倒车雷达

一、 总体方案设计

a) 设计分析

i. 倒车雷达实现的原理是： 在汽车倒车时，利用车尾的超声波模块向四周发送超声波，超声波在接触到障碍物时反射信号，被超声波模块所接受，模块根据超声波发送和返回之间的时间差以及超声波传输的速度，就能计算出车体和障碍物之间的实际距离。对于不同的距离，产生不同的声音来提醒驾驶员，使停车更加容易，更加安全。

ii. 倒车雷达的目标： 通过超声波实现对距离的测量；并且根据不同的距离发出不同音调、不同音量、不同频率的声音。

iii. 倒车雷达的相关原理：

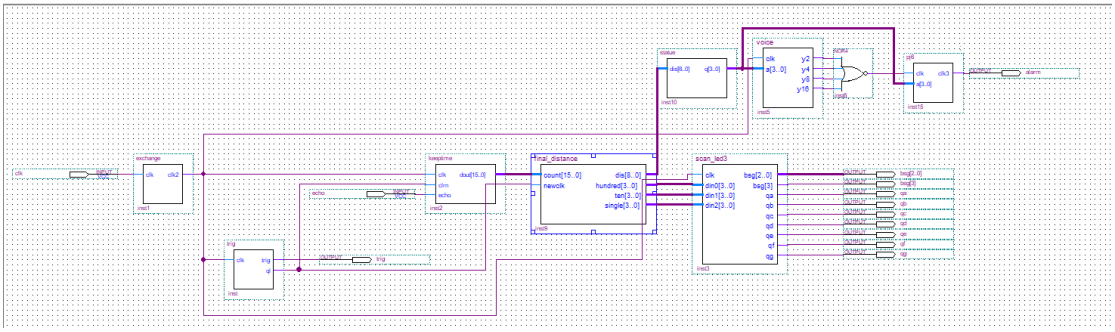
1. 超声波模块有四个引脚，包括 VCC、GND、trig、echo。其中 trig 为触发信号引脚、echo 为回响信号接收引脚
2. 当 trig 引脚上产生一个 10μs 的高电平信号，模块开始工作。它会自动发送 8 个 40kHZ 的方波，并检测是否有信号返回，一旦接收到返回信号，echo 引脚自动变为高电平
3. 超声波模块接收到信号返回，echo 引脚上由高电平自动变成低电平，高电平持续的时间则为超声波传输的时间，根据声波的传输速度，S=(t*340)/2。测量周期设置为 100ms，用来防止发射信号对回想信号的影响。

b) 基本参数：

i. 数码管显示： cm

ii. 时钟频率： 24M hz

二、 顶层文件视图



顶层文件分析： 将时钟信号内部统一为 us 为单位，方便后续部件产生 us 单位的周期信号。

三、 子部件实现

a) 时钟信号统一模块（肖云杰）

模块分析： 因为时钟频率是 24 M hz，即 1 s 有 24 * 10^6 个时钟周期，所以我们把每 24 个自然时钟周期作为一个新的时钟周期 T’，那么 T’就是 1 us。方便后续操作

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  entity exchange is
5       port(clk:in std_logic;
6            clk2:out std_logic);
7  end exchange;
8  architecture arch of exchange is
9      signal count:integer range 0 to 23;
10     begin
11         process(clk)
12         begin
13         if (clk'event and clk='1')then
14             if(count=23)then
15                 count<=0;
16             else
17             count<=count+1;
18             end if;
19         end if;
20         end process;
21         process(clk)
22         begin
23         if(clk'event and clk='1') then
24             if(count=0)then
25                 clk2<='1';
26             elsif (count=12) then clk2 <= '0';
27             end if;
28         end if;
29         end process;
30 end arch;
```

b) 周期性高电平保持时间测量(肖云杰)

模块分析： 将产生的高电平脉冲的保持时间测量出来，并以相应的单位传输给下一个模块

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity keeptime is
port(clk,clrn,echo:in STD_LOGIC;
                dout:out std_logic_vector(15 downto 0));
end keeptime ;

architecture arcl of keeptime  is

signal t:STD_LOGIC_VECTOR(15 downto 0);
signal js:STD_LOGIC_VECTOR(15 downto 0);
begin
   process(echo,clk,clrn,js,t)
   begin
      if clrn='1' then
         js<="0000000000000000";
      else if clk'event and clk='0' then
             if echo='1' then
                  js<=js+1;
             end if;
      end if;
      end if;
      if echo='1' then
         t<="0000000000000000";
      else
         t<=js;
      end if;
      dout<=t;
   end process;
end arcl;
```

