中南大學
CENTRAL SOUTH UNIVERSITY

数字电路
课程设计报告

课题名称：	多种数字信号源发生器
学生姓名：	周万川
学　　号：	8207191622
专业班级：	自动化 1901
同 组 者：	阙章志
指导教师：	陈明义、吕向阳

自动化学院

2021 年 7 月

自动化 学院 数字电路 课程设计任务书

学生姓名	周万川	**班级**	自动化 1901	**学号**	8207191622
指导教师	陈明义			**同组学生**	阙章志
课题名称	多种数字信号源发生器				
课题的主要内容及设计要求	一、多种的数字信号源的设计 1. 能输出正弦波、方波、三角波、锯齿波； 2. 能步进预置不同信号源的输出，并指示输出波形的类型； 3. 信号源频率为 1kHz，峰-峰值为 10V； 4. 方波的占空比为 0.5; 5. 锯齿波的下降时间为 0.1mS 6. 三角波为对称三角波； 7. 能实测并显示输出波形的类型。 除此之外，我们还预置了九档频率				
实践时间	**从 7 月 5 日至 7 月 9 日，小计：5 天**			**实践地点**	综合实验楼 304

目 录

摘要

本次多种数字信号源设计使用 Quartus 13.1 的门电路模块图以及 Verilog HDL 模块混合，仿真验证使用 Modelsim 2020.4，FPGA 型号为 EP4CE10E22C8。

主要设计分为顶层采样参数设计、多分频器模块设计、状态控制计数器设计、可选择自循环式 ROM 设计、波形分辨计数器设计、以及 DA 和电压比较模块设计。

最终实现的功能有：

1. 按键预置四种标准波形
2. 按键预置 9 档频率
3. 数码管显示波形的设置参数
4. LED 显示实测波形种类

整体的执行原则为：先原理图，再代码，再仿真，最后实际搭建电路，调试直到产生预期现象，本文也是按照此逻辑组织阐述的。

关键词 Verilog HDL，数字信号源

Abstract

This multiple digital signal source design uses Quartus 13.1 gate circuit block diagram and Verilog HDL module, simulation verification uses Modelsim 2020.4, FPGA type is EP4CE10E22C8.

The main design is divided into top-level sampling parameter design, multi-frequency divider module design, state control counter design, optional self-circulating ROM design, waveform resolution counter design, and DA and voltage comparison circuit design.

The functions finally we realized are:

1. Press the button to preset four standard waveforms
2. Press the button to preset 9 levels of frequency
3. Display the parameters of current waveform through the digital tube
4. LED display the type of measured waveform

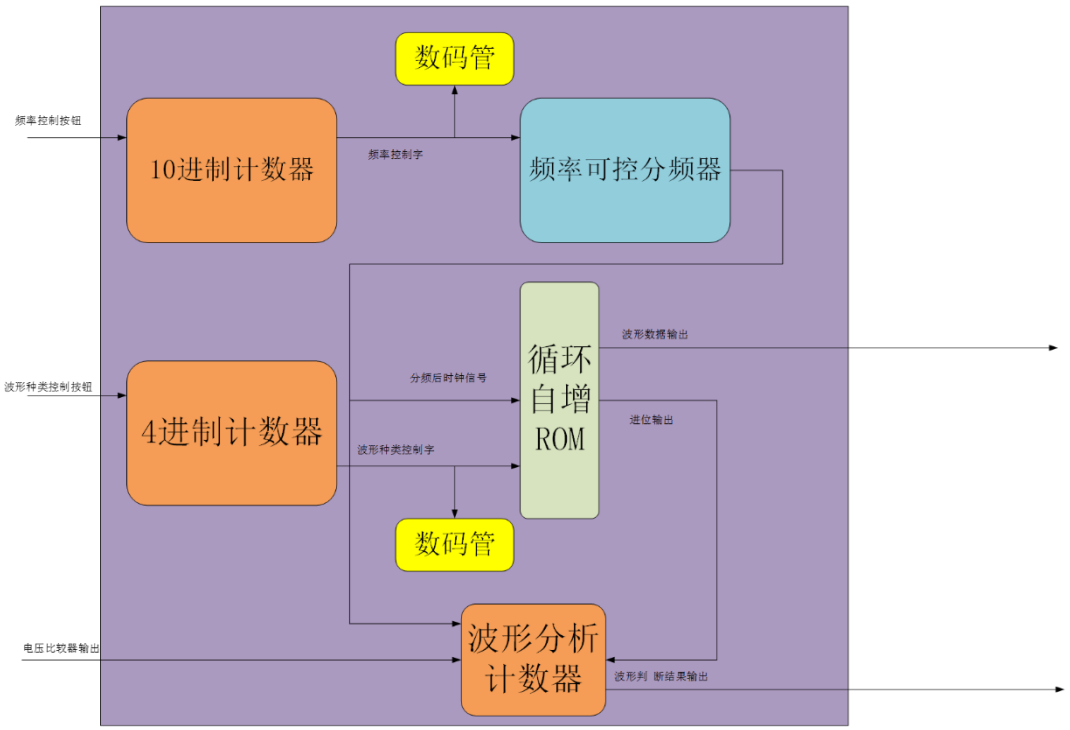
Keyword

Verilog HDL, Multiple Digital Signal Source.

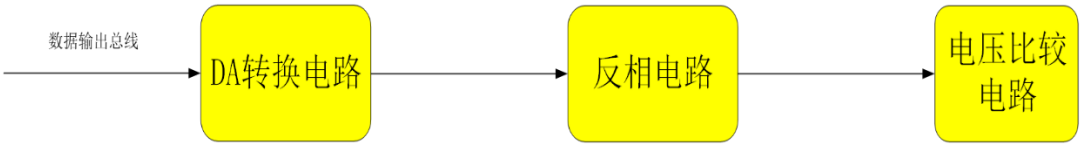
一　系统概述

1.1　总体原理框图

数字电路部分如下图：



模拟电路部分如下图：



1.2　总体工作流程

1.2.1　部分名词解释

数码管：该数码管自带译码器，使用时 DCBA 输入 8421BCD 码即可

频率可控分频器：通过频率控制字译码实现变模

循环自增 ROM：内置计数器，可按照选择状态读取并输出对应存储位置的数据

1.2.2　数字部分

按下频率控制按钮，十进制计数器跳转到下一状态，得到对应的数字码，该码通过译码器，选通对应的数码管段，并改变分频器的分频值，使得循环自增 ROM 的基频改变，从而输出波形的频率改变。

按下模式控制按钮，四进制计数器跳转到下一状态，得到对应状态的高位地址，这样就改变了在 ROM 中读取的波形数据类型。

1.2.3　模拟部分

D7-D0 共 8 位数据，对应引脚连接到 DAC0832 的 DI7-DI0 位，DAC0832 此时输出电流信号，将电流负出端接地，另一端引到一个 OP07。

