《数字逻辑》实验报告

姓名		年级	2022 级
学号		专业、班级	计算机科学与技术（卓越） 1 班

实验名称	实验七 数字钟设计		
实验时间	2023.10.26	实验地点	计算机机房 DS1410
实验成绩		实验性质	☐验证性 ☑设计性 ☐综合性

教师评价：

☐算法/实验过程正确； ☐源程序/实验内容提交 ☐程序结构/实验步骤合理；

☐实验结果正确； ☐语法、语义正确； ☐报告规范；

评语：

评价教师签名（电子签名）：

一、实验目的

通过实验，充分理解和掌握数字钟的原理，理解多进制计时器和数码管的工作原理，学会设计自己的数字钟。

二、实验项目内容

1、实现一个六十进制数字时钟，秒到 60 则归零重加，同时让分加 1，分加到 60 归零重加，并让小时加 1，小时加到 24 归零重加。

2、设计用于数字钟的计数器（60 进制，24 进制）并编写仿真文件，观察其波形图并验证期正确性。

3、将设计好的计数器连接数码管并显示数值，数码管 1,0 显示秒值，数码管 2,3 显示分值，数码管 4,5 显示小时（可以十六进制形式显示在 led 灯上）。

4、综合、实验、生成 bit 流，下载到开发板进行验证。

三、实验设计（实验原理、真值表、原理图等）

verilog HDL 语言编写方法：

我们将时钟的编写分成了三个模块:（1）实现 1s 的分频器；（2）时钟的时分秒进位的模拟文件编写；（3）数码管的输出实现代码编写

设计文件模块如下：

四、实验过程或算法(关键步骤、核心代码注解等)

verilog 语言设计方法：

（1）为了模拟现实中的 1s 我们可以采用对周期信号进行计数的方式来实现，很容易就想到 clk 信号，而时钟信号是周期激励，显然我们不可能手动对它完成输入，那这时候我们就需要接到 FPGA 开发板上自带的时钟信号激励源上。以我手上的 BASYS 3 为例，自带 100MHz 的 clk 激励，也就是 1ns 产生一次 clk 信号，那我们就可以通过对 clk 计数，当 counter 达到 100_000_000 次时，时长就达到了 1s。输出就用 0 和 1 区分。

具体代码如下：

```
23  module clk_1s(
24          input clk,
25          input rstn,
26          output flag_1s
27      );
28      parameter num_1s=100_000_000;
29      reg [26:0]cnt_1s;
30      always@(posedge clk or negedge rstn)
31      begin
32      if(!rstn)begin
33      cnt_1s<=0;
34      end
35      else
36      begin
37      if(cnt_1s==num_1s) cnt_1s<=0;
38      else cnt_1s<=cnt_1s+1;
39      end
40      end
41      assign flag_1s=(cnt_1s==num_1s)?1:0;
42  endmodule
43
```

（2）第二步我们来处理时钟的模拟，这其中的进位问题我们只需要用 if-else 语句来实现就可以了，具体代码如下：

```
module clock(
        input clk,
        input rstn,
        input flag_1s,
        output reg[3:0] data0,
        output reg[3:0] data1,
        output reg[3:0] data2,
        output reg[3:0] data3,
        output reg[3:0] data4,
        output reg[3:0] data5
    );
 reg [5:0] second;
 reg [5:0] minute;
 reg [4:0] hour;
always@(posedge clk or negedge rstn) begin
    if(!rstn)begin
        second<=0;
    end else begin
        if(second==59&&flag_1s)begin
          second<=0;
        end else if(flag_1s)begin
            second=second+1;
        end else second<=second;
    end
end
always@(posedge clk or negedge rstn) begin
    if(!rstn)minute<=0;
    else begin
    if(minute==59&&second==59&&flag_1s) minute<=0;
    else if(second==59&&flag_1s) minute<=minute+1;
    else minute<=minute;
    end
end
always@(posedge clk or negedge rstn) begin
    if(!rstn) hour<=0;
    else begin
        if(hour==23&&minute==59&second==59) hour<=0;
        else if(minute==59&&second==59&&flag_1s) hour<=hour+1;
        else hour<=hour;
    end
end
always@(posedge clk or negedge rstn) begin
    if(!rstn) begin
        data0 <= 0;
        data1 <= 0;
        data2 <= 0;
        data3 <= 0;
        data4 <= 0;
        data5 <= 0;
    end
    else begin
      data0 <= hour/10;
      data1 <= hour%10;
      data2 <= minute/10;
      data3 <= minute%10;
      data4 <= second/10;
      data5 <= second%10;
    end
end
endmodule
```