c) 周期性高电平信号产生（肖云杰）

模块分析： 在 trig 引脚产生一个大于 10us 的周期性高电平信号，以便可以启用其内模块，完成模块内部的功能。

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4
5  entity trig is
6        port(clk:in std_logic;
7             trig,ql:out std_logic);
8  end entity trig;
9
10 architecture arch of trig is
11 signal count:std_logic_vector(15 downto 0);
12  begin
13  process(clk)
14    begin
15    if(clk'event and clk='1') then
16         if(count="1111111111111111")then
17              count<="0000000000000000";
18              ql<='1';
19         else
20              count<=count+1;
21              ql<='0';
22         end if;
23    end if;
24    end process;
25   process(clk)
26    begin
27    if(clk'event and clk='1') then
28         if(count="0000000000000000")then
29         trig<='1';
30         elsif(count="0000000001111111")then
31         trig<='0';
32
33    end if;
34    end if;
35    end process;
36 end arch;
```

模块 a-c

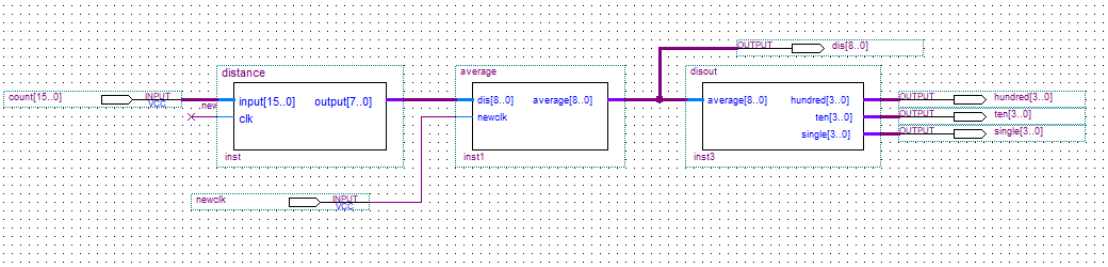
实验总结：

作为倒车雷达的启动前控制模块，是所有模块的核心，统一各部件的 clk 时发生了一些小问题，解决问题的过程中，我了解到了各模块协同工作开发的乐趣的困难，但是成功解决。

d) 距离计算（李星沛）

预期功能：根据 echo 信号高脉冲保持时间计算出超声波模块与障碍物之间的距离

输入输出：



优化：通过打表输出距离和优化平均值模块，进而优化速度。

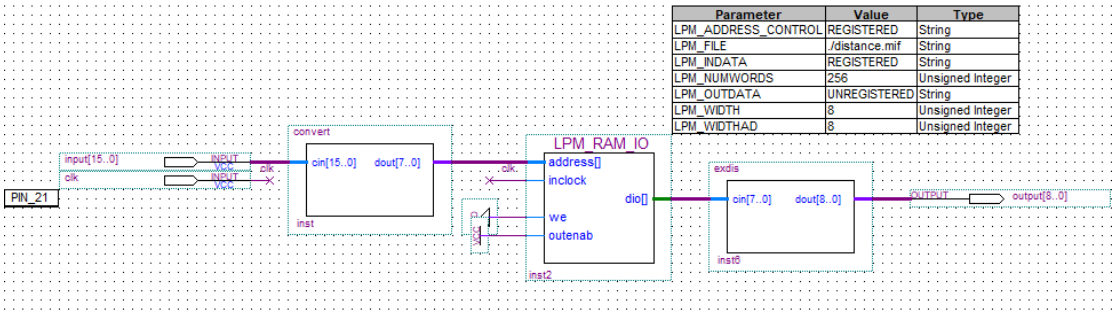
模块具体实现：

1） 实验内容：通过接收 echo 引脚上周期性高脉冲的保持时间计算出距离

2） 设计分析：第二个模块传给我的时间单位为 0.1ms，所以我传出的单位 cm 级；由于结果承线性变化，我选择打表的方式，获得时间即可输出距离。但这里由于是 15 位我认为表不用这么长，则我只取最开始 8 位即可。（距离设置太长了，表不好设置）。

3） 实现功能：根据接收 echo 引脚 shang 周期性的高脉冲的保持时间计算出距离。公式为：测试距离=（高电平时间*声速（340m/s））/2；所以 1S 距离就是 170m。

电路图如下：



Ram 中的初值为

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	00000000	00000010	00000011	00000101	00000111	00001001	00001010	00001100
8	00001110	00010000	00010001	00010011	00010101	00010110	00011000	00011010
16	00011011	00011101	00011111	00100001	00100010	00100100	00100110	00100111
24	00101001	00101011	00101100	00101110	00110000	00110010	00110011	00110101
32	00110111	00111000	00111010	00111100	00111101	00111111	01000001	01000011
40	01000100	01000110	01001000	01001001	01001011	01001101	01001110	01010000
48	01010010	01010100	01010101	01010111	01011001	01011010	01011100	01011110
56	01011111	01100001	01100011	01100101	01100110	01101000	01101010	01101011
64	01101101	01101111	01110000	01110010	01110100	01110110	01110111	01111001
72	01111011	01111100	01111110	10000000	10000001	10000011	10000101	10000111
80	10001000	10001010	10001100	10001101	10001111	10010001	10010010	10010100
88	10010110	10011000	10011001	10011011	10011101	10011110	10100000	10100010
96	10100011	10100101	10100111	10101001	10101010	10101100	10101110	10101111
104	10110001	10110011	10110100	10110110	10111000	10111010	10111011	10111101
112	10111111	11000000	11000010	11000100	11000101	11000111	11001001	11001011
120	11001100	11001110	11010000	11010001	11010011	11010101	11010110	11011000
128	11011010	11011100	11011101	11011111	11100001	11100010	11100100	11100110
136	11100111	11101001	11101011	11101101	11101110	11110000	11110010	11110011
144	11110101	11110111	11111000	11111010	00000000	00000000	00000000	00000000
152	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
160	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
168	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
176	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
184	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
192	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
200	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
208	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
216	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
224	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
232	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
240	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
248	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

通过计算可得，S = 1.7 * t，单位为 cm，故 RAM 中 mif 文件中的距离每次增加 1.7。距离的范围到 0-250cm。

故 RAM 中 mif 文件中的距离每次增加约 1.7。

注意要把头和尾的数据都处理好，这里接入的是一个 16 位的时间信号值。输出也要转化为距离值输出。

转化器如下

