计数器的设计

傅舟涛 2017010682

零、实验内容与说明

1、实验内容

使用实验平台上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，时钟上升沿计数一次，当计数到 59 的时候，要求两个数码管都能复位到 00 的状态，重新计数，实验还要求设置一个复位键，可以随时重新恢复到 00 的状态继续计数。使用实验平台上的 1MHz 时钟，将计数器改成秒表，在秒表中使用开关控制秒表启动暂停。

2、说明

由于我的理解错误，导致最初搭建 D 触发器时使用的六个与非门，这让我走了许多弯路，不过在解决问题的过程中有了不少收获，记录在第一部分中。第二部分是用 event 写的 D 触发器搭建的同步与异步电路，第三部分是完整代码、仿真结果，第四部分是收获。

第一部分较长，因此该部分每节的关键信息均**加重**表示，且在结尾 P10 有总结。

一、六与非门 D 触发器

1、D 触发器与 16 进制计数器的搭建：元件例化与延迟，初始化

六与非门搭建 D 触发器有两种方式：采用元件例化的方式，先搭建与非门，再搭建 D 触发器；直接搭建 D 触发器，如图 1 所示。

这两者单独仿真结果均正常，但将其串联成 16 进制计数器得到的结果不同，如图 2 所示。原因之一是，**元件例化改变了综合方式，阻止了 quartus 自优化，增加了一级门延迟**，如图 3 所示。（还有其他原因，见后文）

```
entity DTrigger is
    port(
        D, CP, Rd, Sd: in std_logic;
        Q: out std_logic := '0';
        nQ: out std_logic := '1'
    );
end DTrigger;

architecture T of DTrigger is
    component NAnd03
        port(
            a, b, c: in std_logic;
            d: out std_logic
        );
    end component;
    signal ot: std_logic_vector(6 downto 1);
begin
    ot(1)<=not(ot(3) and D and Rd);
    ot(2)<=not(ot(1) and Sd and ot(4));
    ot(3)<=not(CP and ot(1) and ot(4));
    ot(4)<=not(Rd and ot(2) and CP);
    ot(5)<=not(Rd and ot(3) and ot(6));
    ot(6)<=not(ot(5) and ot(4) and Sd);
    nQ<=ot(5);
    Q<=ot(6);
end T;
```

```
entity NAnd03 is
    port(
        a, b, c: in std_logic;
        d: out std_logic
    );
end NAnd03;

architecture N3 of NAnd03 is
begin
    d<=not(a and b and c);
end N3;
architecture T of DTrigger is
    component NAnd03
        port(
            a, b, c: in std_logic;
            d: out std_logic
        );
    end component;
    signal ot: std_logic_vector(6 downto 1);
begin
    N1:NAnd03 port map(ot(3),D,Rd,ot(1));
    N2:NAnd03 port map(ot(1),Sd,ot(4),ot(2));
    N3:NAnd03 port map(CP,ot(1),ot(4),ot(3));
    N4:NAnd03 port map(Rd,ot(2),CP,ot(4));
    N5:NAnd03 port map(Rd,ot(3),ot(6),ot(5));
    N6:NAnd03 port map(ot(5),ot(4),Sd,ot(6));
    nQ<=ot(5);
    Q<=ot(6);
end T;
```