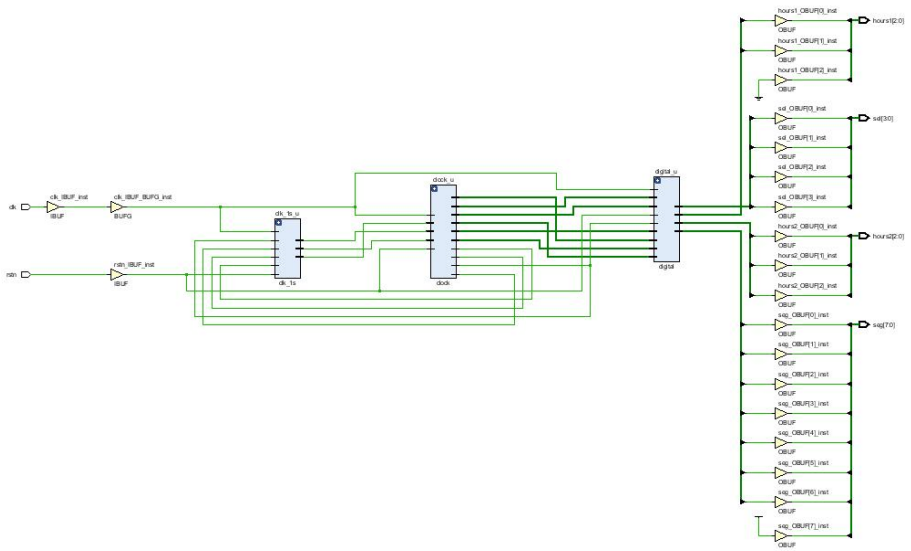
（3）第三步我们来编写数码管的显示代码：

```
module digital(
  input clk,
  input rstn,
  input [3:0]data0,
  input [3:0]data1,
  input [3:0]data2,
  input [3:0]data3,
  input [3:0]data4,
  input [3:0]data5,
  output reg[7:0]seg,
  output reg[3:0]sel,
  output reg[2:0]hours1,
  output reg[2:0]hours2
);
reg[15:0]cnl;
reg clk1k;
always@(posedge clk or negedge rstn)//分频
  begin
    if(!rstn)
    begin
      cnl<=0;
      clk1k<=0;
    end
    else if(cnl>=49999)
    begin
      clk1k<=!clk1k;
      cnl<=0;
    end
    else cnl<=cnl+1;
  end
reg [3:0]tub;//中间变量
reg [2:0]state;//状态变量
always@(posedge clk1k or negedge rstn)
  begin
    if(!rstn)
    begin
      tub<=0;
      state<=0;
      sel<=0;
      hours1<=0;
      hours2<=0;
    end
    else
    begin
      case(state)//每种状态的数码管选择，0111即第四个数码管，
        0:begin hours1<=data0;sel<=4'b1111;state<=1;end
        1:begin hours2<=data1;sel<=4'b1111;state<=2;end
        2:begin tub<=data2;sel<=4'b0111;state<=3;end
        3:begin tub<=data3;sel<=4'b1011;state<=4;end
        4:begin tub<=data4;sel<=4'b1101;state<=5;end
        5:begin tub<=data5;sel<=4'b1110;state<=0;end
        default:state<=0;
      endcase
    end
  end
always@(*)//数码管数字显示
  if(!rstn)
    seg<=8'b1100_0000;
  else
    case(tub)
      0:seg<=8'b1100_0000;
      1:seg<=8'b1111_1001;
      2:seg<=8'b1010_0100;
      3:seg<=8'b1011_0000;
      4:seg<=8'b1001_1001;
      5:seg<=8'b1001_0010;
      6:seg<=8'b1000_0010;
      7:seg<=8'b1111_1000;
      8:seg<=8'b1000_0000;
      9:seg<=8'b1001_0000;
      default:seg<=8'b1100_0000;
    endcase
endmodule
```

（4）最后我们使用一个顶层模块来将三个子模块串起来：

```
module top_tb();
reg clk;
reg rstn;
wire [7:0]seg;
wire [3:0]sel;
wire [2:0]hours1;
wire [2:0]hours2;
top top(
   .clk(clk),
   .rstn(rstn),
   .seg(seg),
   .sel(sel),
   .hours1(hours1),
   .hours2(hours2)
    );
    initial begin
           clk=0;rstn=0;
     #17    rstn=1;
     #10  $stop;
end
always#1
clk=~clk;
endmodule
```

