同濟大學

可编程逻辑器件大作业报告

学院(系)	电子与信息工程学院
专　　业	电子信息工程
年　　级	2020 级
姓　　名	黄锴康
学　　号	1951706
实验课程	可编程逻辑器件
任课教师	林　林
项目名称	顶层为原理图的模 16 减法计数器输出值的显示
日期	2023.5.6

目录

装

订

线

一、实验内容

本次实验完成一个能显示模 16 减法计数器输出值的工程，即内置一个最大值为 16 的减法器，其中包含分频器、模 16 计数器、解码器、数据转换器和二选一模块等电路，最后通过数码显示器显示出来。

需要完成各个模块的 VHDL 描述及仿真，并完成该工程在硬件平台 NEXYS4 上演示。

二、实验步骤及结果演示

如实验内容所述，本实验主要分为：VHDL 代码编写—模块仿真与验证—电路综合—烧录并观察现象。本工程的顶层原理图如图 2.1 所示，该图同时也是最后顶层 Schmatic 的原理图。

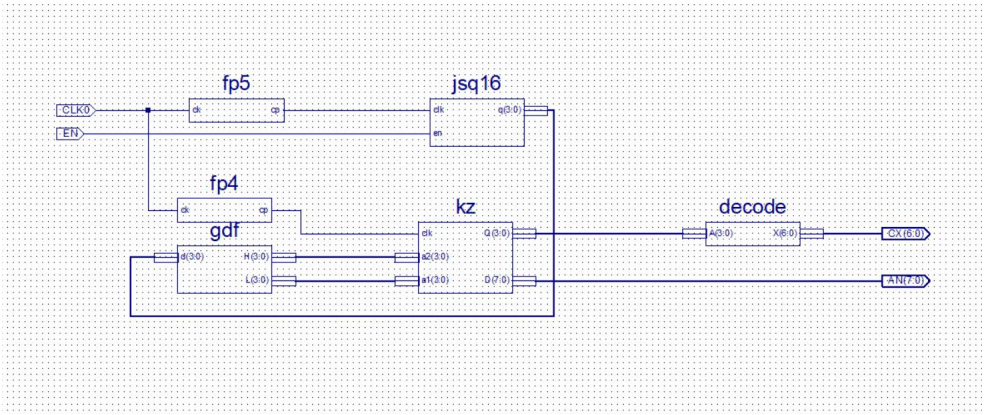

图 2.1 顶层原理图

观察顶层设计图可以理解整个工程的工作原理，首先本工程有两个输入端口(CLK0、EN)，CLK0 作为系统时钟，作为模块中的时钟源，EN 控制模 16 计数器。两个输出端口(CX(6:0)、AN(7:0))，CX 作为单个数码管二极管的控制信号，AN 控制数码管的使能。在理解电路整体原理后，即可分模块编写 VHDL 代码了。

- 分频器设计（fP5、fP4）

通过设计通用分频器 fpV，分频数由 Generic 确定，初始值取为 16。通过以上设计可以实现代码的复用。

fpV

通用分频器

```
entity fpV is
    generic (
        N : integer := 16
    );
    port (ck : in std_logic;
        cp : out std_logic
    );
end entity fpV;

architecture Behavioral of fpV is
begin
process (ck)
variable a : integer range 0 to N
    begin
```

```
if (ck'event and ck='1')    then
            if a=N -1 then
                a:=0;
             else
                a:=a+1;
             end if;
             if a< N /2 then
                cp<='1';
            else
                cp<='0';
            end if;
       end if;
     end process;
end architecture Behavioral;
```

通过以上代码实现 fpV，在实现 fp5 和 fp4 时，只需要将 fpV 作为 component 调用既可，同时也可以把这个过程看做对 fpV 的实例化，fp5 和 fp4 的代码如下：

Fp5

通用分频器

```
entity fp5 is
      Port ( ck : in    STD_LOGIC;
                cp : out    STD_LOGIC);
end fp5;
architecture struct of fp5 is
component fpV
generic(n:integer);
port(ck:in std_logic;
      cp:out std_logic);
end component;
begin
u1:fpV generic map(100000000) port map(ck,cp);--通过传递 Generic 值实现 100M 分频
end struct;
```

