实验题目：频率计设计

班级：无 42

学号：2014011058

姓名：陈誉博

日期：2016 年 5 月 6 日

一、 实验目的

1．掌握频率计原理和实现方法；

二、 实验原理及设计思路

1. 实验原理

频率计用于对一个未知频率的周期信号进行频率测量，在 1s 内对信号周期进行计数，即为此周期信号的频率。

频率计内部实现框图如下所示，其内部包括频率量程处理模块（10 分频）、时钟频率产生模块、控制信号产生模块、十进制计数器模块、锁存器模块、译码显示模块等。

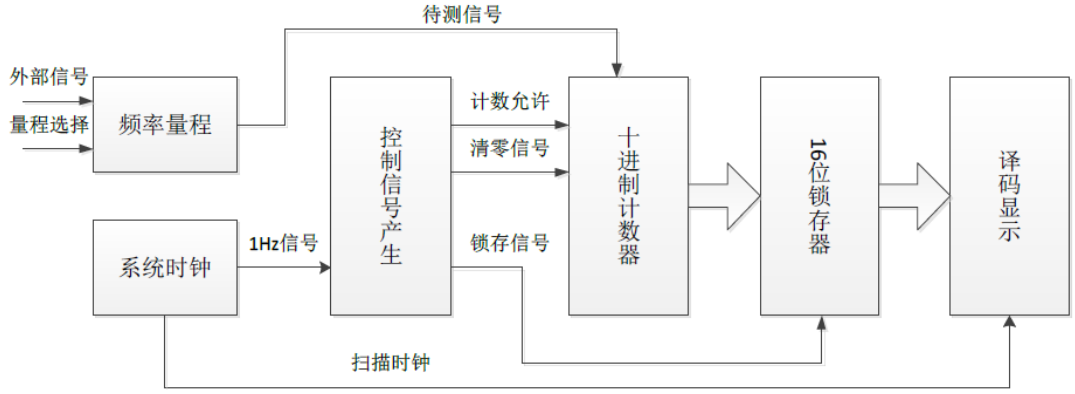


图 1 频率计的原理框图

利用系统时钟产生 1Hz 的控制信号，在 1s 的时长内利用计数器对待测信号进行计数，将计数结果锁存（或者保存，不是指 latch）并输出到数码管中显示。其中频率量程模块负责根据设定的量程控制信号决定是否对输入信号进行 10 分频；系统时钟模块根据外部输入的参考时钟产生标准 1Hz 的控制信号；控制信号产生模块产生计数所需的使能、清零信号以及保存测量结果所需的锁存信号和扫描显示所需的扫描时钟信号；十进制计数模块在计数使能、清零信号控制下对外部输入信号（或其 10 分频信号）在 1s 周期内对其进行计数操作；锁存器模块在计数完成之后对计数结果进行锁存，保存上一测量周期的测量结果；译码显示模块将测量结果输出到 LED 数码管显示，采用扫描的方式实现多位数据的同时显示。

2. 设计思路

1) 频率量程模块

先对外部信号做十分频处理并存在一个临时寄存器中。然后用一个受量程选择信号控制的二路选择器选择输出外部信号还是十分频之后的信号。分频操作思路见下一条。

2) 1Hz 和 1kHz 信号产生模块

首先设定分频比，100MHz 分频为 1Hz 和 1kHz 的分频比分别为 100M 和 100k。设定好分频比之后检查系统时钟的上升沿，用一个寄存器变量对上升沿的数目进行计数。如果上升沿的数量达到了分频比的一半的话，输出分频信号的逻辑翻转，并对计数变量清零。这样的话可以实现对系统时钟的分频操作。用同样的思路，将分频比设为 10，就可以对外部信号进行十分频的操作。

3) 控制信号产生模块

在本次实验的设计中，计数使能、reset 和锁存的控制都由同一个控制信号完成。当 1Hz 的信号上升沿来临的时候，控制信号的逻辑翻转。这样的话控制信号每两次逻辑翻转之间的时间间隔为 1s，可以实现在 1s 的时间之内计数的要求。当控制信号为低电平的时候进行

reset 和锁存操作。

4) 十进制计数器

十进制计数器受控制信号和频率量程模块的输出信号控制。当控制信号为低电平时，输出的四位计数变量恒为 0。当控制信号为高电平时，每当频率量程模块的上升沿来临，计数变量加一。