图 1 不例化 D 触发器（左）与例化 D 触发器（右）

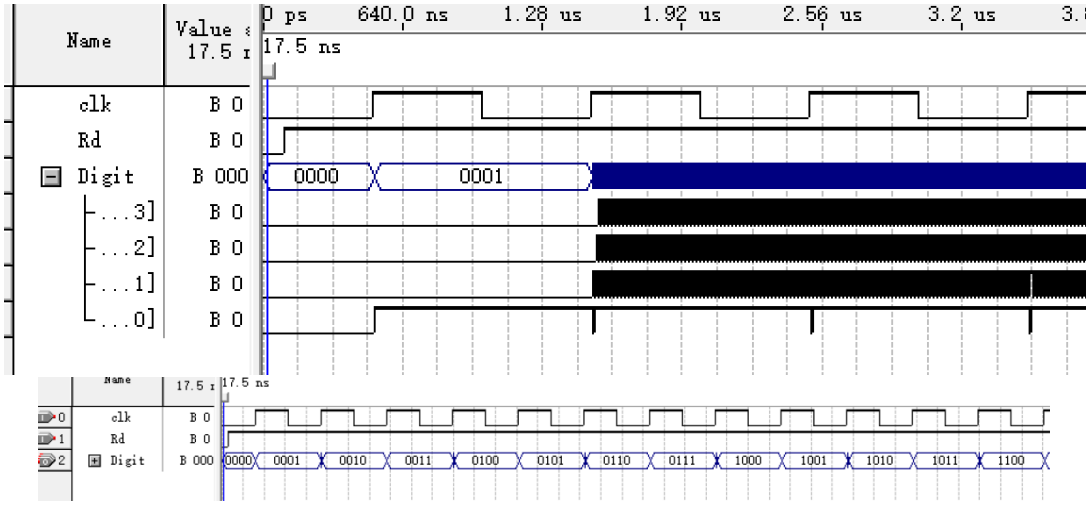

图 2 不例化 D 触发器搭建的 16 位计数器（上）与例化搭建的（下）

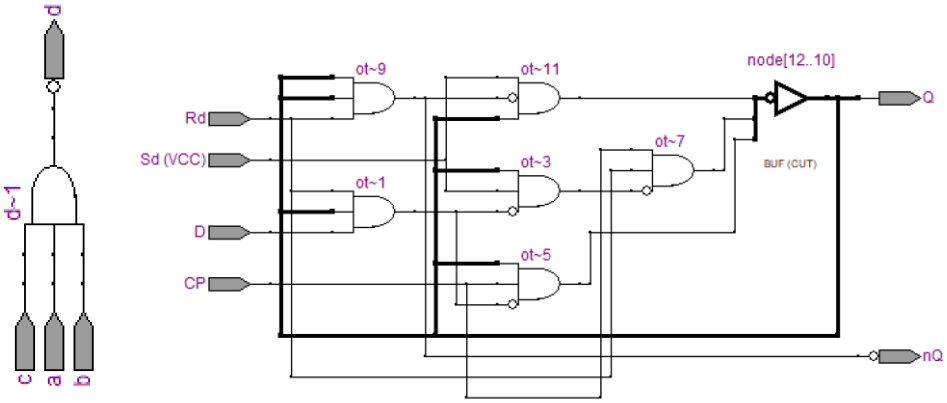

图 3 例化的与非门生成电路（左）和不例化的 D 触发器生成电路（右）

无论是 D 触发器还是计数器，需要通过 Rd（rst）来进行初始化，而不能靠设初值初始化，设初值初始化的方式在仿真中无法显示，如图 4 所示。

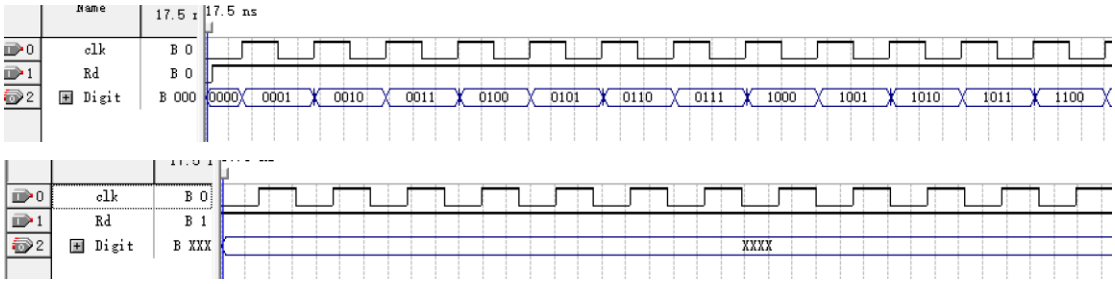

图 4 使用 Rd 初始化（上）与不使用 Rd 初始化（下）

2、利用异步 16+16 搭建异步 60 进制计数器的失败：延时与 D 触发器参数

采用两个 16 进制 D 触发器搭建异步 60 进制计数器失败，仿真结果如图 5 所示，代码如图 6 所示。

经过仔细排查，发现是因为 RdL 与 RdH 的持续时间过短，使得没有完全重置 Rd 就变成 1 了（**Rd 变成 0 的持续时间小于 D 触发器对应参数要求时间**），因此需要将重置的逻辑写在内部，即搭建一个异步 6 进制计数器和异步 10 进制计数器。另外，vhdl 中的 after 关键字是不可综合的，因此在仿真中不会有影响，并不能以此改变仿真结果。

对于使用六与非门 D 触发器 16 进制+16 进制搭建的异步 60 进制计数器，我还采用了不少其他方式，但都没有成功。虽然理论上这种做法可行（使用 event D 触发器也成功实现了这种做法），但与非门 D 触发器与 event D 触发器相比还是有很大劣势，很难以此思路搭建能正常工作的计数器。

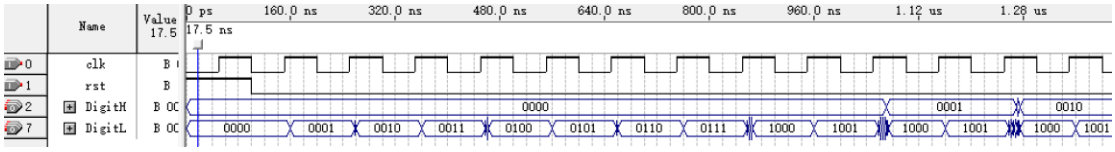

图 5 异步 16+16 的 60 进制计数器仿真结果

```
architecture asyl6 of Cnt60 is
    component Cntl6
        port(
            clk, Rd: in std_logic;
            Digit: out std_logic_vector(3 downto 0)
        );
    end component;
    signal RdL, RdH: std_logic;
    signal QL, QH: std_logic_vector(3 downto 0);
begin
    CL:Cntl6 port map(clk,RdL,QL);
    CH:Cntl6 port map(not QL(3),RdH,QH);
    process(rst, QL)
    begin
        if (rst = '1' or QL = "1010") then
            RdL <= '0';
        else
            RdL <= '1';
        end if;
    end process;
    process(rst, QH)
    begin
        if (rst = '1' or QH = "0110") then
            RdH <= '0';
        else
            RdH <= '1';
        end if;
    end process;
    DigitL <= QL;
    DigitH <= QH;
    RR<=RdL;
end asyl6;
```

图 6 异步 16+16 的 60 进制计数器代码

3、利用异步 6+10 搭建异步 60 进制计数器的搭建：群时延，竞争与冒险

根据上一节的经验，需要把重置信息写在 16 进制计数器内部，再向外传递进位信息。最初，低位计数器（10 进制）的重置 Rd 和进位信息 v 是在输出变为"1010"时为'1'，其余为'0'，这样仍然有上一节的问题，即低位计数器归 0 异常。同时，由于各信号延迟时间不同**（群时延）**，进位信息变化速度快于高位计数器所需保持时间；由于**竞争与冒险使得进位信息 v 带有毛刺**，而 v 是高位计数器的时钟输入：这两点共同导致高位计数器跳动混乱。