Fp4

通用分频器

```
entity fp4 is
      Port ( ck : in    STD_LOGIC;
                cp : out    STD_LOGIC);
end fp4;
architecture struct of fp4 is
component fpV
generic(n:integer);
port(ck:in std_logic;
      cp:out std_logic);
end component;
begin
u2:fpV generic map(100000) port map(ck,cp);--通过传递 Generic 值实现 100K 分频
end struct;
```

在本项目中时钟周期为 100MHz，100M 分频后时钟周期为 1s,而 100K 分频 1ms 因此两个分频器输出的时钟周期将直接影响后续元件的工作频率。

下面将对两个分频器进行仿真，图 2.2 为 fp5 的仿真，运行时间为 1.6s，观察以下波形可以发现，分频器将时钟周期分为 1s，并且在 0.5s 时发生跳变，结果符合预期。

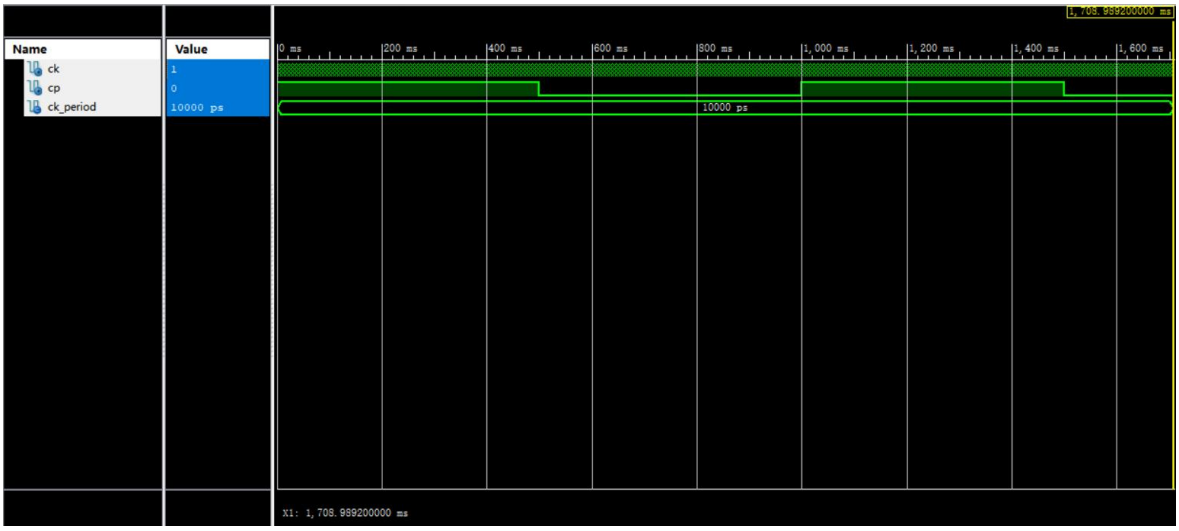

图 2.2 fp5 仿真波形

同样图 2.3 为 fp4 的仿真波形，每隔 500us 变换时钟值，从而实现 1ms 时钟周期的分频。

图 2.3 fp4 仿真波形

- jsq16（模 16 计数器）

模 16 计数器受 fp5 分频后的时钟 cp 控制，每当检测到时钟上升沿，计数器计数值加 1，直到数过 16 个时钟，则重新计时。由于 fp5 的时钟周期是 1s，因此该计数器也可以理解为 16s 计时器。同时 jsq16 存在一个异步使能输入端，只有在使能信号为高电平时，jsq16 才开始计数。

Jsq16

模 16 计数器

```
entity jsq16 is
    Port ( clk : in    STD_LOGIC;
           en : in    STD_LOGIC;
           q : buffer    STD_LOGIC_VECTOR (3 downto 0));
end jsq16;

architecture Behavioral of jsq16 is
begin
```

```
process (clk,en)
begin
      if en='0' then
        q<="0000";
      elsif clk='1' and clk'event then
            if q="1111" then
                  q<="0000";
            else
                  q<=q+1;
            end if;
   end if;
end process;
end Behavioral;
```

通过 testbench 对 jsq16 进行测试，把输出数字以 16 进制显示，结果如图 2.4 所示，在从 0 计数到 15 后，计数器归零。同时，在使能信号为低电平时，jsq16 不会进行计数，功能如预测所示。