```
library ieee;
use ieee.std_logic_1164.all;
use Ieee.std_logic_unsigned.all;
use Ieee.std_logic_arith.all;
entity exdis is
port(
     cin:in std_logic_vector(7 downto 0);
     dout:out integer range 0 to 500
      );
end entity;

architecture bav of exdis is
begin
  dout<=conv_integer(cin);
end;
```

这里设置是传出上限为 500cm 的整数。

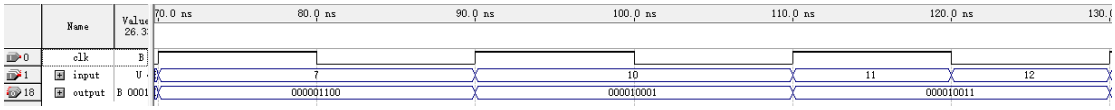
当然，这里还可加上一个计数平均器。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_unsigned.ALL;
ENTITY average IS
    PORT(
        dis: in integer range 0 to 500;
        newclk: in std_logic;
        average: out integer range 0 to 500
        );
END ENTITY average ;
ARCHITECTURE average_arch OF average  IS
signal n:integer range 0 to 10;
signal count:integer range 0 to 12000;
begin
process(newclk)
begin
if(newclk'event and newclk='1') then
  if(n=10) then
     n<=0;average<=count/10;
  elsif(n=0) then n<=n+1;count<=0;--numout<=0;
  else n<=n+1;count<=dis+count;--numout<=1;
  end if;
end if;
end process;
end average_arch;
```

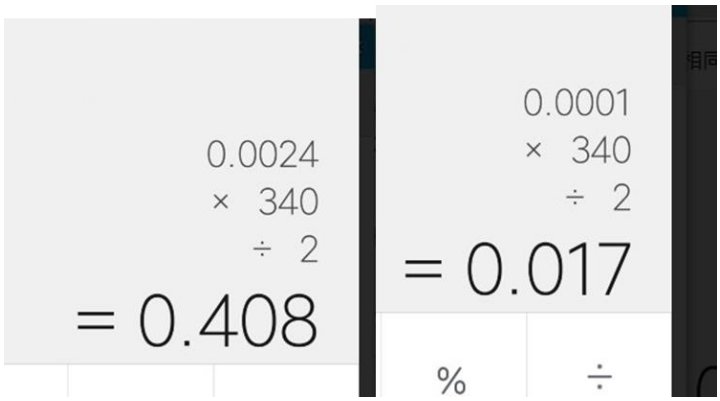
这里和示范代码唯一不一样的就是，这里是取 10 个平均，因为在验收时，能明显地感受到一些延迟，所以这里减少取数平均的个数。