之后，Rd 改成在"1010"时为 1，"0000"时为 0，其余不变；v 改成在"1001"时为 1，"0000"时为 0，其余不变，使得 Rd 和 v 可以持续足够长的时间，从而高位和低位计数器都可以正常工作，仿真如图 7 所示，代码如图 8 所示，其中 D 触发器用图 1 右侧（例化法）实现的。

改成异步 6+10 之后，图 1 的两种 D 触发器写法均可以使用，除了电路变化导致延迟变化以外，另一个原因是负载的变化，参考本章第四节。

从图 7 中仍可以看出明显的毛刺，**说明异步电路中的竞争与冒险是比较明显的**。

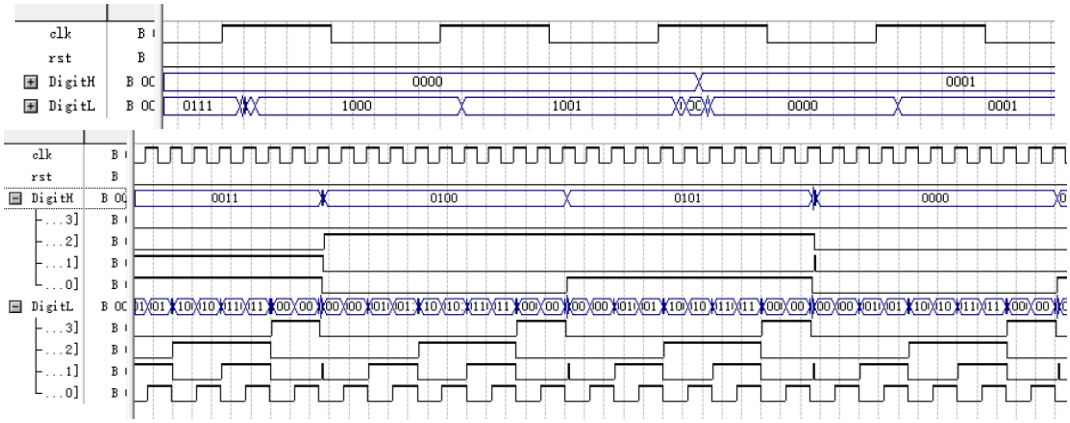

图 7 异步 6+10 的 60 进制计数器仿真结果

```
entity Cnt6 is
    port(
        clk, Rd: in std_logic;
        Digit: out std_logic_vector(3 downto 0)
    );
end Cnt6;
entity Cnt10 is
    port(
        clk, Rd: in std_logic;
        v: out std_logic
        Digit: out std_logic_vector(3 downto 0)
    );
end Cnt10;
entity Cnt60 is
    port(
        clk, rst: in std_logic;
        DigitH, DigitL: out std_logic_vector(3 downto 0)
    );
end Cnt60;

  architecture asyl6 of Cnt60 is
      component Cnt10
          port(
              clk, Rd: in std_logic;
              v: out std_logic;
              Digit: out std_logic_vector(3 downto 0)
          );
      end component;
      component Cnt6
          port(
              clk, Rd: in std_logic;
              Digit: out std_logic_vector(3 downto 0)
          );
      end component;
      signal v, Rd: std_logic;
      signal QL, QH: std_logic_vector(3 downto 0);
  begin
      CL:Cnt10 port map(clk,Rd,v,QL);
      CH:Cnt6 port map(v,Rd,QH);
      Rd <= not rst;
      DigitL <= QL;
      DigitH <= QH;
  end asyl6;
```

```
architecture asyl0 of Cntl0 is
    component DTrigger
        port(
            D, CP, Rd, Sd: in std_logic;
            Q: out std_logic := '0';
            nQ: out std_logic := '1'
        );
    end component;
    signal RRd: std_logic;
    signal nQ, Q: std_logic_vector(3 downto 0);
begin
    DL0:DTrigger port map(not Q(0),clk,RRd,'1',Q(0),nQ(0));
    DL1:DTrigger port map(not Q(1),not Q(0),RRd,'1',Q(1),nQ(1));
    DL2:DTrigger port map(not Q(2),not Q(1),RRd,'1',Q(2),nQ(2));
    DL3:DTrigger port map(not Q(3),not Q(2),RRd,'1',Q(3),nQ(3));
    Process(Rd,Q)
    begin
        if (Rd='0'or Q="1010")then
            RRd<='0';
        elsif (Q="0000") then
            RRd<='1';
        end if;
    end process;
    process(Q)
    begin
        if (Q="1010") then
            Digit<="0000";
            v<='1';
        elsif (Q="0000") then
            Digit<="0000";
            v<='0';
        else
            Digit<=Q;
        end if;
    end process;
end asyl0;

architecture asy6 of Cnt6 is
    component DTrigger
        port(
            D, CP, Rd, Sd: in std_logic;
            Q: out std_logic := '0';
            nQ: out std_logic := '1'
        );
    end component;
    signal RRd: std_logic;
    signal nQ, Q: std_logic_vector(3 downto 0);
begin
    DL0:DTrigger port map(not Q(0),clk,RRd,'1',Q(0),nQ(0));
    DL1:DTrigger port map(not Q(1),not Q(0),RRd,'1',Q(1),nQ(1));
    DL2:DTrigger port map(not Q(2),not Q(1),RRd,'1',Q(2),nQ(2));
    Q(3)<='0';
    Process(Rd,Q)
    begin
        if (Rd='0'or Q="0110")then
            RRd<='0';
        elsif (Q="0000") then
            RRd<='1';
        end if;
    end process;
    process(Q)
    begin
        if (Q="0110") then
            Digit<="0000";
        else
            Digit<=Q;
        end if;
    end process;
end asy6;
```