图 2.4 jsq16 仿真波形

- gdf（数据转换器模块）

gdf 的功能是将 16 进制码转换为两位 10 进制码。具体来说，在本项目中该模块接收一个 4 位的 16 进制输入信号，将其转换为两个 4 位的 10 进制输出信号。高位输出表示 16 进制码的十位数值，低位输出表示 16 进制码的个位数值。gdf 的代码如下所示。

Gdf

数据转换器

```
entity gdf is
      Port ( d : in    STD_LOGIC_VECTOR (3 downto 0);
                H : out    STD_LOGIC_VECTOR (3 downto 0);
                L : out    STD_LOGIC_VECTOR (3 downto 0));
end gdf;

architecture Behavioral of gdf is
begin
PROCESS(d)
BEGIN
CASE d is
when "0000" => L<="0000"; H <="0000";
when "0001" => L<="0001"; H <="0000";
when "0010" => L<="0010"; H <="0000";
```

```
when "0011" => L<="0011"; H <="0000";
when "0100" => L<="0100"; H <="0000";
when "0101" => L<="0101"; H <="0000";
when "0110" => L<="0110"; H <="0000";
when "0111" => L<="0111" ; H <="0000";
when "1000" => L<="1000"; H <="0000";
when "1001" => L<="1001"; H <="0000";
when "1010" => L<="0000"; H <="0001";
when "1011" => L<="0001"; H <="0001";
when "1100" => L<="0010"; H <="0001";
when "1101" => L<="0011"; H <="0001";
when "1110" => L<="0100"; H <="0001";
when "1111" => L<="0101"; H <="0001";
when others => L<="1111"; H <="1111";
    end case;
end process;
end Behavioral;
```

对于 gdf_tb 的编写，只需要将输入信号从”0000”一直增加到”1111”并观察 H 和 L 的输出即可。再输入 D 为 0-9 时，H 保持为 0，L 和 D 值相同；在 D 为 11-15 时，H 保持为 1，L 和 D 的个位相同，结果符合预期。

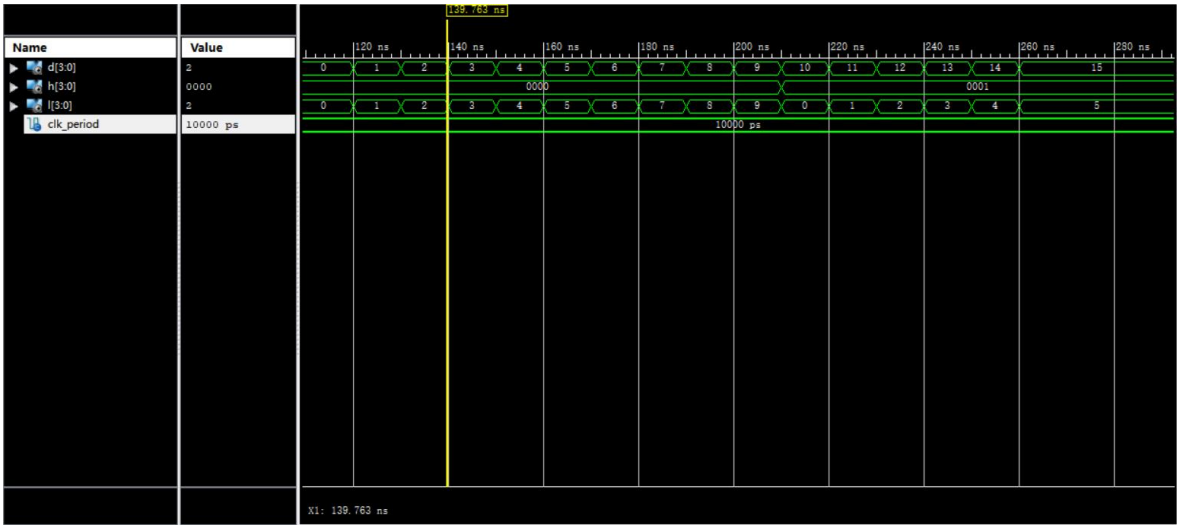

图 2.5 jsq16 仿真波形

- kz（二选一模块）

kz 的功能是实现两位 10 进制数码的轮流输出。该模块具有两个输入向量和两个输出向量。输入向量 Q(3:0)用于轮流输出个位数和十位数的数码值，输出向量 D(7:0)用于控制对应的 8 位数码显示器的位选信号，即指示在数码显示器上显示的位置。通过控制输入向量，kz 模块可以实现两位数码值的轮流显示。