5) 16 位锁存器

当控制信号的下降沿来临时，锁存模块将计数器的当前输出锁存到另外四个寄存器变量中并输出。

6) 译码显示

将锁存模块的四位输出转换成 8 位 BCD 码控制实验平台上的数码管显示数字。同时采用 1ms 刷新一次的方法实现四个数字数码管显示不同数字的功能。

三、 关键代码及源文件清单

1. 四位数字转为八位 BCD 码：bcd7.v:

```
//bcd7.v
module BCD7(din,dout);
input [3:0] din;
output [7:0] dout;
assign dout=(din==4'b0000)?8'b0111_1110:
           (din==4'b0001)?8'b0011_0000:
           (din==4'b0010)?8'b0110_1101:
           (din==4'b0011)?8'b0111_1001:
           (din==4'b0100)?8'b0011_0011:
           (din==4'b0101)?8'b0101_1011:
           (din==4'b0110)?8'b0101_1111:
           (din==4'b0111)?8'b0111_0000:
           (din==4'b1000)?8'b0111_1111:
           (din==4'b1001)?8'b0111_1011:8'b0000_0000;

endmodule
```

2. 测试信号产生模块：signalinput.v

```
module signalinput(input [1:0] testmode,
//00,01,10,11 mean four frequencies, 3125,6250,50,12500Hz
//ocntrol by SW1-SW0
input sysclk,
output sigin1);
reg [20:0] state;
reg [20:0] divide;
reg sigin;
assign sigin1=sigin;
initial begin
   sigin=0;
   state=21'b000000000000000000000;
   divide=21'b000000_1111_1010_0000000;
```

```
end
always @(testmode)
begin
    case(testmode[1:0])
    2'b00:divide=21'b000000_1111_1010_0000000;//3125Hz
    2'b01:divide=21'b0000000_1111_1010_000000;//6250Hz
    2'b10:divide=21'b1111_0100_0010_0100_00000;//50Hz
    2'b11:divide=21'b00000000_1111_1010_00000;//12500Hz
    endcase
end
always @(posedge sysclk)//divide the frequency by divide
begin
    if(state==0)
        sigin=~sigin;
    state=state+21'b0_00_0000_0000_0000_0000_10;
    if(state==divide)
        state=27'b000_0000_0000_0000_0000_0000_0000;
end
endmodule
```