功能验证 仿真图（gridsize：24MHz / endtime：101ms）

EndTime = 1us; GridSize = 10us;

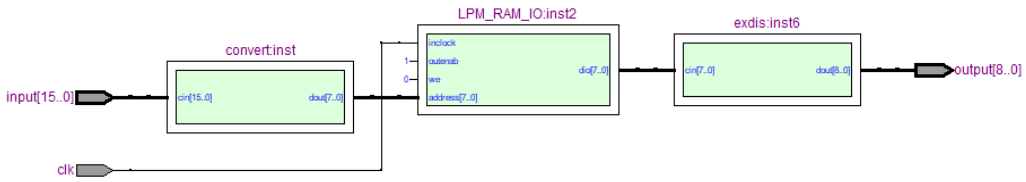


由仿真图可知：当输入时间为 1（单位 0.1ms）时，传出的距离为 2cm；计算得：(0.1*10^(-3)*340)/2 = 1.7 * 10^(-2)；由于输出为整数，所以四舍五入结果后为 2cm。与仿真结果相同。



当时间为 24 时，输出为 41.通过计算公式可得：40.8cm，四舍五入后答案为 41cm，与仿真结果相符。

RTL 视图



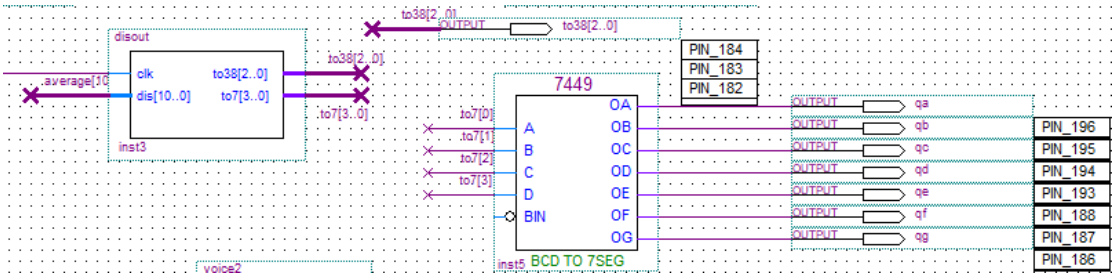
取平均数的那个模块的就不放上来了。

实验总结:经过波形仿真得结果与实际计算相比，答案相吻合。故设计正确。经过此次实验自己也收获了很多，觉得自己在完成工作时要多想

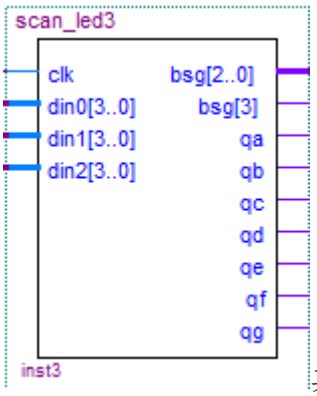
优化:用读表得方式比直接用公式计算获得速度更快和效率更高。且与数码管显示部分的同学联动，优化计算和显示过程。

e) 数码管显示（张馀玎）

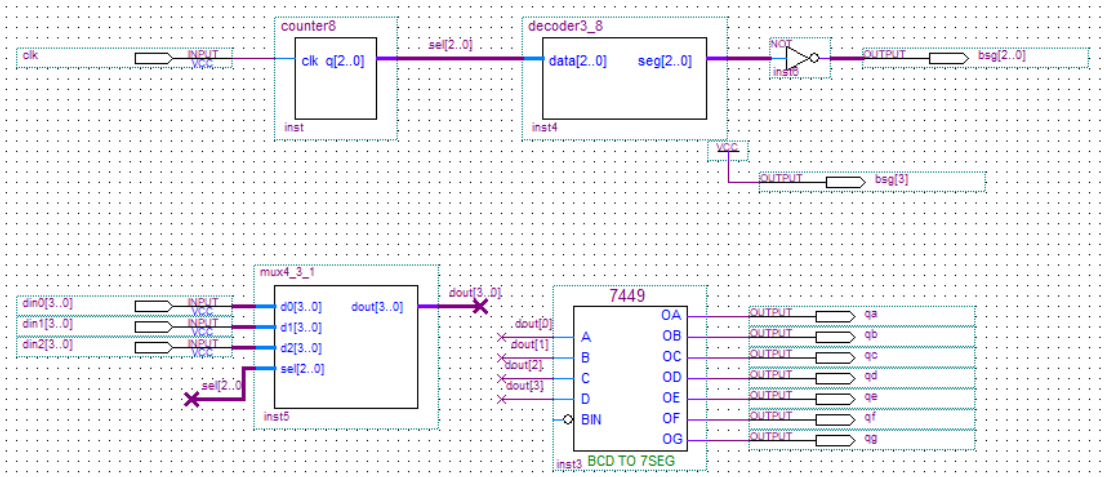
模块分析： 距离显示，即在数码管上显示距离。参照示例工程，该模块所实现的内容为将计算出来的距离通过 7449 和位选使其在数码管上显示出来。对于此方案，其由两个小模块组成，一个模块为将接收到的距离转化为 BCD 码，同时产生数码管的位选信号；另一个模块为将 BCD 码，通过 7449 使其译码显示在数码管上。