值得注意的是，kz 由 fp4 的时钟信号控制，也就是说 fp4 的工作频率较高，因此在上文提及的 jsq16 模块会以 1s 的周期发生变化，而 kz 可以及时检测到 jsq16 的数据变化，而不会产生时延。

Kz

二选一模块

```
entity kz is
    Port ( clk : in    STD_LOGIC;
           a2 : in    STD_LOGIC_VECTOR (3 downto 0);
           a1 : in    STD_LOGIC_VECTOR (3 downto 0);
           Q : out    STD_LOGIC_VECTOR (3 downto 0);
```

```
                D : out    STD_LOGIC_VECTOR (7 downto 0));
end kz;

architecture Behavioral of kz is

begin
process (clk, a2, a1)
variable b : integer range 0 to 2 ;
begin
      if clk='1' and clk'event then
            if b= 1 then
                  b:=0;
                  Q<=a2;
                  D<="11111101";
             else
                  b:=b+1;
                  Q<=a1;
                  D<="11111110";
            end if;
            --D<="11111111";
      end if;
end process;
end Behavioral;
```

装

- decode（解码器）

decode 模块的功能是将输入的二进制编码转换为对应的七段码输出，用于驱动七段数码管显示特定的数字或字符(（a、b、c、d、e、f、g）。本开发板是共阳极显示管，也就是说控制信号为低电平时，数码管被点亮。

订

在本实验中解码器的真值表如下表所示，注意数码管七段码与控制信号的对应，7 位数字由小到大按 gfedcba 排列，decode 解码器的真值表如表 2.1 所示，代码按真值表逻辑编写即可。

线

表 2.1 decode 真值表

Input (4-bit)	Output (7-bit)
0000	0000001
0001	1001111
0010	0010010
0011	0000110
0100	1001100
0101	0100100
0110	0100000
0111	0001111
1000	0000000
1001	0000100