再将此时的电压输出解接 OP07 反相，再接入参考电压为+5V 的比较电路中，

这样不同的波形达到+5V 的时间不同，产生的电压比较器的输出波形也必然为占空比不同的方波，可以实现波形检测的目的。

二 单元电路的设计与分析

2.1 顶层参数设计

在我们使用的 FPGA 中，系统时钟信号从 PIN24 管脚引出，频率为 20MHz，而我们目标要求的波形频率为 1kHz，所以我们这里设预期输出正弦波在采样前的表达式为：

$$\sin(2\pi f_{1k}t)$$

我们设采样序列数为 n，采样频率为f_{sample}，那么采样后得到的函数表达式为：

$$\sin\left(2\pi\frac{f_{1k}}{f_{sample}}n\right)$$

若我们设 ROM 中，分配给每个种类波形的存储空间数为 1k 个，那么对于每个高位地址而言，将有 24 个地址存储值为 0（或任意数，因为这段空间在切换基址时以及在顺序读取时都是被跳过的），那么，一个波形数据的完整地址表示为：

$$\{Waveform,n\}$$
$$\downarrow \quad \downarrow$$
$$2bit \quad 10bit$$

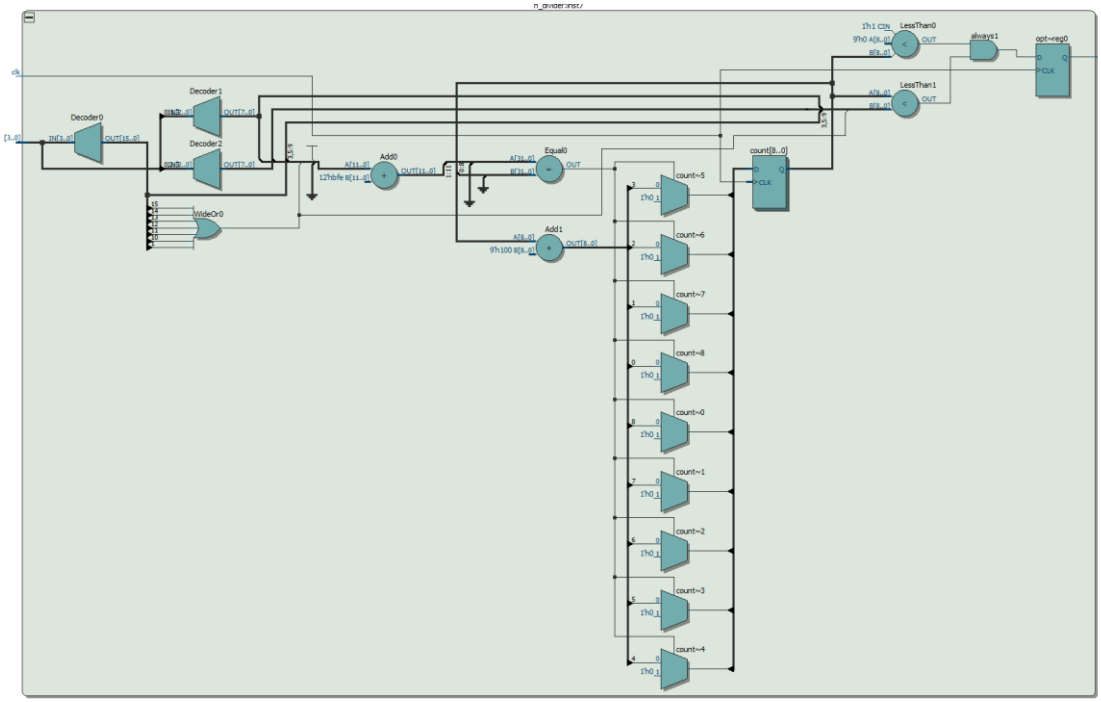
这样我们有以下约束：

$$f_{sample}=f_{1k}\times 1000$$

于是有$f_{sample}=1MHz$，这样，当我们使用可变频率分频器使得输出采样频率改变为它的 1/2，1/4，1/8，...倍时，对应波形的周期也会改变为它的 2，4，8，...而这在示波器上是容易观察验证的。

2.2 多分频器模块设计

在这里我们先给出多分频器的门级电路设计：



代码如下：

```
//将时钟分频为 2^(1~9)+0：20
module n_divider (
    input [3:0] n,
    input clk,
    output reg opt
);

    reg [9:0] div;
     reg [8:0] pulse;

    always @(*) begin
        case(n)
        4'd0:   div=10'd20;
        4'd1:   div=10'd2;
        4'd2:   div=10'd4;
        4'd3:   div=10'd8;
        4'd4:   div=10'd16;
        4'd5:   div=10'd32;
        4'd6:   div=10'd64;
        4'd7:   div=10'd128;
        4'd8:   div=10'd256;
        4'd9:   div=10'd512;
        default: div=10'd2;
        endcase
        pulse=div/10'd2;
    end
```

```
    reg [8:0] count=0;
    always @(posedge clk) begin
        if (count==div-1) begin
            count<=0;
        end
        else begin
            count<=count+1'b1;
        end

        if ((count >= 0) && (count<pulse))
            opt<=1;

        else opt<=0;
    end
    endmodule
```

可以看出，此模块实现的方法为：

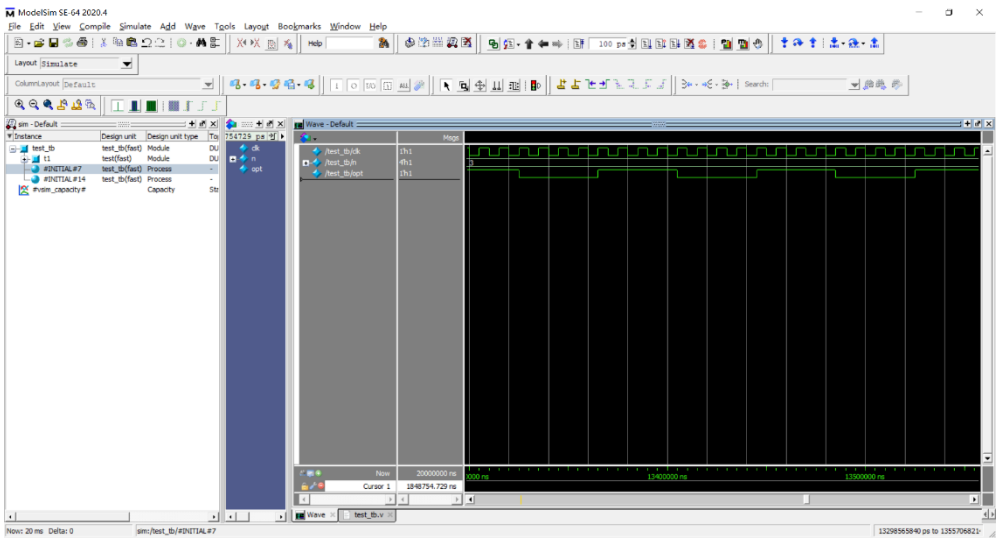
首先，利用一个 10*10bit 的 ROM 存储与分频数对应的计数次数，0 为默认，这样分频得到 1MHz 的时钟信号，1-9 对应分频得到的时钟信号为 20/(2^n)MHz 的频率，通过译码得到对应的计数值后，除以 2，可得当前的高电平计数值，因为时钟信号的默认占空比为 0.5，这之后，再实现一个特殊的变模计数器，它的模等于计数次数，每个时钟触发一次，自动判断当前的计数个数是否超过占空计数个数，若超过，则将输出信号置一，否则置 0。