模块改进： 我们将该模块接收到的数据直接改为 BCD 码，通过 3 组 BCD 码与一个时钟信号的输入，直接将其转化为段选和位选信号输出，以使系统运算更加高效



其内部结构为：

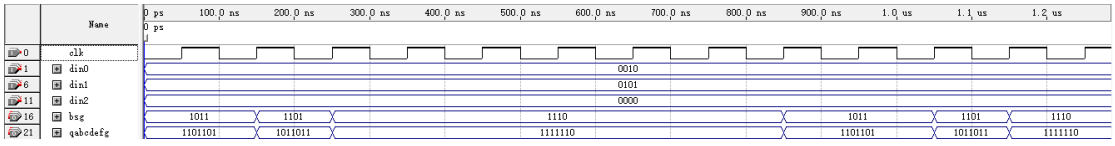


设计方式采用 bdf 而不是 vhdl，性能上有所提升。

验证方法：

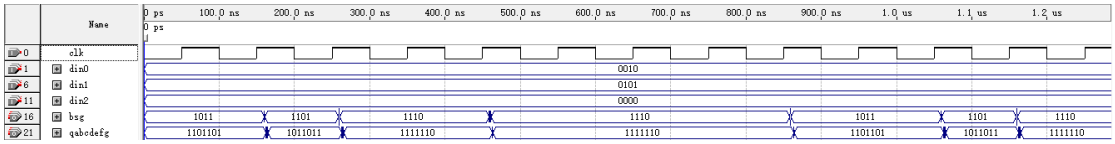
通过 Quartus II 软件的波形图进行验证，先进行功能仿真查看是否能实现相应功能，然后时序仿真观察是否能在存在延时的情况下得到正确结果。仿真结果如下

功能仿真：



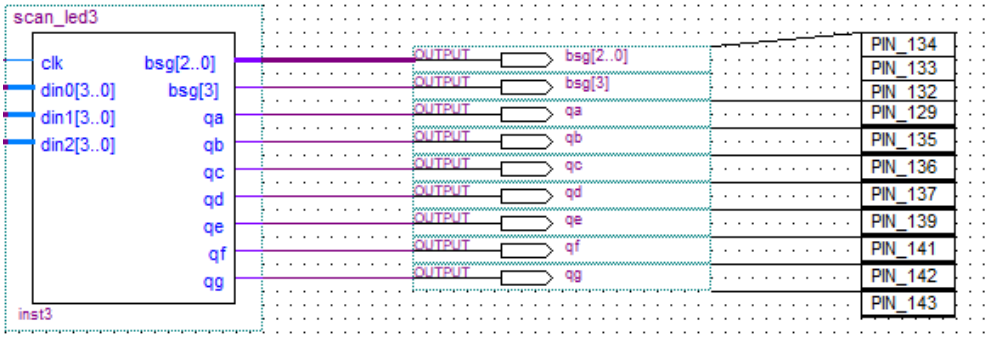
可以看到在 0-850ns，对于每个数码管上的数字，din0-din2 为十进制数 250，在位选信号下，对应的数码管上显示数字，段选信号 q 都能够正确译码，成功显示。

时序仿真：



在存在延时的情况下，仍然能够正常运行，得到正确结果。

管脚分配：



如图所示，将位选与段选接上输出，同时分配引脚。前端的输入与上一模块即距离结果的输出相接，然后全部模块整合运行

实验总结：

倒车雷达的主要作用是在汽车倒车时，利用车尾的超声波模块向四周发送超声波，超声波在接触到障碍物时发射信号，被超声波模块所接受，模块根据超声波发送和返回之间的时间差以及超声波传输的速度，就能计算出车体和障碍物之间的实际距离。对于不同的距离产生不同的声音来提醒驾驶员，使停车更加容易，更加安全。在本次倒车雷达的设计中，我们小组通力合作，成功实现了相应功能。同样的，对于一个完整工程的实现，只凭一人是需要消耗大量时间的。我们在设计过程中，各模块的分工与协调需要完美的商量才促成了本次实验的成功。