在完成以上模块的编写后，按照顶层原理图连线即可。

为了将代码正确综合为电路，必须对模块的引脚进行配置，按照开发板的原理图，编写 ucf 引脚配置文件。

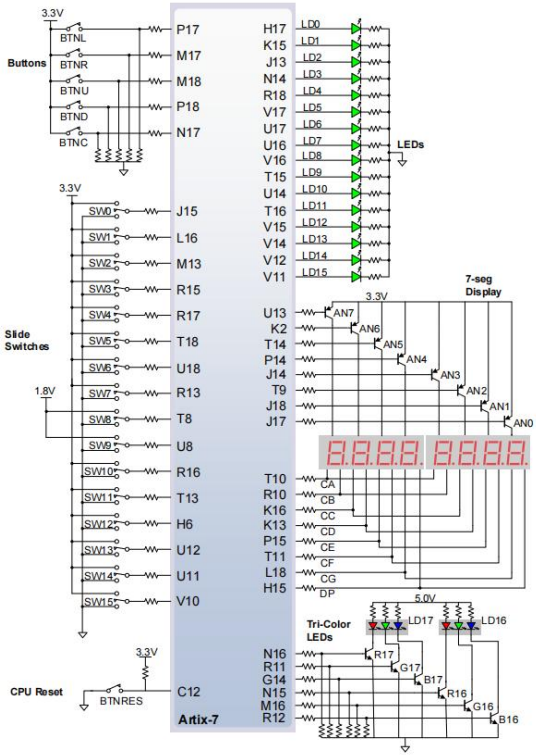

图 2.6 开发板原理图

Ucf
引脚配置文件

```
NET "EN"          LOC = "J15"    | IOSTANDARD =   "LVCMOS33" ;
NET "CLK0"        LOC = "E3"     | IOSTANDARD =   "LVCMOS33" ;

NET "AN<0>"          LOC = "J17" | IOSTANDARD = "LVCMOS33";
NET "AN<1>"          LOC = "J18" | IOSTANDARD = "LVCMOS33";
NET "AN<2>"          LOC = "T9"  | IOSTANDARD = "LVCMOS33";
NET "AN<3>"          LOC = "J14" | IOSTANDARD = "LVCMOS33";
NET "AN<4>"          LOC = "P14" | IOSTANDARD = "LVCMOS33";
NET "AN<5>"          LOC = "T14" | IOSTANDARD = "LVCMOS33";
NET "AN<6>"          LOC = "K2"  | IOSTANDARD = "LVCMOS33";
NET "AN<7>"        LOC = "U13"   | IOSTANDARD = "LVCMOS33";

NET "CX<0>"          LOC = "L18" | IOSTANDARD = "LVCMOS33";
NET "CX<1>"          LOC = "T11" | IOSTANDARD = "LVCMOS33";
NET "CX<2>"          LOC = "P15" | IOSTANDARD = "LVCMOS33";
NET "CX<3>"          LOC = "K13"       | IOSTANDARD = "LVCMOS33";
NET "CX<4>"          LOC = "K16"       | IOSTANDARD = "LVCMOS33";
NET "CX<5>"          LOC = "R10" | IOSTANDARD = "LVCMOS33";
NET "CX<6>"          LOC = "T10" | IOSTANDARD = "LVCMOS33";
```

实际运行效果如图 2.7 所示，结果如预期所示。

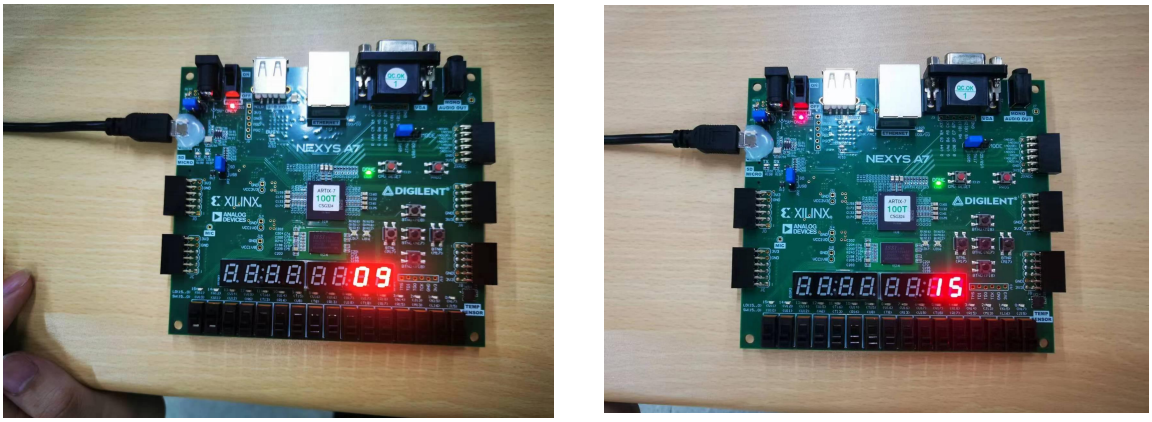

图 2.7 开发板实际运行效果

三、遇到的主要困难及解决对策

- 设计逻辑复杂

项目涉及多个模块的设计和相互调用，其中包括分频器、减法计数器、数据转换器、显示模块等。理解和实现这些模块之间的逻辑关系可能会很复杂，需要仔细思考和分析。

- 时序问题

在设计数字电路时，时序是一个关键问题。特别是涉及到时钟信号和时钟分频器时，需要考虑时钟的频率和延迟等因素，确保电路的正确运行和稳定性。

- 硬件平台兼容性

当将设计的电路实现在硬件平台上时，由于引脚不对应导致完成代码编写后，开发板没有正确显示。这涉及引脚分配、时钟设置、信号接口等方面的调整和配置。

四、感想和体会

在完成这个项目的过程以及可编程逻辑器件课程中，我获得了许多宝贵的经验和体会：

- 通过实际设计和调试数字电路项目，我深刻理解到实践的重要性。仅仅学习理论知识是远远不够的，只有亲自动手去设计、编码和测试，才能真正掌握和应用所学的知识。
- 在项目中，经常会遇到各种问题和困难。通过不断的尝试、调试和寻找解决方案，我培养了解决问题的能力。这包括理解问题的本质、分析原因、寻找解决方案和评估效果。
- 在实践中，很少一次就能完成完美的电路。通过多次迭代和改进，这让我认识到工程设计是一个不断迭代和改进的过程，需要不断学习和改进自己的设计能力。

总的来说，这个项目让我更深入地了解了数字电路设计的实践过程，理解了现代可编程逻辑器件的总体思想。