我们编写 Modelsim 测试程序如下：

```
`timescale 1us/1ps
module test_tb;
    reg clk;
    reg [3:0] n;
    wire opt;
    test t1(clk,n,opt);
    initial begin
        n=1;
        #10000
        n=3;
        #10000
        $stop;
    end
    initial begin
        clk=0;
        forever begin
        #5 clk=~clk;
        end
    end
```

```
endmodule
```

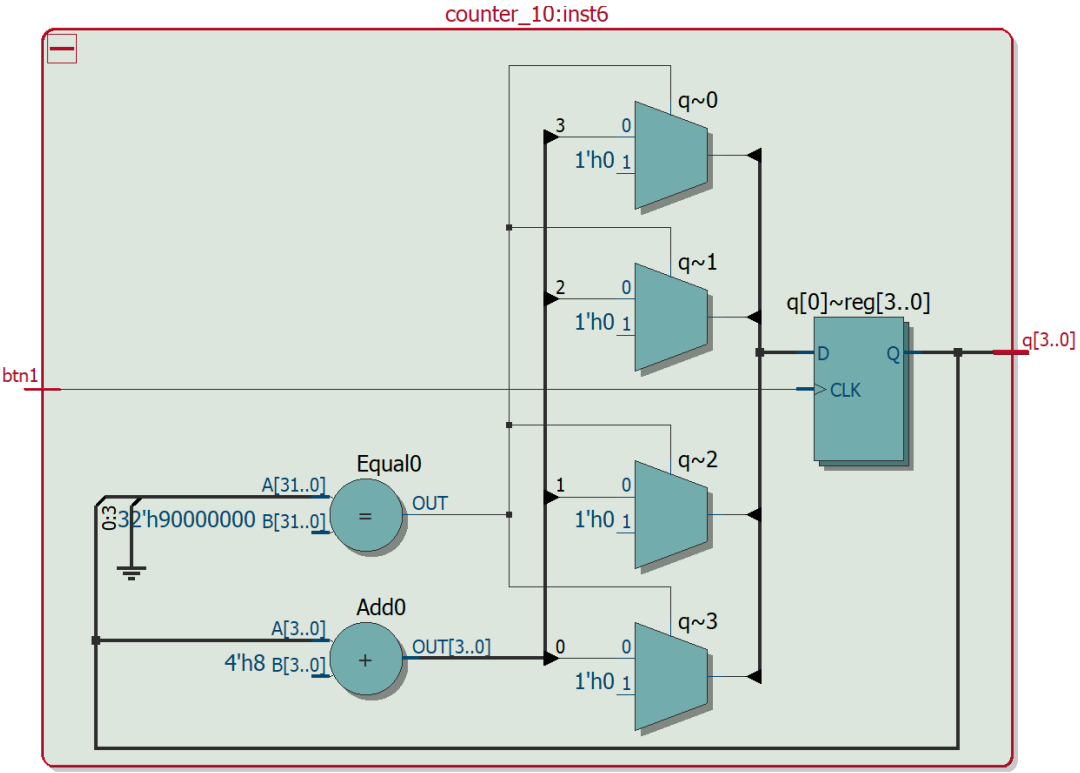
该程序实现的功能为：输入一个 0.1MHz 的时钟信号，每隔 10ms 改变一次分频数，这样我们只需测试得到的输出是否与我们的分频数对应即可验证我们程序的功能，验证结果如下：

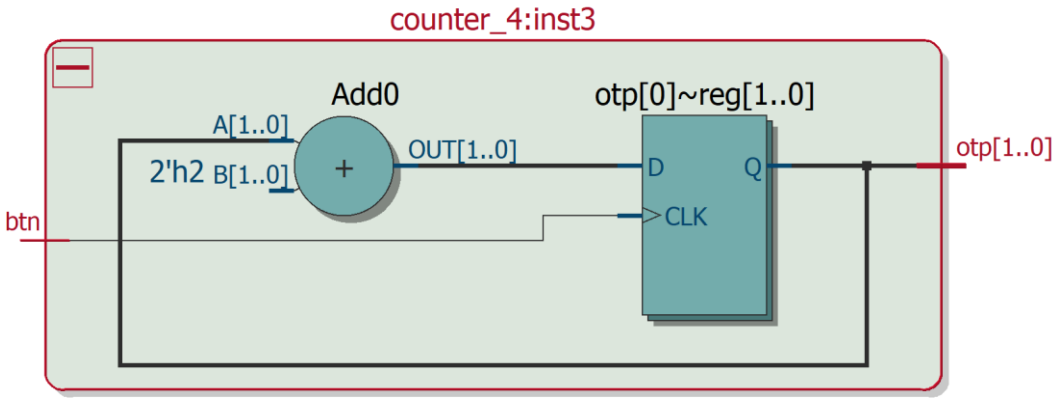


可以看出，该模块设计正常。

2.3 状态控制计数器设计

我们先给出状态控制计数器的门级电路设计：





它们的实现代码分别给出如下：

```
//btn1 周期增加，显示几表示      基频/2^n
//共 1+九档频率  0 默认为 20 分频
module counter_10 (
    btn1,q
);
    input   btn1;
    output reg[3:0] q;
    initial begin
        q=4'h0;
    end

    always @(posedge btn1) begin

            if(q==9) q= (4'h0);
            else
            q=(q+1'b1);


    end

    endmodule

module counter_4 (
    btn,
    otp,

);
    input btn;
    output reg[1:0] otp;
    initial begin
        otp=2'b00;
    end
```

```
    always @(posedge btn) begin
        otp<=(otp+1'b1)%4;
    end
    endmodule
```

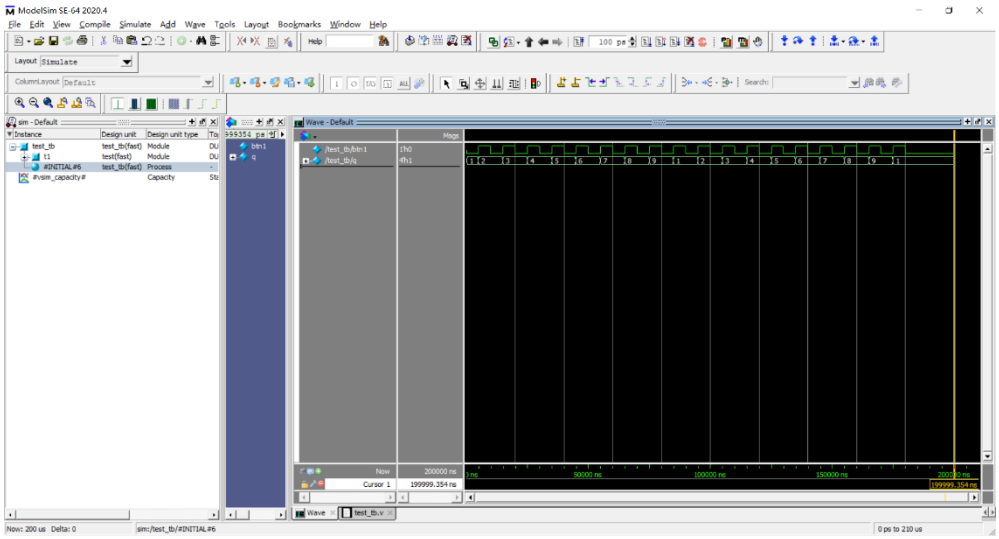
该代码实现的功能很简单，即我们只需要让该计数值在模之内递增即可。

我们编写 10 进制程序的测试文件如下：

```
`timescale 1us/1ps
module test_tb;
reg btn1;
wire [3:0] q;
test t1(btn1,q);
initial begin
btn1=0;
repeat(18)
 #5 btn1=~btn1;
 repeat(18)
  #5 btn1=~btn1;
  #20 $stop;
end


    endmodule
```