图 8 异步 6+10 的 60 进制计数器代码

4、同步 60 进制计数器的搭建：负载

十进制计数器的同步时序电路的设计过程在数字逻辑设计课程中已有详细说明，六进制计数器的同步时序电路的设计也较为简单，因此不再赘述。将两者组合成 60 进制计数器的常见思路为，在低位额外输出一个进位信息 v，

但是，仅仅在低位添加一个进位信息 v 后，便出现了异常情况，再去掉这个进位信息输出后则又恢复了正常。在仔细排查以及与同学交流后，认定应该是**负载**的问题（因为相当于**输出对输入产生了影响**）。于是将进位信息在外部用 D 触发器实现，仍有异常，关键代码与仿真结果如图 9 所示，低位在 0111 之后出现了不断跳动。与一些同学交流后，发现同学们有的是用 not Q 作为$\bar{Q}$使用，有的直接用 D 触发器所给的$\bar{Q}$，我对两者分别进行了尝试，发现从 not Q 改成$\bar{Q}$之后正常，关键代码与仿真结果如图 10 所示。

数字逻辑设计课上，老师曾经指出时序逻辑电路设计中在负载中的考虑：**内部多使用 D 触发器的$\bar{Q}$**，我曾经对此没有在意，在这里才明白这个问题。

同步 60 进制计数器的完整代码如图 11 所示，D 触发器采用的图 1 右侧（例化法）。

从图 10 中可以看出毛刺较图 7 少了一些，**同步时序电路的竞争与冒险问题比异步较少，这是因为异步有群时延问题**。

```
D(0)<=v xor Q(0);
D(1)<=((not v) and Q(1)) or (v and (not Q(2) and (Q(1) xor Q(0))));
D(2)<=((not v) and Q(2)) or (v and ((Q(2) and not Q(0)) or (Q(1) and Q(0))));
Digit(2 downto 0)<=Q;
Digit(3)<='0';

D(0)<=not Q(0);
D(1)<=(not Q(0) xor not Q(1)) and (Q(0) nand Q(3));
D(2)<=((not Q(0)) or (not Q(1))) xnor Q(2);
D(3)<=((not ((not Q(0) or not Q(1) or not Q(2)))) xnor not Q(3)) and (Q(0) nand Q(3));
Digit<=Q;
```

图 9 错误的高位（上）和低位（中）关键代码与仿真结果（下）

```
D(0)<=v xor Q(0);
D(1)<=((not v) and Q(1)) or (v and (nQ(2) and (Q(1) xor Q(0))));
D(2)<=((not v) and Q(2)) or (v and ((Q(2) and nQ(0)) or (Q(1) and Q(0))));
Digit(2 downto 0)<=Q;
Digit(3)<='0';

D(0)<=nQ(0);
D(1)<=(nQ(0) xor nQ(1)) and (Q(0) nand Q(3));
D(2)<=((not Q(0)) or (not Q(1))) xnor Q(2);
D(3)<=((not ((nQ(0) or nQ(1) or nQ(2)))) xnor nQ(3)) and (Q(0) nand Q(3));
Digit<=Q;
```