通过本学期的各类实验，我从一开始的啥也不会，到现在对计算机硬件技术基础的理解掌握，以及 Quartus II 软件的熟练运用。期间积累的各种知识都让我受益匪浅，我相信通过本课程所教给我的各种知识一定会在以后得到充分的应用！

f) 音频信号发生器(纪韬)

模块功能： 利用实验平台上的基准频率分频产生一个高、中、低三种声音的音频信号，将信号传输到下一个部件当中。

模块原理： 实验平台的蜂鸣器上根据输入不同频率的方波信号就可以听到不同的声音，根据基准频率分成不同的音频系数，进而得到不同频率的方波信号；所以•要将方波型号转化成对应的音频信号。这个就是音频信号发生器的主要功能

模块实现：设计目的和设计要求是能根据传进来的距离信号选择不同的音频和音量控制信号。设计一个 8 位的输入表示传进来的距离信号，根据距离信号的大小分成 6 个范围，每个范围对应一个音频控制信号和一个音量控制信号，然后将控制信号输出。

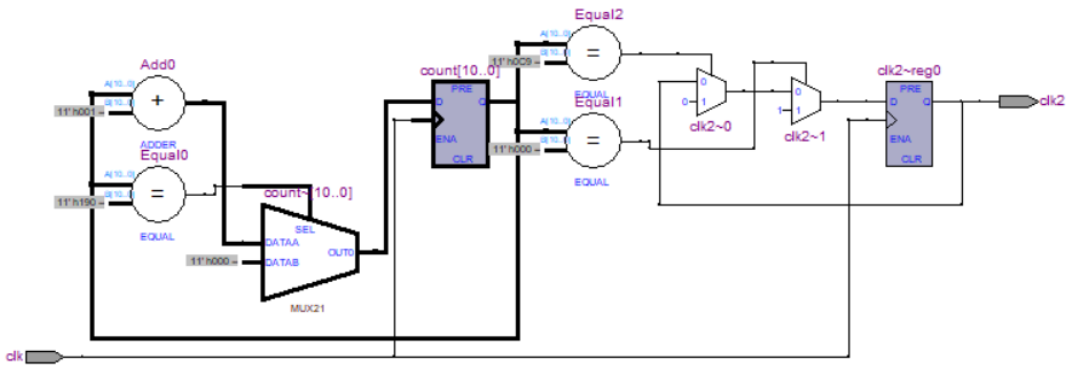
```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity statue is
port(dis:in integer range 0 to 500;
     q:out STD_LOGIC_VECTOR(3 downto 0));
end statue;
architecture fun of statue is
begin
process(dis)
    begin

            IF (dis=0) THEN q<="0000";
            ELSIF (dis>0 )and(dis<=50) THEN q<="0001";
            ELSIF (dis>50 )and( dis<=100) THEN q<="0010";
            ELSIF (dis>100 )and(dis<=150) THEN q<="0100";
            ELSIF (dis>150 ) THEN q<="1000";
            ELSE q<="0000";
        end IF;
    end process;
end fun;
```