该测试文件模拟按下 18 次按键，我们只需观察输出值的变化是否在 0-9 之间递增变化即可验证模块功能。

将文件加载到 Modelsim 中，运行结果如下：



我们可看出，该计数器工作正常。

2.4 变模式自循环 ROM 设计

2.4.1 说明

该 ROM 中，地址的高 2 位无法由计数器改变，时钟上升沿到来时，1000 进

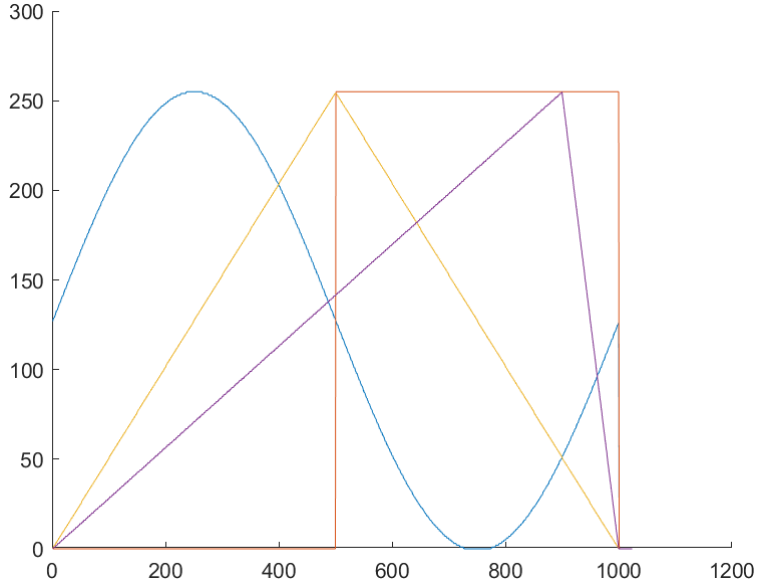
制计数器自动改变一次状态，如此可实现名称中的功能。

2.4.2 数据生成

我们使用 matlab 编程，并在程序中直接将其转换为 8 位数据，给出程序如下：

```
Sample_sin=round(127+128*sin(w*T));

Sample_square=[linspace(0,0,DataLength/2),255*linspace(1,1,Da
taLength/2)];

rise1=2*10^3*T(1:DataLength/2);
Sample_iso=round(255*[rise1 rise1(end:-1:1)]);

rise2=1/(0.9*10^(-3)) * T(1:DataLength*0.9);
down2=10^4 * T(1:DataLength*0.1);
Sample_jag=round(255*[rise2 down2(end:-1:1)]);

s=[Sample_sin , linspace(0,0,24) , Sample_square ,
linspace(0,0,24) , Sample_iso , linspace(0,0,24) ,
Sample_jag , linspace(0,0,24)];
```

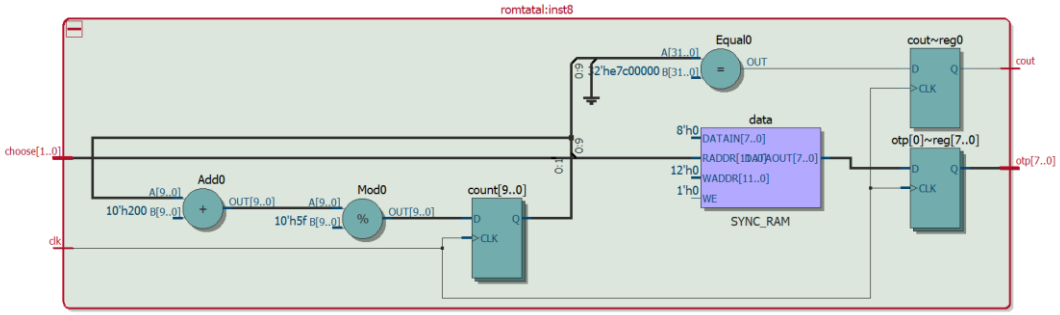
实际上，DAC0832 输出的数据是反向的，所以在数据生成时还需要使用 255-s(x)，这样输出的数据才为正向的。用 matlab 生成采样数据图像如下：



可以看出 1000 个采样点，实现的采样效果还是较好的。

2.4.3 ROM 程序代码及验证仿真

首先给出其门级电路图如下：



代码如下，因为数据点太多，所以这里就不显示了。

```
module romtatal (
    clk,
    choose,
    cout,
    otp
);
    input clk;
    input [1:0] choose;
    output reg[7:0] otp;
    output reg cout;
    reg [7:0] data[4095:0];
    reg [9:0] count;
    reg [9:0] mod;

    initial begin
        cout=0;
         otp=0;
         count=0;
        mod=1000;
           data[4095]=0;
    ...
           data[0]=127;

    end
    always @(posedge clk) begin

         otp<=data[{choose,count}];
        count<=(count+1'b1)%mod;
    end
    always @(posedge clk) begin
        if(count==999) cout<=1;
        else cout<=0;
    end
    endmodule
```

```
    reg [7:0] data[4095:0];
```

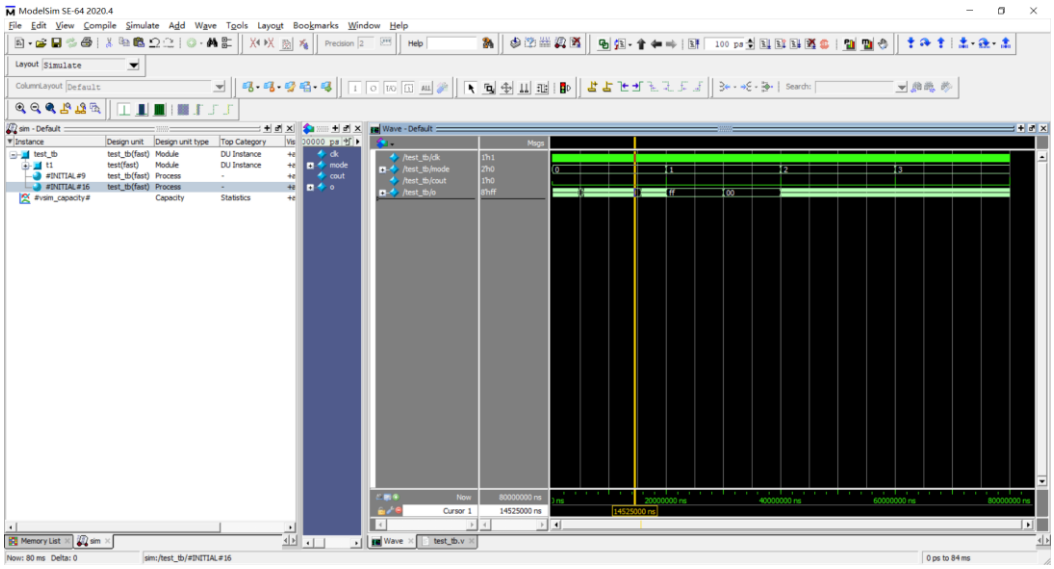
即定义了一个 8*4096 的 ROM，在每次完成 1000 次计数时，将输出一个进位信号，用于给分辨计数器判断波形周期完结置位，其他按说明理解即可。