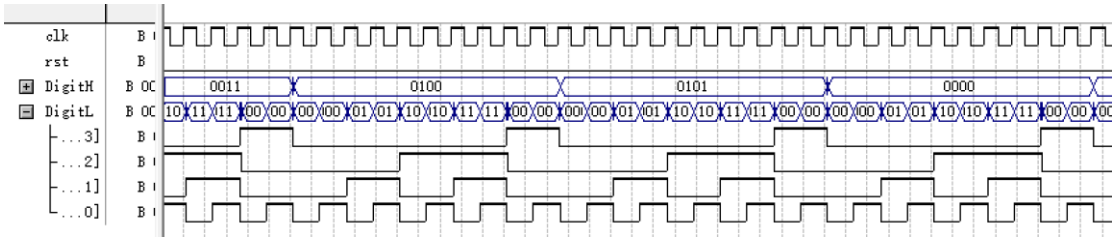

图 10 正确的高位（上）和低位（中）关键代码与仿真结果（下）

```
entity Cnt6 is
    port(
        clk, Rd, v: in std_logic;
        Digit: out std_logic_vector(3 downto 0)
    );
end Cnt6;
architecture asy6 of Cnt6 is
    component DTrigger
        port(
            D, CP, Rd, Sd: in std_logic;
            Q: out std_logic := '0';
            nQ: out std_logic := '1'
        );
    end component;
    signal D, nQ, Q: std_logic_vector(2 downto 0);
    signal tmp1, tmp2: std_logic;
begin
    label0: for i in 0 to 2 generate
        Di:DTrigger port map(D(i),clk,Rd,'1',Q(i),nQ(i));
    end generate label0;
    D(0)<=v xor Q(0);
    D(1)<=((not v) and Q(1)) or (v and (nQ(2) and (Q(1) xor Q(0))));
    D(2)<=((not v) and Q(2)) or (v and ((Q(2) and nQ(0)) or (Q(1) and Q(0))));
    Digit(2 downto 0)<=Q;
    Digit(3)<='0';
end asy6;

entity Cnt10 is
    port(
        clk, Rd: in std_logic;
        Digit: out std_logic_vector(3 downto 0)
    );
end Cnt10;
architecture asy10 of Cnt10 is
    component DTrigger
        port(
            D, CP, Rd, Sd: in std_logic;
            Q: out std_logic := '0';
            nQ: out std_logic := '1'
        );
    end component;
    signal D, nQ, Q: std_logic_vector(3 downto 0);
    signal tmp1, tmp2: std_logic;
begin
    label0: for i in 0 to 3 generate
        Di:DTrigger port map(D(i),clk,Rd,'1',Q(i),nQ(i));
    end generate label0;
    D(0)<=nQ(0);
    D(1)<=(nQ(0) xor nQ(1)) and (Q(0) nand Q(3));
    D(2)<=((not Q(0)) or (not Q(1))) xnor Q(2);
    D(3)<=((not ((nQ(0) or nQ(1) or nQ(2)))) xnor nQ(3)) and (Q(0) nand Q(3));
    Digit<=Q;
end asy10;
```

```
entity Cnt60 is
    port(
        clk, rst: in std_logic;
        DigitL, DigitH: out std_logic_vector(3 downto 0)
    );
end Cnt60;
architecture asy60 of Cnt60 is
    component Cnt10
        port(
            clk, Rd: in std_logic;
            Digit: out std_logic_vector(3 downto 0)
        );
    end component;
    component Cnt6
        port(
            clk, Rd, v: in std_logic;
            Digit: out std_logic_vector(3 downto 0)
        );
    end component;
    component DTrigger
        port(
            D, CP, Rd, Sd: in std_logic;
            Q: out std_logic := '0';
            nQ: out std_logic := '1'
        );
    end component;
    signal D, v, Rd: std_logic;
    signal QL, QH: std_logic_vector(3 downto 0);
begin
    CL:Cnt10 port map(clk,Rd,QL);
    CH:Cnt6 port map(clk,Rd,v,QH);
    Dv:DTrigger port map(D=>D,CP=>clk,Rd=>Rd,Sd=>'1',Q=>v);
    Rd <= not rst;
    D<=(QL(3) and (not QL(2))) and ((not QL(1)) and (not QL(0)));
    DigitL <= QL;
    DigitH <= QH;
end asy60;
```

图 11 同步 60 进制计数器代码

5、秒表的搭建：D 触发器参数 th

首先实现 60 进制的译码工作，在 Cnt 内核上加入“点亮数字人生”中的译码装置即可，如图 12 所示。再将分频计、暂停与清零功能加入即可，如图 13 所示。

当时分频计设计时出现了一个问题，错误的关键代码如图 14 所示，错误的现象为 59 的下一秒变成 20 而不是 00。错误的原因在于，**时钟在 clk 一次跳动后就变成 0，高位迅速变为低位，使高位保持时间小于 th，因而产生错误**。即，错误的计数器获得了一个占空比过低的分频信号，并非我们所需的分频信号，改进为图 13 后即解决了这个问题。

```
entity Counter is
    port(
        clk, rst: in std_logic;
        H, L: out std_logic_vector(6 downto 0)
    );
end Counter;

architecture Count of Counter is
    component Digit
        port(
            Digit4: in std_logic_vector(3 downto 0);
            Light6: out std_logic_vector(6 downto 0)
        );
    end component;
    component Cnt60
        port(
            clk, rst: in std_logic;
            DigitH, DigitL: out std_logic_vector(3 downto 0)
        );
    end component;
    signal DigitH, DigitL: std_logic_vector(3 downto 0);
begin
    cnt: Cnt60 port map(clk,rst,DigitH,DigitL);
    digL: Digit port map(DigitH,H);
    digH: Digit port map(DigitL,L);
end Count;
```

```
entity Digit is
    port(
        Digit4: in std_logic_vector(3 downto 0);
        Light6: out std_logic_vector(6 downto 0)
    );--输入8421码，输出数码管码
end Digit;

architecture Lighting of Digit is
begin
    process(Digit4)--复用点亮数字人生代码
    begin
        case Digit4 is
            when "0000"=>Light6<="1111110";
            when "0001"=>Light6<="0110000";
            when "0010"=>Light6<="1101101";
            when "0011"=>Light6<="1111001";
            when "0100"=>Light6<="0110011";
            when "0101"=>Light6<="1011011";
            when "0110"=>Light6<="0011111";
            when "0111"=>Light6<="1110000";
            when "1000"=>Light6<="1111111";
            when "1001"=>Light6<="1110011";
            when "1010"=>Light6<="1110111";
            when "1011"=>Light6<="0011111";
            when "1100"=>Light6<="1001110";
            when "1101"=>Light6<="0111101";
            when "1110"=>Light6<="1001111";
            when "1111"=>Light6<="1000111";
            when others=>Light6<="0000000";
        end case;
    end process;
end Lighting;
```

图 12 带译码的 60 进制计数器代码

```
entity Stopwatch is
    port(
    --clk为自动时钟，pau为暂停按键（用rst接口）
    --rst='1'时异步清零
        clk, rst, pau: in std_logic;
        H, L: out std_logic_vector(6 downto 0)
    );
end Stopwatch;

architecture watch of Stopwatch is
    component Counter
        port(
            clk, rst: in std_logic;
            H, L: out std_logic_vector(6 downto 0)
        );
    end component;
    --pause='1'表示暂停
    signal clk_in: std_logic := '0';
    signal cnt: integer := 0;
begin
    --Counter作为内核，外部实现pause、rst和时钟
    Count:Counter port map(clk=>clk_in,rst=>rst,H=>H,L=>L);
    --clk_in的实现
    process(pau, clk, rst)
    begin
        if (rst='1') then
            cnt<=0;
            clk_in<='0';
        elsif (clk'event and clk='1' and pau='0') then
            if (cnt = 1000000) then
                cnt<=0;
                clk_in<='1';
            elsif (cnt = 500000) then
                cnt<=cnt+1;
                clk_in<='0';
            else
                cnt<=cnt+1;
            end if;
        end if;
    end process;
end watch;
```