3. 频率计模块：frequency2.v

```
module frequency2(sigin,sysclk,modelcontrol,highfreq,cathodes,AN);
input sigin;
input sysclk;
input modelcontrol;
output highfreq;
//output control_ref;
output [7:0]cathodes;
output [3:0]AN;
reg [3:0]AN;
reg clk_one_hz;
reg clk_one_khz;
reg [27:0] clk_one_counter;
reg [27:0] clk_one_divide;
reg [17:0] clk_onek_counter;
reg [17:0] clk_onek_divide;
reg test_signal;
reg [3:0]test_signal_counter;
reg [3:0]test_signal_divide;
reg sigin_final;
reg control_signal;
reg [3:0] number4;
reg [3:0] number3;
reg [3:0] number2;
reg [3:0] number1;
```

```
reg [3:0] savenum4;
reg [3:0] savenum3;
reg [3:0] savenum2;
reg [3:0] savenum1;
reg [3:0]save;
wire [7:0]digi;
//reg control_ref;
assign highfreq=modelcontrol;
initial begin
    clk_one_hz=0;
    clk_one_khz=0;
    clk_one_counter=27'b000000000000000000000000000;
    clk_one_divide=27'b101111101011110000100000000;
    clk_onek_counter=17'b00000000000000000;
    clk_onek_divide=17'b11000011010100000;
    test_signal=0;
    test_signal_counter=4'b0000;
    test_signal_divide=4'b1010;
    sigin_final=0;
    control_signal=0;
    number4=4'b0000;
    number3=4'b0000;
    number2=4'b0000;
    number1=4'b0000;
    savenum4=4'b0000;
    savenum3=4'b0000;
    savenum2=4'b0000;
    savenum1=4'b0000;
    AN[0]=1;
    AN[1]=1;
    AN[2]=1;
    AN[3]=0;
    save=4'b0000;
    //control_ref=0;
end
//generate a 1hz signal
always @(posedge sysclk)
begin
    if(clk_one_counter==0) begin
    clk_one_hz=~clk_one_hz;
    end
    clk_one_counter=clk_one_counter+27'b000000000000000000000000001
0;
    if(clk_one_counter==clk_one_divide) begin
```

```
    clk_one_counter=27'b000000000000000000000000000;
    end
end
//generate a 1khz signal
always @(posedge sysclk)
begin
    if(clk_onek_counter==0) begin
    clk_one_khz=~clk_one_khz;
    end
    clk_onek_counter=clk_onek_counter+17'b00000000000000010;
    if(clk_onek_counter==clk_onek_divide) begin
    clk_onek_counter=17'b00000000000000000;
    end
end
//divide frequency depend on modelcontrol
always @(posedge sigin)
begin
       if(test_signal_counter==0) begin
          test_signal=~test_signal;
       end
       test_signal_counter=test_signal_counter+4'b0010;
       if(test_signal_counter==test_signal_divide) begin
          test_signal_counter=4'b0000;
       end
end
always @(sysclk)
begin
    if(modelcontrol==1) sigin_final=test_signal;
    else sigin_final=sigin;
end
//generate the control signal
always @(posedge clk_one_hz)
begin
    control_signal=~control_signal;
    /*if(control_signal==1)
         control_ref=1;
    else
         control_ref=0;*/
end
//dec_counter module
always @(posedge sigin_final)
begin
    if(control_signal==0) begin
       number4=4'b0000;
```

```
        number3=4'b0000;
        number2=4'b0000;
        number1=4'b0000;
    end
    else begin
        number1=number1+4'b0001;
        if(number1==4'b1010) begin
            number2=number2+4'b0001;
            number1=4'b0000;
        end
        if(number2==4'b1010) begin
            number3=number3+4'b0001;
            number2=4'b0000;
        end
        if(number3==4'b1010) begin
            number4=number4+4'b0001;
            number3=4'b0000;
        end
    end
end
//save module
always @(negedge control_signal)
begin
    savenum4=number4;
    savenum3=number3;
    savenum2=number2;
    savenum1=number1;
end
//display module
always @(posedge clk_one_khz)
begin
    if(AN==4'b1110) begin
        AN=4'b1101;
        save=savenum3;
    end
    else if(AN==4'b1101) begin
        AN=4'b1011;
        save=savenum2;
    end
    else if(AN==4'b1011) begin
        AN=4'b0111;
        save=savenum1;
    end
    else if(AN==4'b0111) begin
```

```
        AN=4'b1110;
        save=savenum4;
    end
    else AN=4'b1110;
end
BCD7 bcd7(save,digi);
assign cathodes=~digi;
endmodule

module test(input [1:0] testmode,input sysclk,input modelcontrol,
output highfreq,output [7:0]cathodes,output [3:0]test_AN);
wire sigin;
signalinput signalin(testmode,sysclk,sigin);
frequency2
freq(sigin,sysclk,modelcontrol,highfreq,cathodes,test_AN);
endmodule
/*
module testbench;
reg [1:0] testmode;
reg sysclk;
reg modelcontrol;
wire highfreq;
wire [7:0] cathodes;
wire [3:0]AN;
test t1(testmode,sysclk,modelcontrol,highfreq,cathodes,AN);
initial begin
    testmode<=2'b00;
    sysclk<=0;
    modelcontrol<=0;
end
initial begin
    forever #5 sysclk<=~sysclk;
end
endmodule*/
```

