" ;
        IF state_inputs = '0'  THEN  next_state <= s3;
        ELSE  next_state<=s4;
              END IF;
      WHEN s4 =>cout<="100" ;
        IF state_inputs = '0'  THEN  next_state <= s4;
        ELSE  next_state<=s5;
              END IF;
      WHEN s5 =>cout<="101" ;
        IF state_inputs = '0'  THEN  next_state <= s5;
        ELSE  next_state <= s6;
```

```
        END IF;
    WHEN s6 =>cout<="110" ;
     IF state_inputs = '0'  THEN  next_state <= s6;
     ELSE  next_state <= s7;
        END IF;
    WHEN s7 =>cout<="111" ;
     IF state_inputs = '0'  THEN  next_state <= s7;
     ELSE  next_state <= s0;
     END IF;
  END case;
 END PROCESS;

EDG:PROCESS(sel,current_state)
     BEGIN
     if    sel="00"then
        CASE current_state IS
            WHEN s0=> comb_outputs<= 1;
            WHEN s1=> comb_outputs<= 2;
            WHEN s2=> comb_outputs<= 4;
            WHEN s3=> comb_outputs <=8;
            WHEN s4=> comb_outputs <=16;
            WHEN s5=> comb_outputs <=0;
            WHEN s6=> comb_outputs <=31;
            WHEN s7=> comb_outputs <=0;
        END CASE;
     elsif sel="01"      then
         CASE current_state IS
            WHEN s0=> comb_outputs <= 16;
            WHEN s1=> comb_outputs <= 8;
            WHEN s2=> comb_outputs <= 4;
            WHEN s3=> comb_outputs <= 2;
            WHEN s4=> comb_outputs <= 1;
            WHEN s5=> comb_outputs <= 0;
            WHEN s6=> comb_outputs <= 31;
            WHEN s7=> comb_outputs <= 0;
