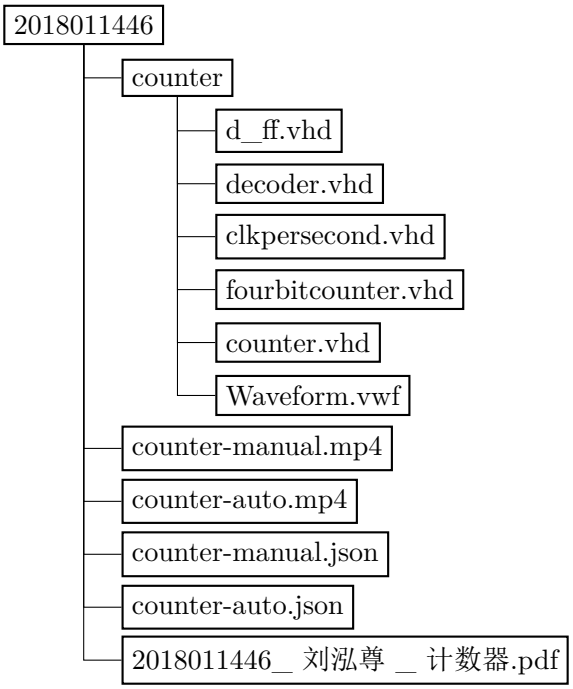
计数器的设计：实验报告

刘泓尊　2018011446　计 84

2020 年 4 月 1 日

目录

1 File Structure



2 实验目的

(1) 掌握时序逻辑电路的基本分析和设计方法.

(2) 理解同步时序逻辑电路和异步时序逻辑电路的区别.

(3) 掌握计数器电路设计原理，用硬件描述语言实现指定功能的计数器设计.

(4) 学会利用软件仿真实现对数字电路的逻辑功能进行验证和分析.

3 实验任务

(1) 使用两个未经译码处理的数码管显示计数，手动单次时钟进行计数，到 59 的时候回到 00。

(2) 使用实验平台上的 1MHz 时钟，将计数器改成秒表，并使用开关控制秒表的启动、暂停。

4 集成“手动”与“秒表”版本

支持“手动 clk/秒表 clk/暂停/复位”功能。

我将两个部分功能实现了集成，$mode =' 0'$ 的时候是手动模式，可以手动按下 clk 实现计数器的加一。$mode =' 1'$ 是秒表模式，可以开启秒表模式，在当前计数基础上进行每 1s 一次的计数。在实现过程中，我使用了大量“元件例化”的方法，并且没有使用”+”。本部分代码位于./counter 下。

4.1 D 触发器的实现

践行模块化设计的思路，我实现了 D 触发器，实现如下:

d_ff.vhd

```
-- 异步复位D触发器 --
entity d_ff is
    port(
        clk: in std_logic;
        rst: in std_logic;
        pause: in std_logic;
        d: in std_logic;
        q: out std_logic;
        nq: out std_logic
    );
end d_ff;

architecture bhv of d_ff is
begin
    process(clk, rst) begin
        if rst = '1' then
            q <= '0';
            nq <= '1';
        elsif clk'event and clk='1' then
```

```
            if pause='0' then
                q <= d;
                nq <= not d;
            else --维持不变
                q <= not d;
                nq <= d;
            end if;
        end if;
    end process;
end bhv;
```

4.2 计时器模块

为了实现“计时”与“计数”的解耦，我将计时器单独作为一个模块。该模块通过记录时钟上升沿个数，每个 1M 次时钟周期产生一次脉冲。在 1MHz 的时钟频率下，该模块将产生每 1s 一次的脉冲，代码如下:

clkpersecond.vhd

```
-- 产生秒表序列(时钟频率1MHz: 产生1s的周期序列) --
entity clkpersecond is
    port(
        clk: in std_logic;
        rst: in std_logic;
        clkps: out std_logic
    );
end clkpersecond;

architecture bhv of clkpersecond is
    signal outclk: std_logic := '0';
    signal count: integer := 0;
    constant FREQUENCY: integer := 500000;--设置时钟频率1MHz(即1000000/2)
begin
    clkps <= outclk;
    process(clk, rst) begin
        if rst = '1' then
            count <= 0;
            outclk <= '0';
        elsif clk'event and clk='1' then
            if count < FREQUENCY then
                count <= count + 1;
            else
                count <= 0;
                outclk <= (not outclk);--跳变
            end if;
        end if;
```

```
    end process;
end bhv;
```