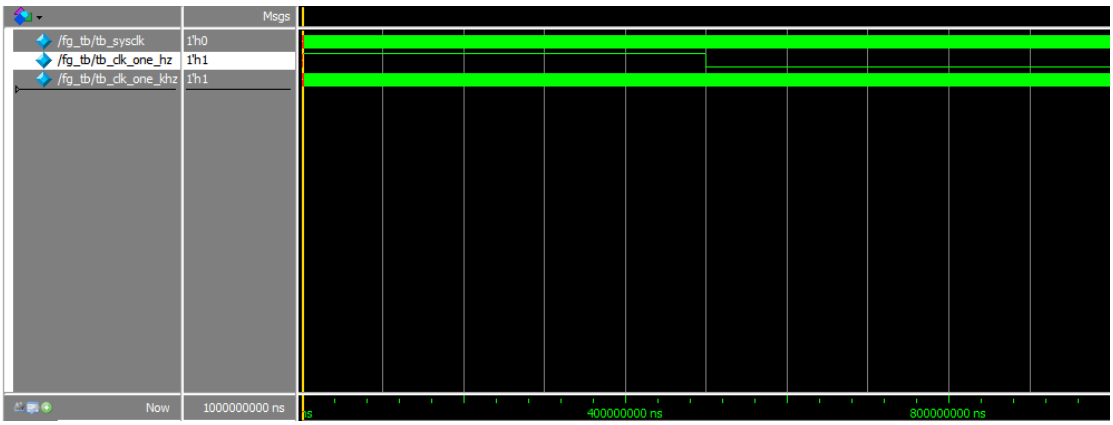
4. 端口约束文件：constrains.xdc

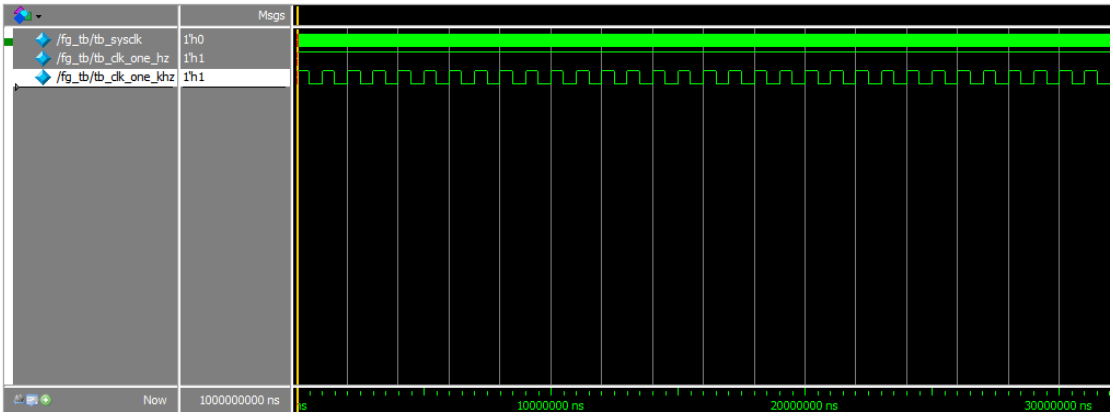
```
set_property PACKAGE_PIN W13 [get_ports {modelcontrol}]
set_property PACKAGE_PIN V16 [get_ports {testmode[1]}]
set_property PACKAGE_PIN V17 [get_ports {testmode[0]}]
set_property PACKAGE_PIN V14 [get_ports {highfreq}]
set_property PACKAGE_PIN V7 [get_ports {cathodes[7]}]
set_property PACKAGE_PIN W7 [get_ports {cathodes[6]}]
set_property PACKAGE_PIN W6 [get_ports {cathodes[5]}]
set_property PACKAGE_PIN U8 [get_ports {cathodes[4]}]
set_property PACKAGE_PIN V8 [get_ports {cathodes[3]}]
```

```
set_property PACKAGE_PIN U5 [get_ports {cathodes[2]}]
set_property PACKAGE_PIN V5 [get_ports {cathodes[1]}]
set_property PACKAGE_PIN U7 [get_ports {cathodes[0]}]
set_property PACKAGE_PIN W4 [get_ports {test_AN[0]}]
set_property PACKAGE_PIN V4 [get_ports {test_AN[1]}]
set_property PACKAGE_PIN U4 [get_ports {test_AN[2]}]
set_property PACKAGE_PIN U2 [get_ports {test_AN[3]}]
set_property PACKAGE_PIN W5 [get_ports sysclk]

create_clock -period 10.000 -name CLK -waveform {0.000 5.000}
[get_ports sysclk]
#set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets sysclk]

set_property IOSTANDARD LVCMOS33 [get_ports sysclk]
set_property IOSTANDARD LVCMOS33 [get_ports {modelcontrol}]
set_property IOSTANDARD LVCMOS33 [get_ports {testmode[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {testmode[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {highfreq}]
set_property IOSTANDARD LVCMOS33 [get_ports {cathodes[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {cathodes[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {cathodes[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {cathodes[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {cathodes[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {cathodes[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {cathodes[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {cathodes[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {test_AN[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {test_AN[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {test_AN[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {test_AN[3]}]
```