        END CASE;
     elsif sel="10"   then
        CASE current_state IS
            WHEN s0=> comb_outputs<= 1;
            WHEN s1=> comb_outputs<= 4;
            WHEN s2=> comb_outputs<= 16;
            WHEN s3=> comb_outputs <=2;
            WHEN s4=> comb_outputs <=8;
            WHEN s5=> comb_outputs <=0;
```

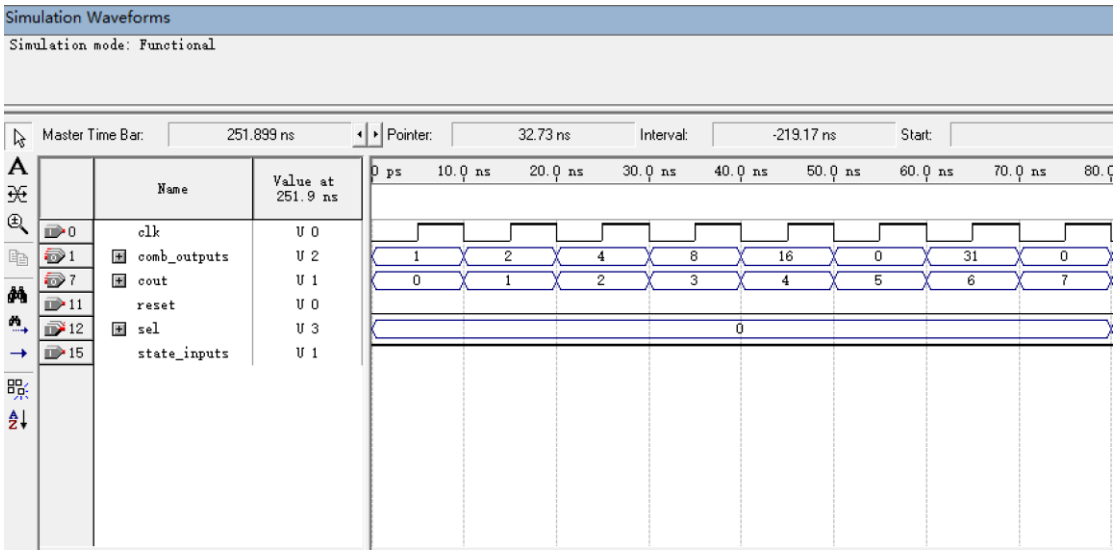
```
            WHEN s6=> comb_outputs <=31;
            WHEN s7=> comb_outputs <=0;
        END CASE;
    elsif sel="11"   then
        CASE current_state IS
            WHEN s0=> comb_outputs <= 8;
            WHEN s1=> comb_outputs <= 2;
            WHEN s2=> comb_outputs <= 16;
            WHEN s3=> comb_outputs <= 4;
            WHEN s4=> comb_outputs <= 1;
            WHEN s5=> comb_outputs <= 0;
            WHEN s6=> comb_outputs <= 31;
            WHEN s7=> comb_outputs <= 0;
        END CASE;
    end if;
END PROCESS;
END LPL;
```

2. 编译运行：

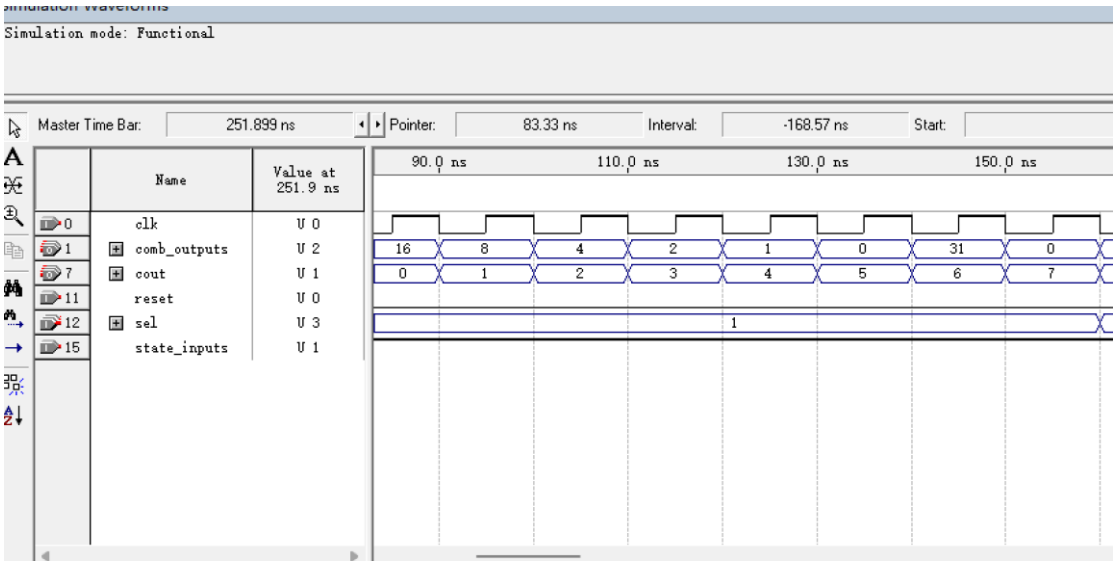


3. 波形仿真及解释：

实现状态机功能，当前状态随 clk 变化而移向下一状态，辅助进程也随之改变。



第一部分：如图，clk 端为周期为 10ns 的占空比 50%的方波，复位 reset 端设置为 0，输入端 state_inputs 设置为高电平，同时功能控制 sel 端表现为 0，表现为功能一。根据 clk 端下降沿到来时，主控制输出端 cout 由 0 到 7 共 8 个状态依次移动，同时辅助进程输出端 comb_outputs 的状态向下一状态转变，显示相应的控制功能 1，即从 1->2->4->8->16->0->31->0。



第二部分：如图，复位 reset 端为 0，输入端 state_inputs 设置为高

电平，同时功能控制 sel 端表现为 1，表现为功能二。在 clk 端下降沿到来的时候，主控制输出端 cout 由 0 到 7 共 8 个状态依次移动，同时辅助进程输出端 comb_outputs 的状态向下一状态转变，显示相应的控制功能 2，即从 16->8->4->2->1->0->31->0。

自评成绩

刘杰 100

侯钧宇 75

杨英豪 75

孟庆豪 75

张子谦 75