4.3 4 位加法计数器

在译码处理前，由于第一位表示 0-9 的数字，第二位表示 0-5 的数据，所以需要 4 位二进制来表示。为了进一步解耦，我实现了用四位二进制数表示计数个数的四位计数器。该模块根据 d 触发器实现时序逻辑的设计，“十分底层”地实现了计数器，而不是使用简单地 +1。具体实现如下:

fourbitcounter.vhd

```
-- 四位加法计数器 --
entity fourbitcounter is
    port(
        clk: in std_logic;
        rst: in std_logic;
        pause: in std_logic;
        qvec: out std_logic_vector(3 downto 0)
    );
end fourbitcounter;

architecture bhv of fourbitcounter is
    component d_ff
        port(
            clk: in std_logic;
            rst: in std_logic;
            pause: in std_logic;
            d: in std_logic;
            q: out std_logic;
            nq: out std_logic
        );
end component;
    signal q_buf: std_logic_vector(3 downto 0);
    signal nq_buf: std_logic_vector(3 downto 0);
begin
    --使用4个d触发器实现4位二进制的自增
    u0: d_ff port map(clk=>clk, rst=>rst, pause=>pause, d=>nq_buf(0), q=>
        q_buf(0), nq=>nq_buf(0) );
    u1: d_ff port map(clk=>nq_buf(0), rst=>rst, pause=>pause, d=>nq_buf(1),
        q=>q_buf(1), nq=>nq_buf(1) );
    u2: d_ff port map(clk=>nq_buf(1), rst=>rst, pause=>pause, d=>nq_buf(2),
        q=>q_buf(2), nq=>nq_buf(2) );
    u3: d_ff port map(clk=>nq_buf(2), rst=>rst, pause=>pause, d=>nq_buf(3),
        q=>q_buf(3), nq=>nq_buf(3) );
    qvec <= q_buf;
end bhv;
```

4.4 计数器

有了上面实现的四位计数器和计时器模块，可以很方便地实现计数器了。该模块是顶层实体，通过计时器识别时钟信号并给出 1s 一次的脉冲，之后四位计数器实现一次自增，最终将输出结果经过译码器 (decoder 模块与“点亮数字人生”中的模块相同，在此不予赘述) 进行显示。代码如下:

counter.vhd

```
-- 计数器(用数码管显示，支持手动clk/秒表clk/暂停/复位 功能, 60进制) --
entity counter is
    port(
        clk: in std_logic;
        rst: in std_logic;
        pause: in std_logic;
        mode: in std_logic;--mode = '1'的时候是秒表，否则是手动加
        highcode: out std_logic_vector(6 downto 0);
        lowcode: out std_logic_vector(6 downto 0)
    );
end counter;

architecture bhv of counter is
    component decoder--译码器
        port(
            bit_4_vec: in std_logic_vector(3 downto 0);
            bit_7_vec: out std_logic_vector(6 downto 0)
        );
    end component;
    component fourbitcounter--四位加法计数器
        port(
            clk: in std_logic;
            rst: in std_logic;
            pause: in std_logic;
            qvec: out std_logic_vector(3 downto 0)
        );
    end component;
    component clkpersecond--产生1s一个上升沿的周期信号
        port(
            clk: in std_logic;
            rst: in std_logic;
            clkps: out std_logic
        );
    end component;
   signal clockUsing: std_logic;--当前使用的clk, 用于区分手动和秒表
    signal clockPerSecond: std_logic;--每1s一次上升沿的clk
```

```
    signal tempresetLow: std_logic := '0';
    signal tempresetHigh: std_logic := '0';
    signal resetLow: std_logic := '0';--低位复位
    signal resetHigh: std_logic := '0';--高位复位
    signal outputLow: std_logic_vector(3 downto 0);--低位输出(4位二进制)
    signal outputHigh: std_logic_vector(3 downto 0);--高位输出(4位二进制)
begin
    clockUsing <= clockPerSecond when mode = '1' else clk;
    getclk:clkpersecond port map(clk=>clk, rst=>rst, clkps=>clockPerSecond);
    u1:fourbitcounter port map(clk=>clockUsing, rst=>resetLow, pause=>pause,
        qvec=>outputLow );
    u2:fourbitcounter port map(clk=>resetLow, rst=>resetHigh, pause=>pause,
       qvec=>outputHigh );
    decoder1: decoder port map(bit_4_vec=>outputLow, bit_7_vec=>lowcode);
    decoder2: decoder port map(bit_4_vec=>outputHigh, bit_7_vec=>highcode);
    tempresetLow <= '1' when outputLow = "1010" else '0';--到10进1
    tempresetHigh <= '1' when outputHigh >= "0110" else '0';--到60归零
    resetLow <= rst or tempresetLow; resetHigh <= rst or tempresetHigh;
end bhv;
```