图 13 秒表代码

```
elsif (clk'event and clk='1' and pau='0') then
    if (cnt = 1000000) then
        cnt<=0;
        clk_in<='1';
    else
        cnt<=cnt+1;
        clk_in<='0';
    end if;
end if;
```

图 14 错误分频计的关键代码

6、六与非门 D 触发器总结

6.1 元件例化会改变 quartus 综合方式，从而可能使得延迟增加；

6.2 D 触发器的需要 D、Rd、CP 等参数有一定的持续时间，受此影响，可能需要：

6.2.1 异步计数器设计时，清零需要在 16 进制计数器内实现，而不能在计数器外；

6.2.2 异步计数器设计时，进位信息与清零信息需要再完全清零后再恢复；

6.2.3 制造秒表的分频计时，应该让占空比=50%，而非 1/1M；

6.3 由于负载问题，D 触发器要尽量使用$\bar{Q}$，而非 not Q，这在同步计数器中更明显；

6.4 由于群时延和竞争与冒险问题，同步计数器与异步计数器均会有毛刺现象，但异步计数器更明显，这是因为异步有群时延问题；

6.5 D 触发器代码如图 1 右侧所示，异步计数器代码如图 8 所示，同步计数器代码如图 11 所示，译码功能如图 12 所示，秒表功能如图 13 所示。

二、event 实现的 D 触发器同步与异步计数器

将 D 触发器代码改成如图 15 所示的代码后，其余与第一章相同，得到了正确的结果，如图 16-17 所示。将其与图 7 和图 10 对比，可以发现 event 实现的 D 触发器产生的竞争与冒险（毛刺）远远小于六与非门实验的 D 触发器，所以不要手动搭 D 触发器了。

将图 16 与图 17 对比，可以发现同步电路几乎没有竞争与冒险（毛刺），而异步时序电路有较明显的竞争与冒险（毛刺）。**同步时序电路的优势在于时序性能优越，竞争与冒险情况少；异步时序电路的优势在于设计方便。**

```
entity DTrigger is
    port(
        D, CP, Rd, Sd: in std_logic;
        Q: out std_logic := '0';
        nQ: out std_logic := '1'
    );
end DTrigger;

architecture T of DTrigger is
begin
    process(CP, Rd, Sd)
    begin
        if (Rd = '0') then
            Q<='0';
            nQ<='1';
        elsif (Sd = '0') then
            Q<='1';
            nQ<='0';
        elsif (CP'event and CP = '1') then
            Q<=D;
            nQ<=not D;
        end if;
    end process;
end T;
```