2.4.4 仿真程序代码

```
`timescale 1us/1ps
module test_tb ;
reg clk;

reg [1:0] mode;
wire cout;
wire [7:0] o;
test t1(clk,mode,cout,o);
initial begin
    clk=0;
    forever begin
        #5 clk=~clk;
    end
end

initial begin
    mode=0;
    #20000
    mode=1;
    #20000
    mode=2;
    #20000
    mode=3;
    #20000
    $stop;
end

    endmodule
```

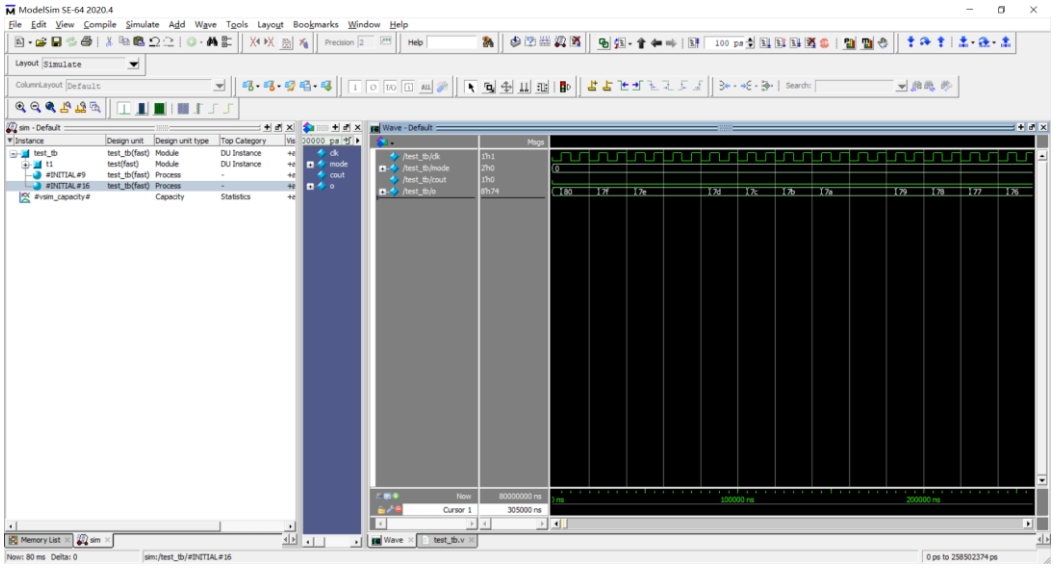
这段代码实现的功能为，给组件一个 0.1MHz 的激励时钟，每 20ms 改变一次状态，预计每个波形数据将被输出 1 次。