发生代码实现：

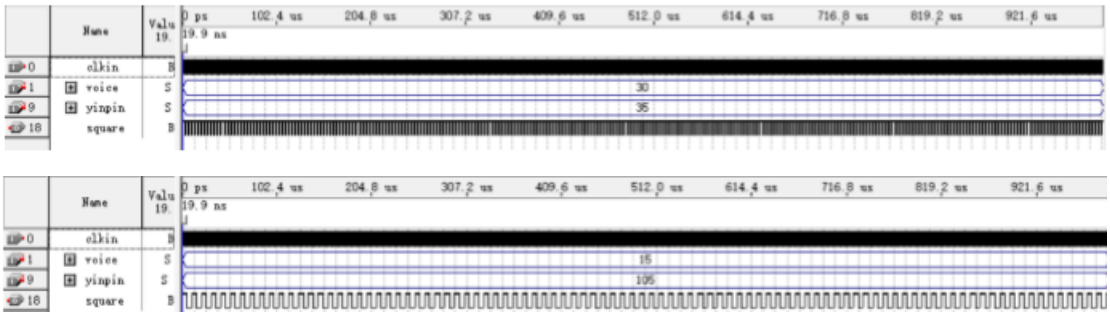
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity Fenpinqi is
     port(clk:in std_logic;
          clk2:out std_logic);
end Fenpinqi;
architecture arch of Fenpinqi is
signal count:integer range 0 to 1200;
begin
process(clk)
begin
if (clk'event and clk='1')then
if(count=400)then
         count<=0;
   else
    count<=count+1;
end if;
end if;
end process;
process(clk)
begin
if(clk'event and clk='1') then
         if(count=0)then
         clk2<='1';
elsif (count=201) then clk2 <= '0';
end if;
end if;
end process;
end arch;
```

实验代码输入为三位的音频控制信号 yinpin_ctrl，主要实现的功能就是将三位的方波转化成对应的八位二进制音频信号。一一对应表明转化关系即可。

RTL 视图



波形仿真



参数设置：end time：1ms；Grid size：100ns；

g） 音量发生器（甘凌志）

模块功能：

设计目的和设计要求是能根据传进来的距离信号选择不同的音频和音量控制信号。

设计一个 8 位的输入表示传进来的距离信号，根据距离信号的大小分成 6 个范围，每个范围对应一个音频控制信号和一个音量控制信号，然后将控制信号输出。

根据前文的 status

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity statue is
port(dis:in integer range 0 to 500;
     q:out STD_LOGIC_VECTOR(3 downto 0));
end statue;
architecture fun of statue is
begin
process(dis)
    begin

            IF (dis=0) THEN q<="0000";
            ELSIF (dis>0 )and(dis<=50) THEN q<="0001";
            ELSIF (dis>50 )and( dis<=100) THEN q<="0010";
            ELSIF (dis>100 )and(dis<=150) THEN q<="0100";
            ELSIF (dis>150 ) THEN q<="1000";
            ELSE q<="0000";
        end IF;
    end process;
end fun;
```

音量发生器实验原理：实验平台的蜂鸣器上输入不同的占空比和频率就可以听到不同的声音，接收到的基准频率按照输入的音频系数分出来的频率进行占空比设计输出到喇叭就可以听到不同的声音

VHDL 代码：

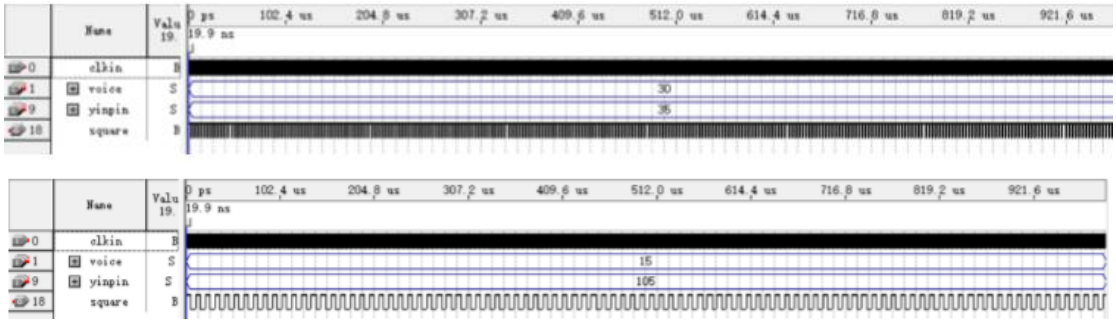
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity pj6 is
    port(clk:in std_logic;
         a : in std_logic_vector(3 downto 0);
         clk3:out std_logic);
end entity pj6;

architecture arch of pj6 is
signal count: std_logic_vector(7 downto 0);
signal clr: std_logic ;
 begin
 process(clk)
   begin
   if(clk'event and clk='1') then
        if(a = "1000")then
            if(count="11111111")then
                count<="00000000";
            elsif(count>"01111111")then
                clr<='1';
            else
                clr<='0';
            end if;
        elsif(a = "0100")then
            if(count="11111111")then
                count<="00000000";
            elsif(count>"00111111")then
                clr<='1';
            else
                clr<='0';
            end if;
        elsif(a = "0010")then
            if(count="11111111")then
                count<="00000000";
            elsif(count>"00011111")then
                clr<='1';
            else
                clr<='0';
            end if;
        else
            if(count="11111111")then
                count<="00000000";
            elsif(count>"01111111")then
                clr<='1';
            else
                clr<='0';
            end if;
        end if;
        count<=count+1;
   end if;
 end process;
 process(clr)
 begin
   if(clr='1') then
       clk3<=clk;
   else
       clk3<='0';
   end if;
 end process;
 end arch;
```

占空比的思考：

最后通过思考以及参考老师的，觉得让固定占空比分母比例，通过控制输入

端输入的音量控制信号作为占空比的分子比例，乘以每个频率可以得到正确的占空比的方波；

仿真：



h) 蜂鸣器发声模块（蔡君浩）

模块功能分析： 对于不同的距离，在喇叭对应的芯片引脚上输入不同的音频信号让喇叭发出声音。

模块方案分析 如图：