图 15 event 实现的 D 触发器

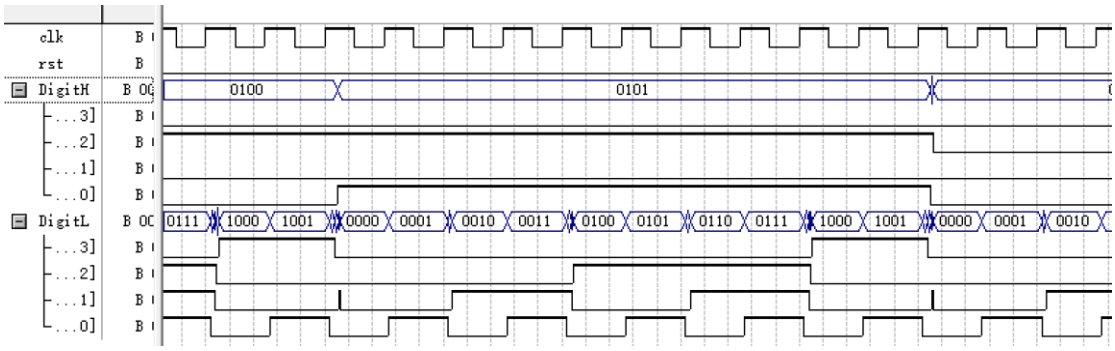

图 16 异步计数器仿真结果

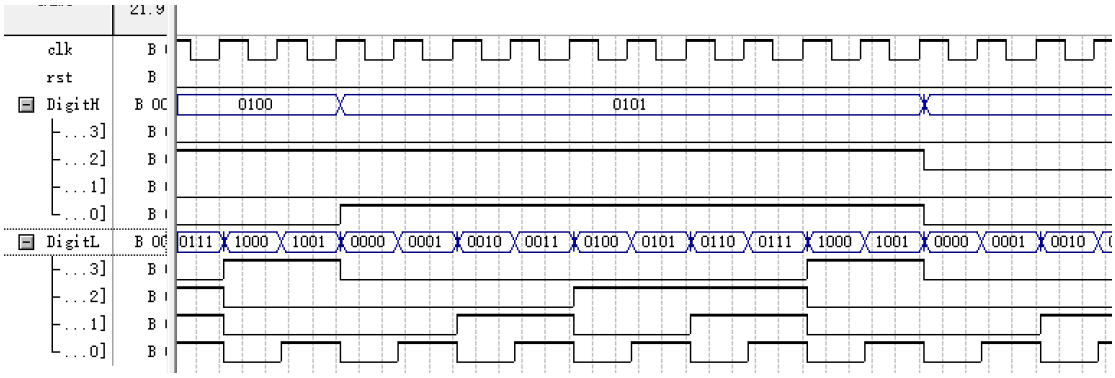

图 17 同步计数器仿真结果

三、完整代码、仿真结果（以同步 event D 为例）

将同步 event D 的代码（图 15、11、12、13）总结，得到完整代码如图 18 所示，仿真结果如图 19-21 所示。

```
--DTrigger.vhd
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

entity DTrigger is
    port(
        D, CP, Rd, Sd: in std_logic;
        Q: out std_logic := '0';
        nQ: out std_logic := '1'
    );
end DTrigger;

architecture T of DTrigger is
begin
    process(CP, Rd, Sd)
    begin
        if (Rd = '0') then
            Q<='0';
            nQ<='1';
        elsif (Sd = '0') then
            Q<='1';
            nQ<='0';
        elsif (CP'event and CP = '1') then
            Q<=D;
            nQ<=not D;
        end if;
    end process;
end T;
```

```
--Cnt10.vhd
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Cnt10 is
    port(
        clk, Rd: in std_logic;
        Digit: out std_logic_vector(3 downto 0)
    );
end Cnt10;
architecture asy10 of Cnt10 is
    component DTrigger
        port(
            D, CP, Rd, Sd: in std_logic;
            Q: out std_logic := '0';
            nQ: out std_logic := '1'
        );
    end component;
    signal D, nQ, Q: std_logic_vector(3 downto 0);
    signal tmp1, tmp2: std_logic;
begin
    label0: for i in 0 to 3 generate
        Di:DTrigger port map(D(i),clk,Rd,'1',Q(i),nQ(i));
    end generate label0;
    D(0)<=nQ(0);
    D(1)<=(nQ(0) xor nQ(1)) and (Q(0) nand Q(3));
    D(2)<=((not Q(0)) or (not Q(1))) xnor Q(2);
    D(3)<=((not ((nQ(0) or nQ(1) or nQ(2)))) xnor nQ(3)) and (Q(0) nand Q(3));
    Digit<=Q;
end asy10;
--Cnt6.vhd
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Cnt6 is
    port(
        clk, Rd, v: in std_logic;
        Digit: out std_logic_vector(3 downto 0)
    );
end Cnt6;
architecture asy6 of Cnt6 is
    component DTrigger
        port(
            D, CP, Rd, Sd: in std_logic;
            Q: out std_logic := '0';
            nQ: out std_logic := '1'
        );
    end component;
    signal D, nQ, Q: std_logic_vector(2 downto 0);
    signal tmp1, tmp2: std_logic;
begin
    label0: for i in 0 to 2 generate
        Di:DTrigger port map(D(i),clk,Rd,'1',Q(i),nQ(i));
    end generate label0;
    D(0)<=v xor Q(0);
    D(1)<=((not v) and Q(1)) or (v and (nQ(2) and (Q(1) xor Q(0))));
    D(2)<=((not v) and Q(2)) or (v and ((Q(2) and nQ(0)) or (Q(1) and Q(0))));
    Digit(2 downto 0)<=Q;
    Digit(3)<='0';
end asy6;
```

```
--Cnt60.vhd
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Cnt60 is
    port(
        clk, rst: in std_logic;
        DigitL, DigitH: out std_logic_vector(3 downto 0)
    );
end Cnt60;
architecture asy60 of Cnt60 is
    component Cnt10
        port(
            clk, Rd: in std_logic;
            Digit: out std_logic_vector(3 downto 0)
        );
    end component;
    component Cnt6
        port(
            clk, Rd, v: in std_logic;
            Digit: out std_logic_vector(3 downto 0)
        );
    end component;
    component DTrigger
        port(
            D, CP, Rd, Sd: in std_logic;
            Q: out std_logic := '0';
            nQ: out std_logic := '1'
        );
    end component;
    signal D, v, Rd: std_logic;
    signal QL, QH: std_logic_vector(3 downto 0);
begin
    CL:Cnt10 port map(clk,Rd,QL);
    CH:Cnt6 port map(clk,Rd,v,QH);
    Dv:DTrigger port map(D=>D,CP=>clk,Rd=>Rd,Sd=>'1',Q=>v);
    Rd <= not rst;
    D<=(QL(3) and (not QL(2))) and ((not QL(1)) and (not QL(0)));
    DigitL <= QL;
    DigitH <= QH;
end asy60;
```

```
--Stopwatch.vhd
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Stopwatch is
    port(
    --clk为自动时钟，pau为暂停按键（用rst接口）
    --rst='1'时异步清零
        clk, rst, pau: in std_logic;
        H, L: out std_logic_vector(6 downto 0)
    );
end Stopwatch;
architecture watch of Stopwatch is
    component Counter
        port(
            clk, rst: in std_logic;
            H, L: out std_logic_vector(6 downto 0)
        );
    end component;
    --pause='1'表示暂停
    signal clk_in: std_logic := '0';
    signal cnt: integer := 0;
begin
    --Counter作为内核，外部实现pause、rst和时钟
    Count:Counter port map(clk=>clk_in,rst=>rst,H=>H,L=>L);
    --clk_in的实现
    process(pau, clk, rst)
    begin
        if (rst='1') then
            cnt<=0;
            clk_in<='0';
        elsif (clk'event and clk='1' and pau='0') then
            if (cnt = 1000000) then
                cnt<=0;
                clk_in<='1';
            elsif (cnt = 500000) then
                cnt<=cnt+1;
                clk_in<='0';
            else
                cnt<=cnt+1;
            end if;
        end if;
    end process;
end watch;
```

```
--Counter.vhd
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Counter is
    port(
        clk, rst: in std_logic;
        H, L: out std_logic_vector(6 downto 0)
    );
end Counter;

architecture Count of Counter is
    component Digit
        port(
            Digit4: in std_logic_vector(3 downto 0);
            Light6: out std_logic_vector(6 downto 0)
        );
    end component;
    component Cnt60
        port(
            clk, rst: in std_logic;
            DigitH, DigitL: out std_logic_vector(3 downto 0)
        );
    end component;
    signal DigitH, DigitL: std_logic_vector(3 downto 0);
begin
    cnt: Cnt60 port map(clk,rst,DigitH,DigitL);
    digL: Digit port map(DigitH,H);
    digH: Digit port map(DigitL,L);
end Count;
```