5 仿真结果

使用 Quartus 的 ModelSim 进行仿真 (附"**./counter/Waveform.vwf**" 文件). 进行功能仿真的结果如下，第一幅图展示了包含“暂停”和“复位”功能。输出时经过译码后的 7 位二进制数。第二幅图展示了到达 59 之后自动回到 0 的过程。

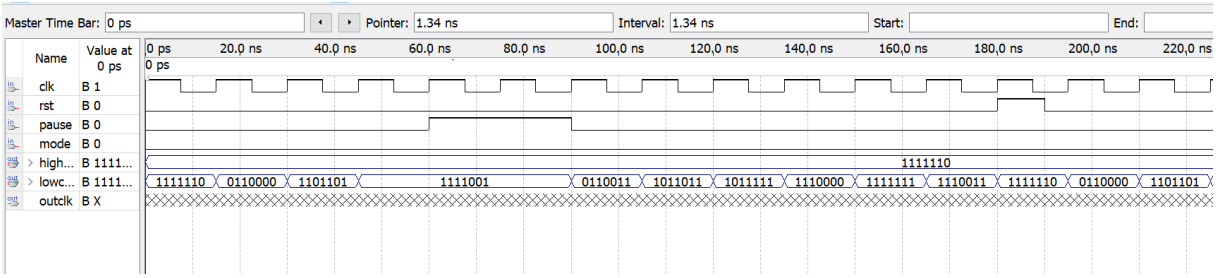


图 1: 包含“暂停”和“复位”功能

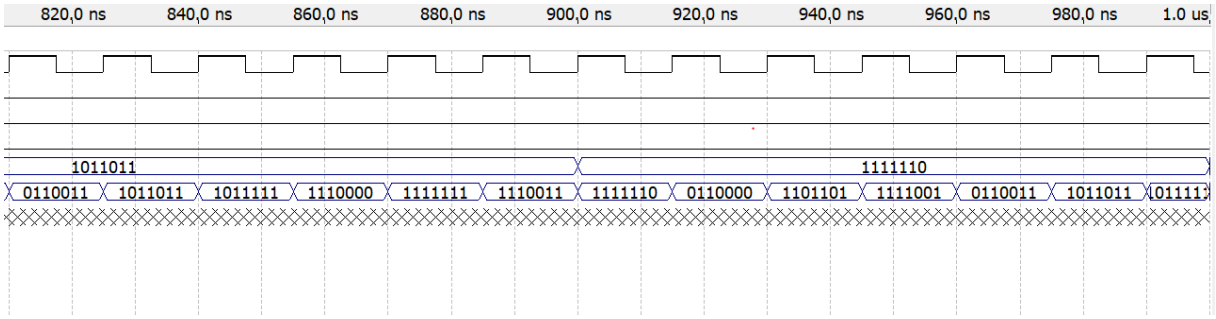


图 2: 到达 59 之后自动回到 0

下图是时序仿真的结果。展示了包含“暂停”和“复位”功能。

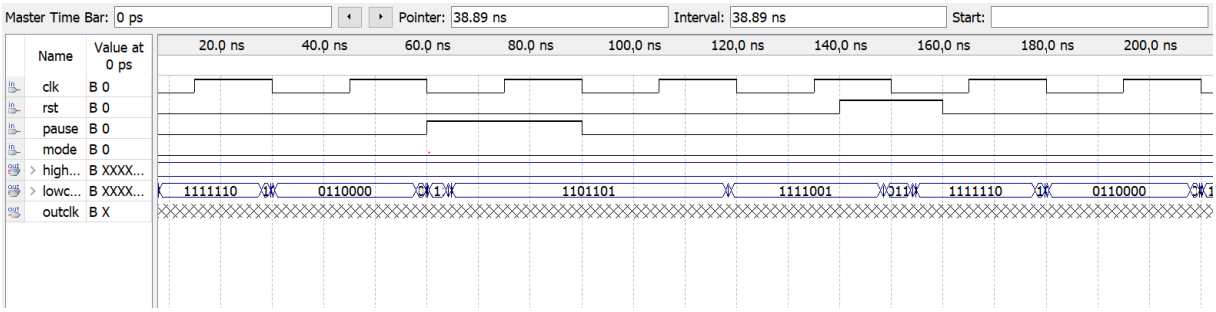


图 3: 时序仿真

6 JieLab 运行结果 (附录屏)

我将代码放在了 JieLabs 实验平台上进行了验证，下面是运行时的截图，在目录下有”**counter-manual.mp4**”(手动计数) 和”**conuter-auto.mp4**”(秒表功能)，视频中还展示了“暂停”，“复位”等功能。