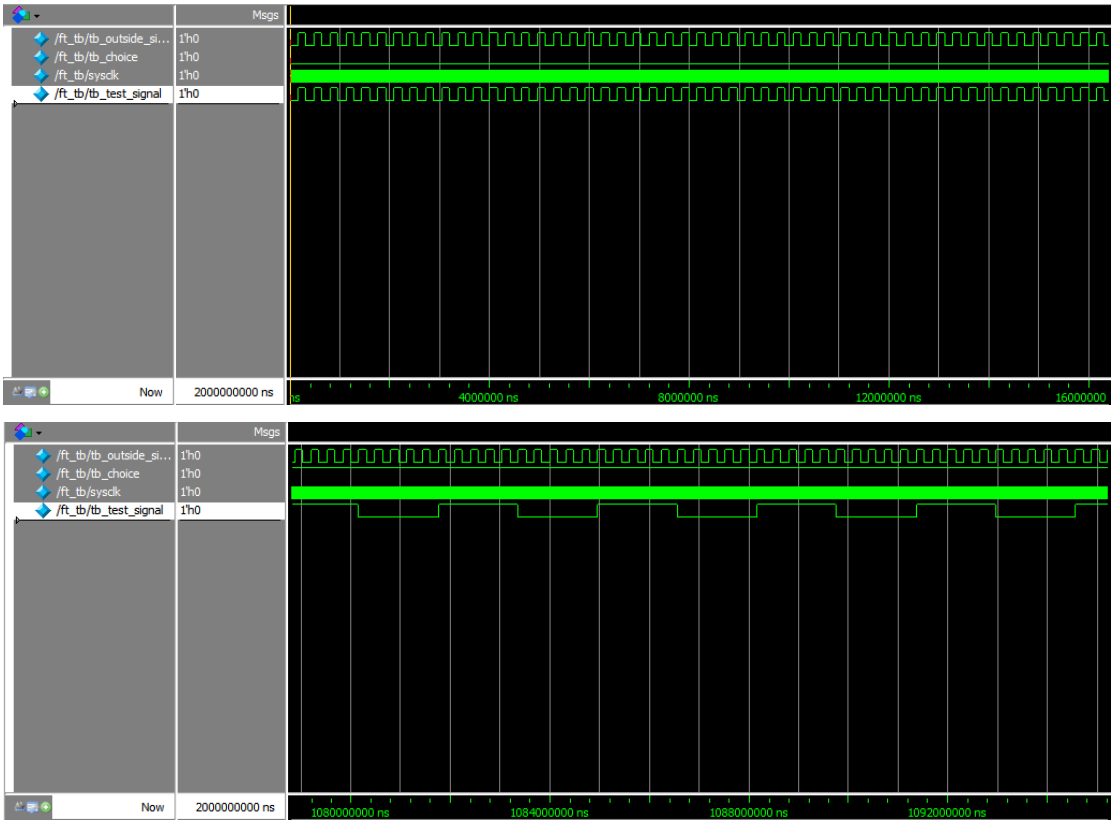
四、 仿真结果与分析

1. 1Hz 和 1kHz 信号产生模块

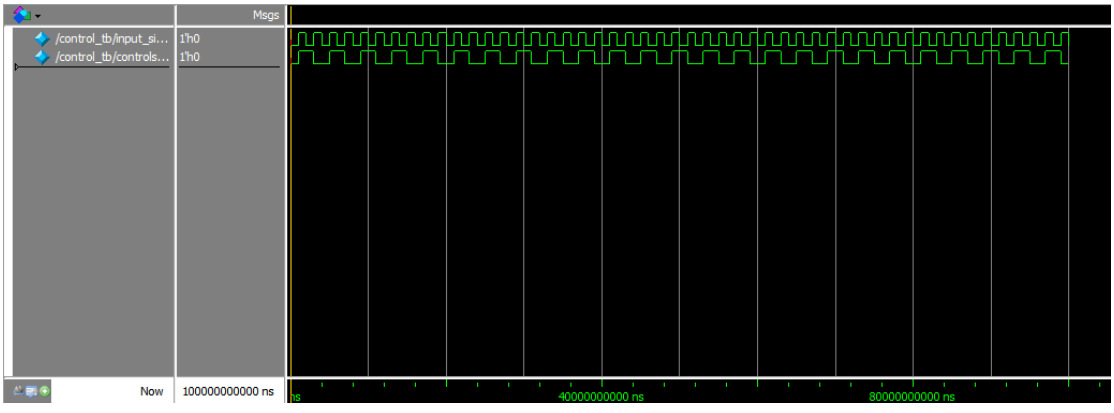




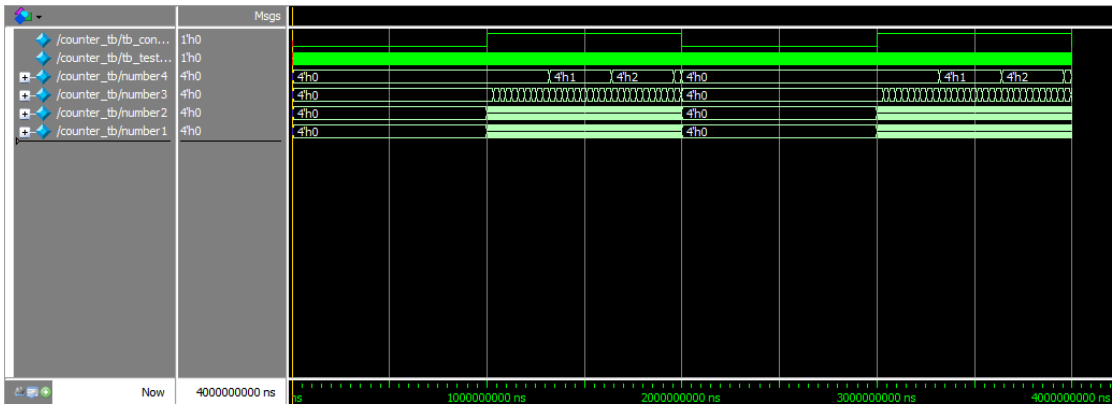
2. 频率量程模块（测试频率为 3125Hz）



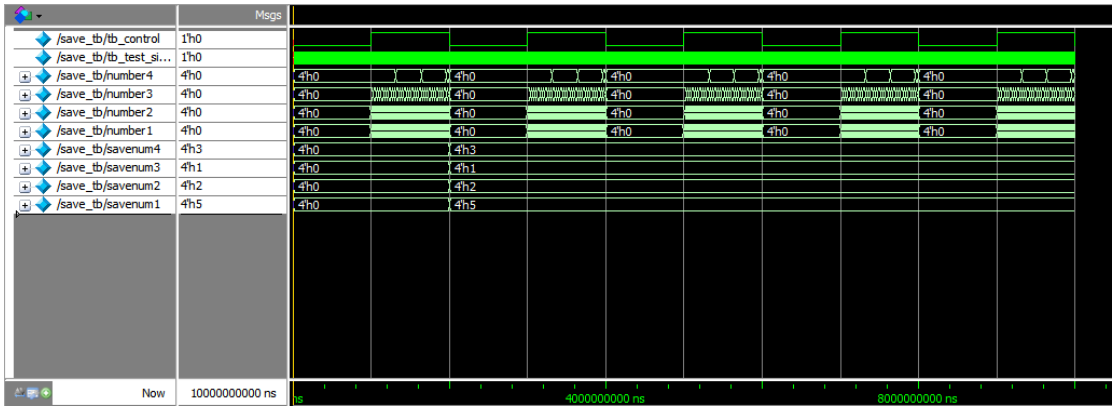
3. 控制信号产生模块



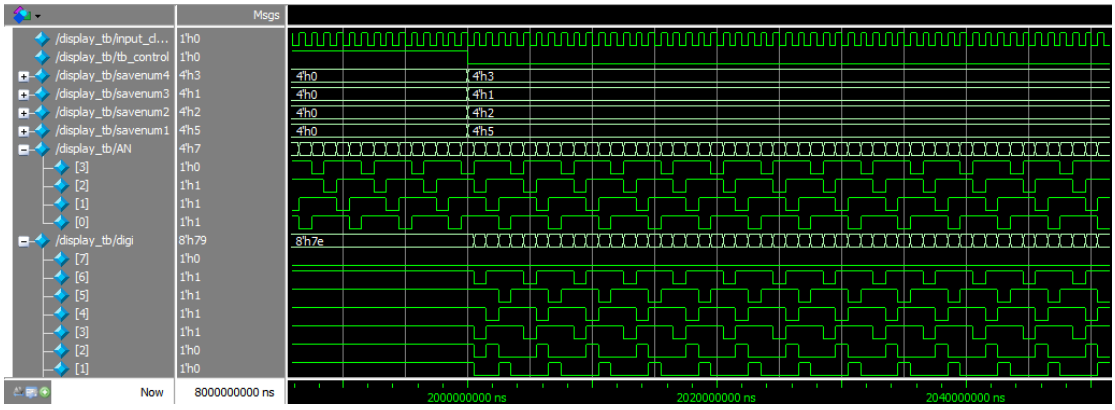
4. 十进制计数模块（测试频率为 3125Hz）



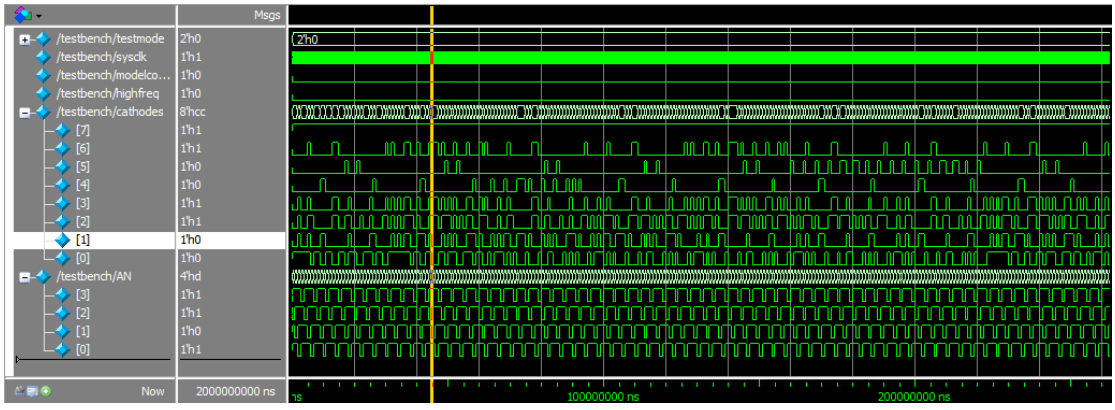