```
entity pj6 is
      port(clk:in std_logic;
           a : in std_logic_vector(3 downto 0);
           clk3:out std_logic);
end entity pj6;

architecture arch of pj6 is
signal count: std_logic_vector(7 downto 0);
signal clr: std_logic ;
 begin
 process(clk)
   begin
   if(clk'event and clk='1') then
        if(a = "1000")then
```

就是 4 位输入信号 a，每个区间进行计数，然后确定发出声音的频率的信号

```
   if(a = "1000")then
       if(count="11111111")then
           count<="00000000";
       elsif(count>"01111111")then
           clr<='1';
       else
           clr<='0';
       end if;
```

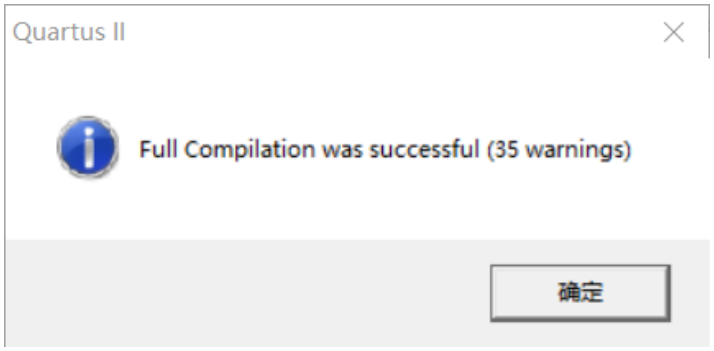
所选方案：每个 4 位输入信号对应不同的区间，计数范围不同如下：

```
if(a = "1000")then
    if(count="11111111")then
        count<="00000000";
    elsif(count>"01111111")then
        clr<='1';
    else
        clr<='0';
    end if;
elsif(a = "0100")then
    if(count="11111111")then
        count<="00000000";
    elsif(count>"00111111")then
        clr<='1';
    else
        clr<='0';
    end if;
elsif(a = "0010")then
    if(count="11111111")then
        count<="00000000";
    elsif(count>"00011111")then
        clr<='1';
    else
        clr<='0';
    end if;
else
    if(count="11111111")then
        count<="00000000";
    elsif(count>"01111111")then
        clr<='1';
    else
        clr<='0';
    end if;
end if;
```

第一个区间位计数为大于 01111111，第二个为大于 00111111，然后 00011111

验证方法：编译成功，下载到实验板上依然正确，并且通过老师的验收。

编译结果：



下一步工作：与输出进行连接即可。

性能分析：

Flow Status	Successful - Fri Dec 27 22:44:40 2019
Quartus II Version	9.0 Build 184 04/29/2009 SP 1 SJ Web Edition
Revision Name	final
Top-level Entity Name	final
Family	Cyclone II
Device	EP2C5T144C8
Timing Models	Final
Met timing requirements	No
Total logic elements	1,146 / 4,608 (25 %)
Total combinational functions	1,146 / 4,608 (25 %)
Dedicated logic registers	97 / 4,608 (2 %)
Total registers	97
Total pins	15 / 89 (17 %)
Total virtual pins	0
Total memory bits	0 / 119,808 (0 %)
Embedded Multiplier 9-bit elements	0 / 26 (0 %)
Total PLLs	0 / 2 (0 %)

总结：这个学期的实验，让我可以准确的去使用 quarts 软件去实现想要实现的功能，并且可以根据 RTL。仿真波形 和时序仿真结果去判断正误，并且新学会了一门语言 VHDL。

四、 实验总结

本次实验，我们小组实验中，遇到了不少问题，比如统一 clk 问题，和数据传输的位数规范问题。

也有 vhdl 设计的问题，比如计数 24 周期，采用二进制计数归零（舍弃）或者 integer 计数转化二进制（采用）的方法，原因是 使用包内函数，大大减少了自己手写代码发生错误的可能性，并且可读性更高