您可以更直观地检查我的实现效果。同时附件中提供了从 JieLabs 存档的“counter-manual.json”和“conuter-auto.json”。

注: 左下角板块的第一个开关代表“mode”，mode = 1 时是秒表模式，mode = 0 是手动模式。第二个开关代表“暂停”。

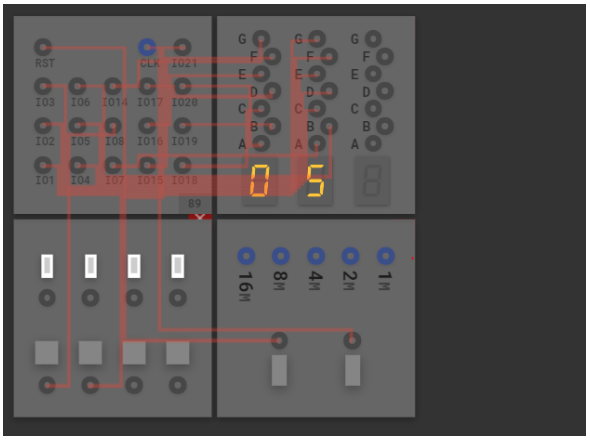


图 5: 手动模式-截图

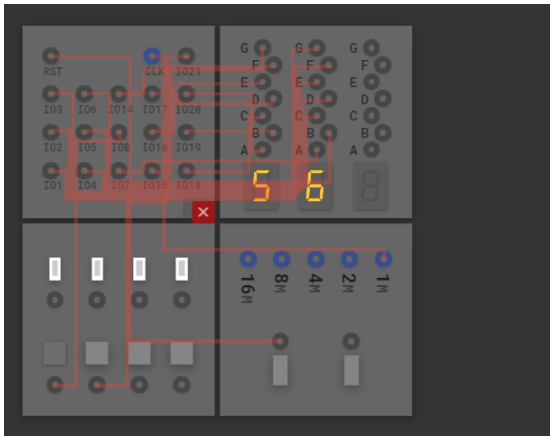


图 6: 秒表模式-截图

7 实验总结

本次实验我采用了大量的“元件例化”方法，在极大程度上实现了“解耦”与模块化设计，我的 VHDL 编程能力得到了极大提升。我自己实现了 d 触发器，进而使用时序逻辑的方式实现了计数器，而不是简单地使用”+1” 语句，切身体会到了时序逻辑的设计方法和电路特点，没有依赖软件的编译与自动转换，对时序逻辑的增进了理解。同时，我也被硬件所能实现的众多有趣功能所吸引，希望在将来能设计出更多有意义的电路。感谢老师助教在微信群中的耐心指导!