仿真结果如下：



可知每个周期输出 cout 正常，可以提示波形周期。

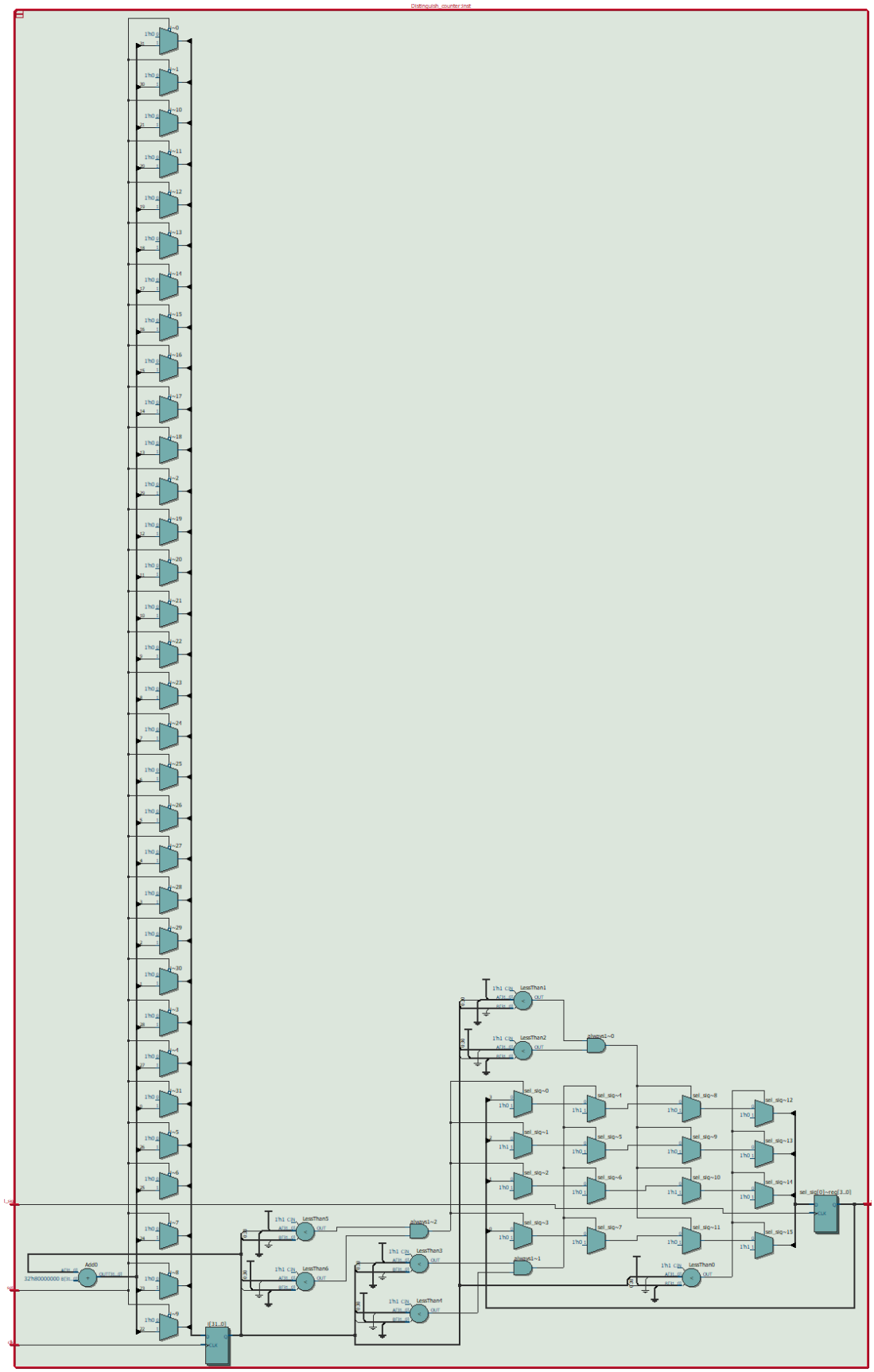
如图，将局部放大，与存储数据对比：



可知此模块设计正常。

2.5 波形分辨计数器设计

首先给出门级电路图如下：



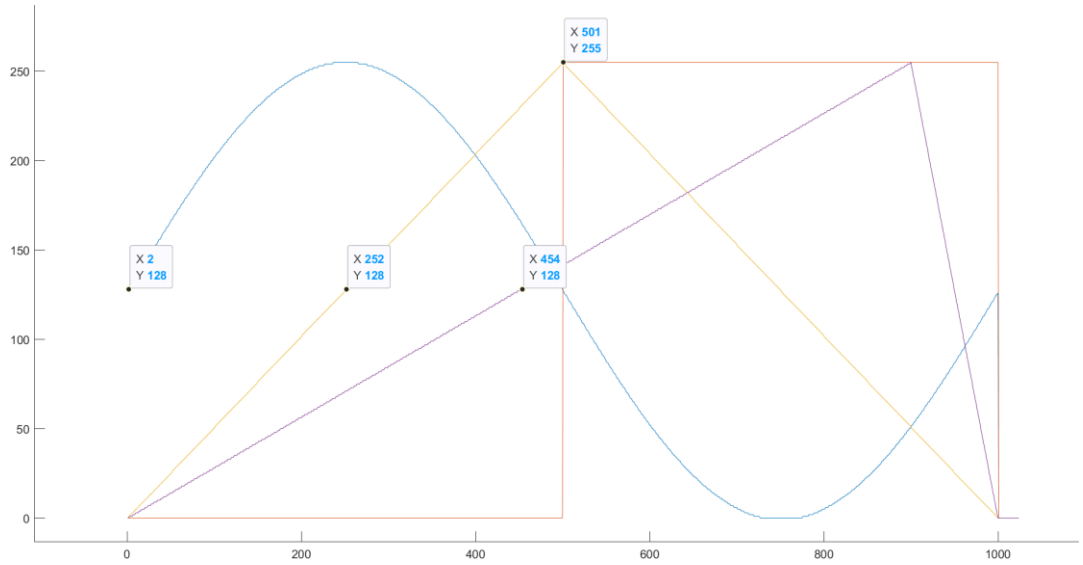
其模块代码如下：

```
//`timescale 1ns/1ps
//cmp1_sig：电压比较器输出信号，超过 Uref 时，置为逻辑一，Uref 编码为 200~201 左右
//clk      ：采样时钟，每遇到一次它的上升沿，计数自增一次,1MHz
//set      ：置位信号，从 1000 进制计数器的进位端引出接上，使得每个周期都可以读取并判断
//sel_sig ：根据计数的多少输出的选择信号，0001 为正弦，0010 为 iso，0100 为 square，1000 为 jag
module Distinguish_counter (
    input wire cmp1_sig,
    input wire clk,
    input wire set,
    output reg [3:0] sel_sig
);
integer i=0;



always @(posedge clk) begin
        if(!set)
        i=i+1;
        else
        i=0;
end

always @(posedge cmp1_sig) begin
    if (i<=40) begin
        sel_sig=4'b0001;
        //$display("1 i=%d",i);
    end else begin
        if (i>=220 && i<=270) begin
            sel_sig=4'b0010;
            //$display("2 i=%d",i);
        end else begin
            if (i>=410 && i<=465) begin
                sel_sig=4'b1000;
                //$display("3 i=%d",i);
            end else begin
                if (i>=485 && i<=520) begin
                    sel_sig=4'b0100;
                    //$display("4 i=%d",i);
                end
            end
```

```
        end
    end
    //$display("i=%d",i);
end

    endmodule
```

其实现的功能为，每个波形周期，将计数器置零，重新检测，在每个周期内，按照输入的时钟频率，让计数器计数，检测到电压比较器输入的方波上升沿后，即可将此时计数器的值与设定范围进行比较，若范围合适，则点亮对应的 LED，因为电压比较器的设定电压为 5V（实验箱环境限制），所以会导致正弦波的判断时多个 LED 同时亮起，这是因为判断时间太短以及模拟电路不精确，在此我们给出设定电压的选定以及计数器比较范围的设定原则：

先给出一些波形图上的数据点：



因为参考电压 5V 对应的二进制码值为 128（目标电压 Vpp=10V）所以在图上标出各个波形达到此值的计数个数，分别为：

正弦波	2
三角波	252
方波	501
锯齿波	454

这代表着，在一个波形周期内，电压比较器输出上升沿的时机，也代表着，在上升沿到来时，计数器计数的个数，因为各个波形不同，所以我们可以依据此设计范围比较（详见代码），来实现分辨。

我们指出，因为参考电压以及模拟电路带来的精度损失，所以我们的正弦波判断时计数次数太少，而方波与锯齿波的计数值靠的比较近，这样就降低了判断的稳定性，在实际测量中我们发现，对于电路在较底频率输出时（7-9 档），输出的判断结果十分稳定且无重复。

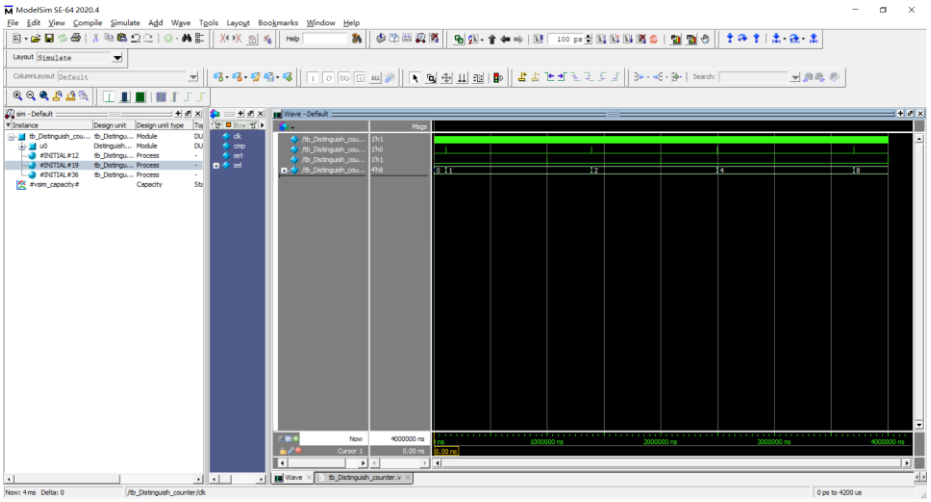
我们要给出程序的仿真以及结果：

```
`timescale 1ns/1ps
module tb_Distinguish_counter;
```

```
reg clk,cmp,set;
wire [3:0] sel;
Distinguish_counter u0(
        cmp,
        clk,
        set,
        sel
);
initial begin
    clk=1'b0;cmp=0;set=0;
forever begin
    #500 clk=~clk;
end
end

initial begin
    #4000_000 $stop;
end
initial begin
    #110000 cmp=1;#8 cmp=0;
end
initial begin
    #1390000 cmp=1;#8 cmp=0;
end

initial begin
    #2499000 cmp=1;#8 cmp=0;
end

initial begin
    #3700000 cmp=1;#8 cmp=0;
end
initial begin
    #999000 set=1;#1000 set=0;
    #999000 set=1;#1000 set=0;
    #999000 set=1;#1000 set=0;
    #999000 set=1;#1000 set=0;

end
    endmodule
```