5. 锁存模块（测试频率为 3125Hz）

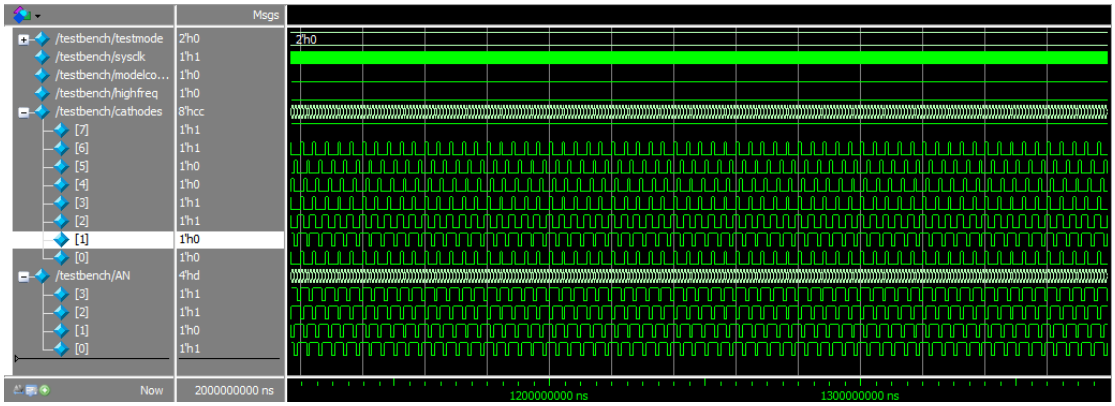


6. 译码显示模块



7. 最终效果（测试频率为 3125Hz）

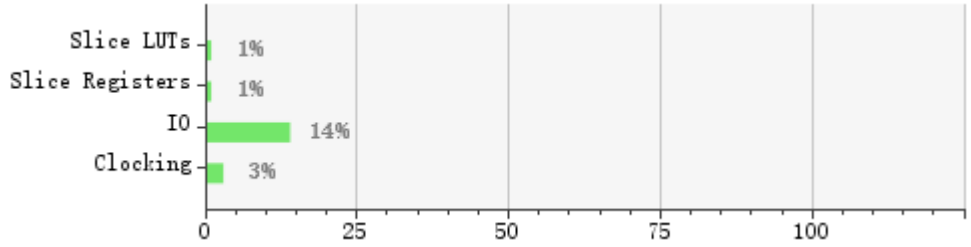




五、 综合报告

1. 面积报告

Resource	Utilization	Available	Utilizat...
Slice LUTs	194	20800	0.93
Slice Registers	112	41600	0.27
IO	17	120	14.17
Clocking	1	32	3.12



2. 时序报告

Name	Waveform	Period (ns)	Frequency (MHz)
CLK	{0.000 5.000}	10.000	100.000