图 18 完整代码

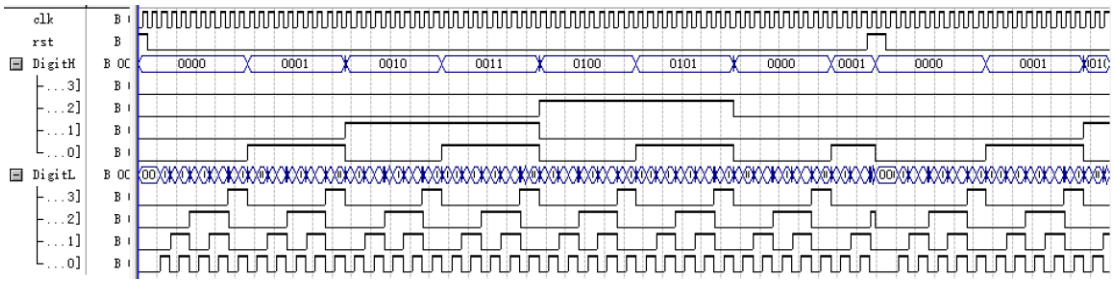

图 19 核心功能的计数、复位仿真

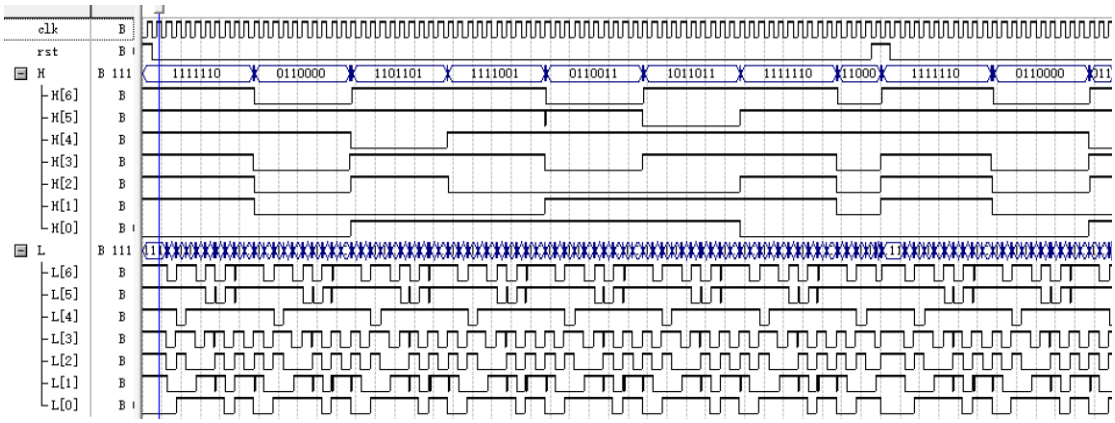

图 20 计数器的计数、复位仿真

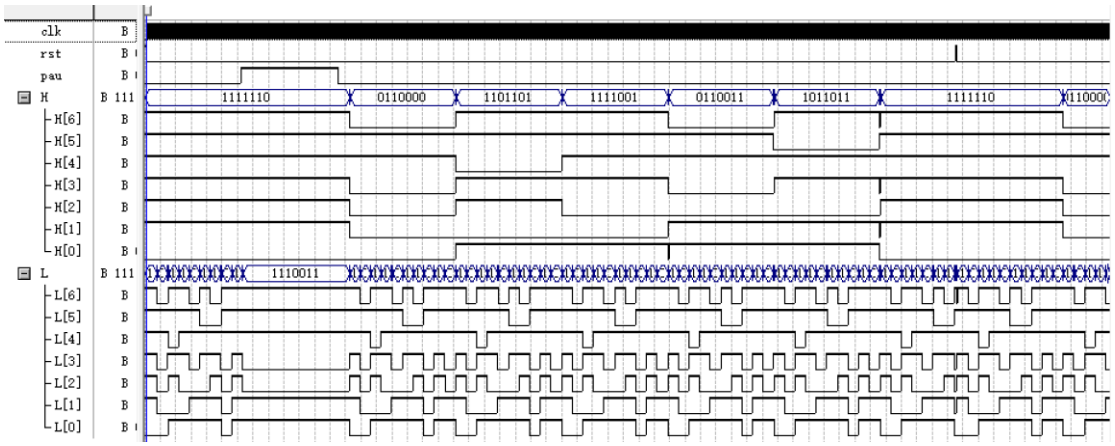

图 21 秒表的计数、暂停、复位仿真（转化了 clk 以缩短仿真时间）

四、收获

通过本次实验，我加深了对以下内容的理解：

1、与非门与 D 触发器的结构；

2、负载问题以及相应的处理办法；

3、群时延、竞争与冒险问题产生的毛刺以及相应的处理办法；

4、同步时序电路的分析与设计。

通过本次实验，我熟悉了以下 quartus 相关内容的使用与理解：

1、熟悉了元件例化的方法；

2、对于元件例化会影响 quartus 优化有了初步的认识；

3、熟悉了仿真的各项内容：

3.1 熟悉了仿真结束时间的设置；

3.2 明白了仿真结束时间一般不能超过 100us，否则仿真需要花费非常久的时间；

3.3 熟悉了时钟在仿真中的设置；

3.4 熟悉了方便显示输出端口的方法，熟悉了显示中间信号的方法。

五、可能的扩展

对于同步时序电路而言，JK 触发器会比 D 触发器实现六进制与十进制计数器更简单，理论上毛刺现象会更加不明显。