该仿真代码实现的功能为，采样周期为 1us，每个波形周期末计数满模值时输出一个 set 置位信号，重新判断波形类型。

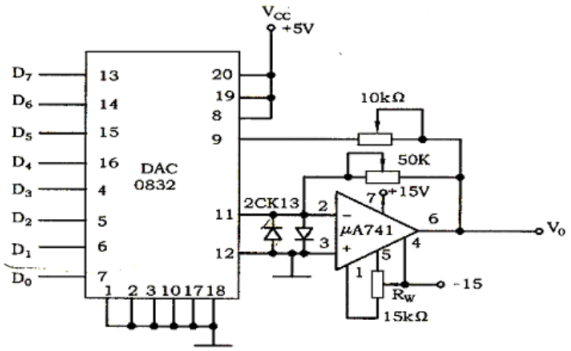
该代码的仿真输出如下：



2.6 模拟电路设计

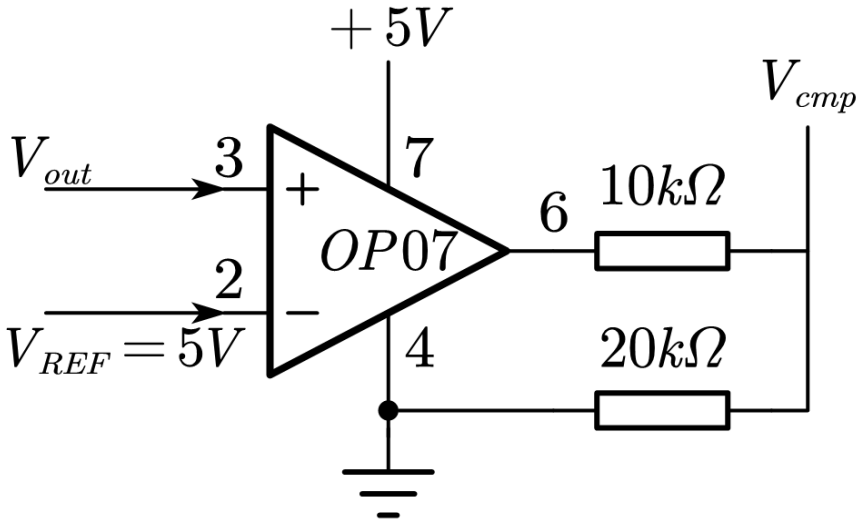
2.6.1 DAC0832

我们采用直通接法，接法如下：



2.6.2 电压比较器

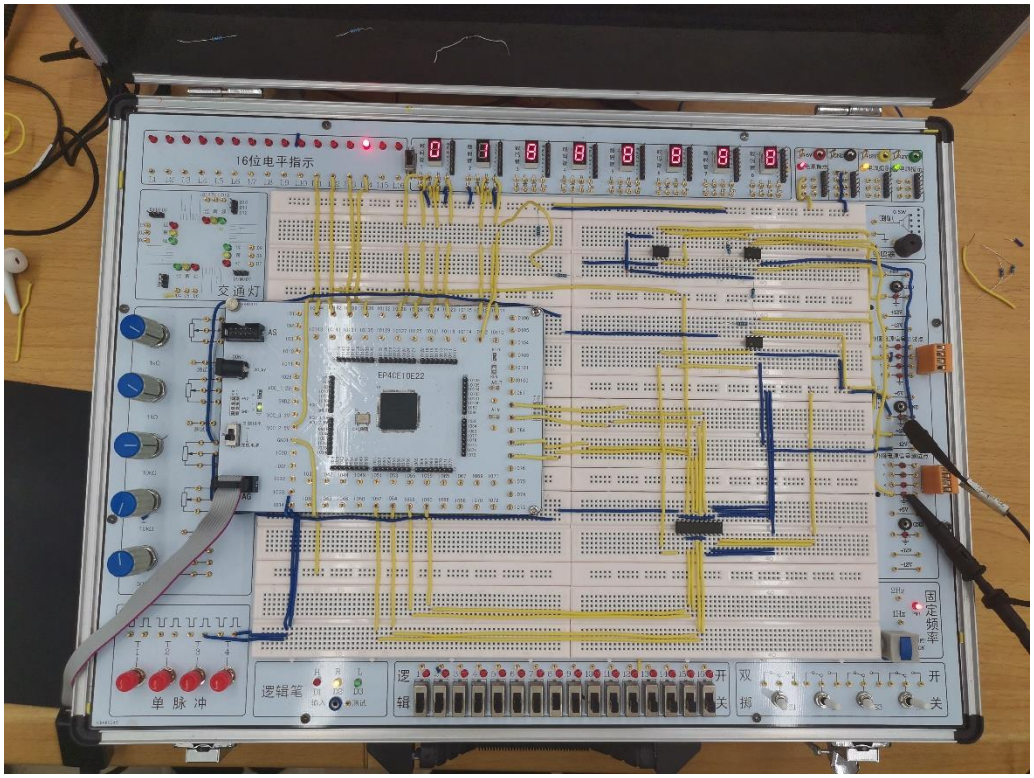
如下图：



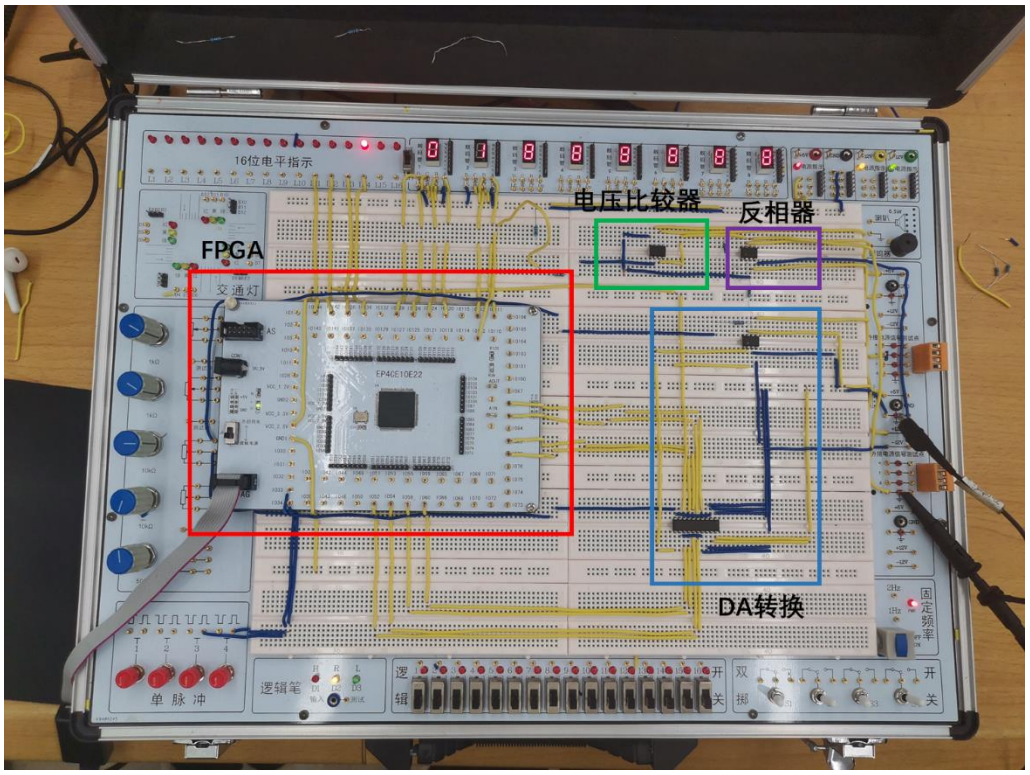
三　电路的安装与调试

3.1　电路插接

插接电路，得到如下：



我们给出标注版：

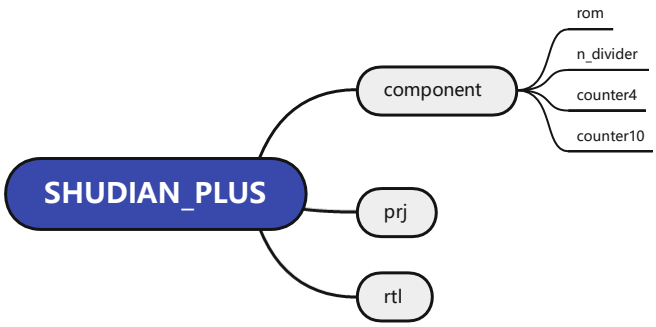


3.2 电路调试

在实际实验过程中，DAC0832 芯片出现了问题，我们花了两天时间排查，先将最终得到的最优排错方案给出如下：

（1）原理级错误排查

这里建议将课设工程文件的结构理清楚，我们给出一种范例（假设工程名为 SHUDIAN_PLUS）：



我们首先构思构成整个系统的模块框图，再在 component 中新建.v 文件编写模块源代码，然后在当前模块目录下新建工程，命名为 test，设计添加 tb 文件仿真，直到达到仿真预期时，认为元件可用，将其添加到项目的 rtl 文件夹作为源码。

所有的模块都完成后，我们在 prj 中新建与项目名称一致的工程，添加对应的组件，编写顶层文件即可。

这时如果有模块怀疑出错了，那么就可以在 component 库中改变 tb 文件重新测试或者改变.v 文件，使得 debug 更有条理。

当然这也依赖于 FPGA 设计的低耦合特性。

（2）DAC 芯片错误排查

在本次实验中，我们小组的 DAC0832 只有 3 个数据引脚可以使用，我们得到最优的排查方法有两种，分别介绍如下：

法一：

此法不需要重新插接。假设此时电路已插接好，但是 DAC 输出异常，在排查完 FPGA 设计错误、DAC 非数据引脚连接错误后，我们可以如此做：

1. 编写一段数据依地址线性变化的 mif 文件烧入 FPGA。使得每一位的数据正常情况下都对输出有影响。