六、 硬件调试情况

首先将 SW1 和 SW0 置为 00，SW7 置为 0。当程序开始运行之后，等待一秒之后，实验平台上的四位数字显示为 3125，此后数字一直不变。将 SW1 和 SW0 置为 01/10，等待一秒之后实验平台上的四位数字显示为 6250/0050。将 SW1 和 SW0 置为 11，并将 SW7 置为 1，此时与 SW7 对应的 LED 灯亮起，等待一秒之后实验平台上的数字显示为 1250。将 SW1 和 SW0 置为 00，SW7 保持为 1，等待一秒之后实验平台的数字显示为 0312。

七、 实验总结与体会

此次实验与前两次实验有所不同，前两次作为熟悉 Verilog 的实验，内容都不算复杂。但是这次实验真正的开始设计具有一定功能的模块。在整个频率计写好之后，仿真波形没有问题，但是将程序烧到实验平台之后，出现了四位显示数字全为 0 的情况。在翻阅了老师上课的讲义之后，发现自己有的模块使用了双沿触发的敏感表，而在实际电路综合中不允许出现双沿触发的触发器。在更改了双沿触发的逻辑之后，还是有四位显示数字全为 0 的情况。

仔细检查代码并翻阅讲义之后发现代码中出现了 if 分支逻辑中有部分分支没有赋值的情况，而这种情况也容易导致仿真和综合结果不一样的情况。修改了相应代码之后，程序正常工作。之前的几次实验由于代码量比较小，没有意识到自己在 Verilog 的仿真和综合中有这么多没有掌握的知识。通过这次实验，我对代码逻辑和电路综合之间的关系理解的更加深刻，同时也更加深刻的理解了 Verilog 作为电路设计语言的作用。这次实验中老师和助教也给了我很大的帮助，非常感谢老师和助教的悉心指导。