2. 依次测量每个 FPGA 的输出数据端波形，确定每个都有波形。
3. 将某个插接线从 FPGA 一端拆下，观察波形变化。
4. 将刚刚的插接线接到其他位的数据输出端，观察波形变化。

若有：

波形先不变，后不变，说明 FPGA 管脚有输出，DAC 对应数据位坏了。

波形先改变，说明有输出，对应位没有坏。

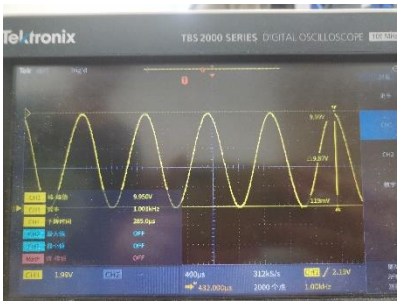
波形先不变，后改变，说明 FPGA 管脚没有输出，DAC 对应数据位是好的。

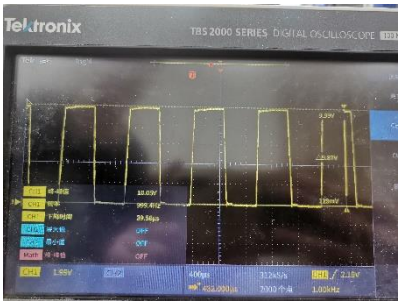
法二：

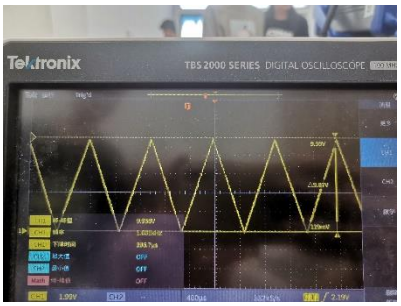
将 DAC0832 的每个数据输入端接到开关，保持其他位不变，改变其他位，测量放大电路端的输出电位值，依次测量，即可得出各个位的引脚情况。

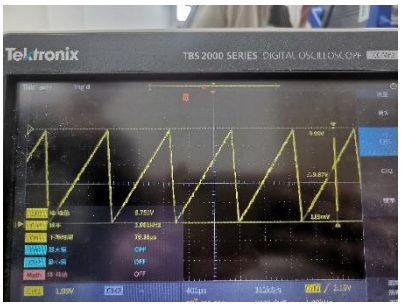
3.3 电路输出效果图

输出效果如下图：









四　结束语

4.1 改进意见

可以加入幅值预置功能，考虑用乘法器或者是增扩 ROM 实现。

检测波形时不能要求峰峰值 10V，否则检测易出现错误。

4.2 收获与感谢

通过本次数电课程设计，我复习提升了数电相关知识，掌握了 Verilog HDL 语言以及 Modelsim 联合 Quartus 调试的方法，最关键的是，在调试电路的漫长过程中，我磨练了心智与能力。

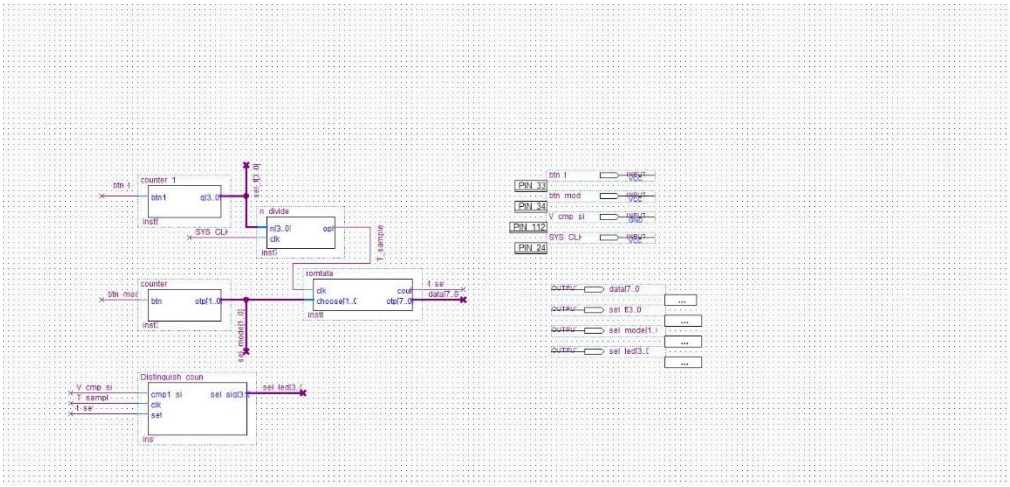
在这里诚挚地感谢我的队友阙章志同学，没有你我们不可能完成错误的排查和提高功能的实现。

诚挚地感谢陈明义、吕向阳老师，你们的指导和帮助对于我们完成本次工程的设计起到了至关重要的作用，谢谢你们！

附件 1：元器件清单

元器件	所需数目
DAC0832	1
OP07	3
电阻$10k\Omega$	1
电阻$20k\Omega$	1
电阻$4.7k\Omega$	2
电位器$100k\Omega$	2
LED	4
数码管（带译码）	2
脉冲按钮	2

附件 2：原理图



参考文献

[1]孙昊. 基于 FPGA 的 DDS 信号源设计[D].电子科技大学,2009.