（5）随后我们进行 RTL 分析，电路图下：

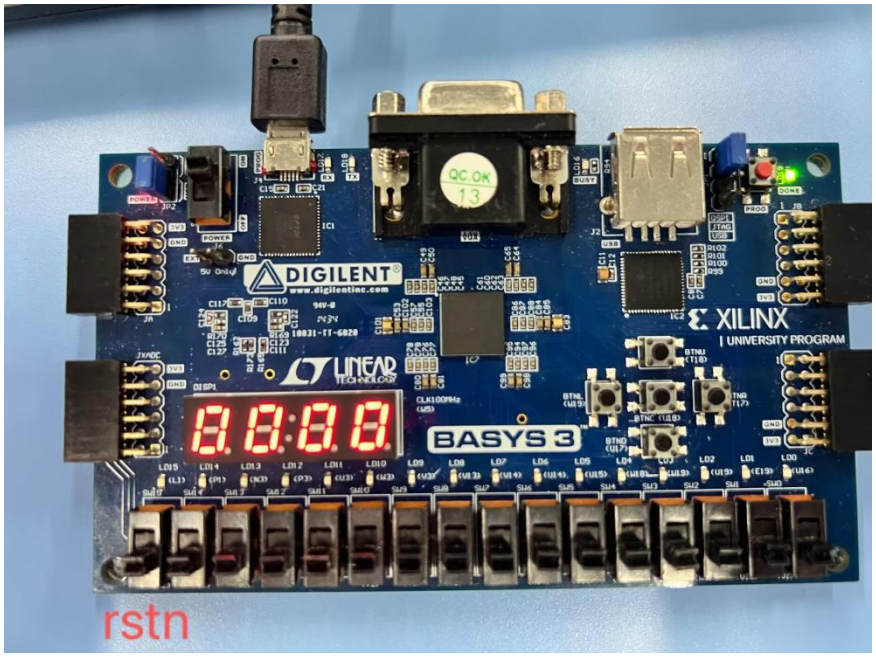


（6）综合、实现、并进行管脚分配：

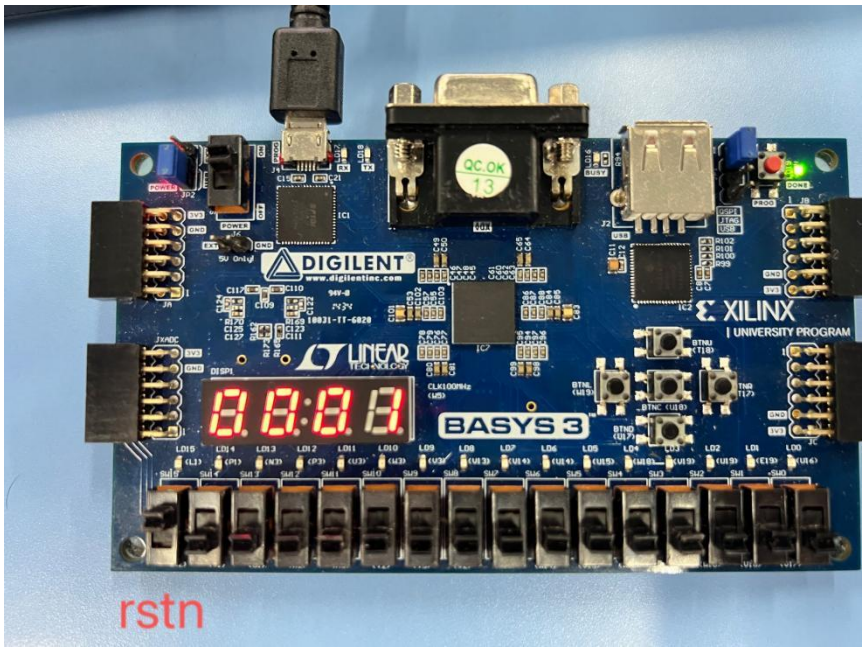
```
set_property PACKAGE_PIN W5 [get_ports clk]
set_property PACKAGE_PIN R2 [get_ports rstn]
set_property PACKAGE_PIN P3 [get_ports {hours1[2]}]
set_property PACKAGE_PIN U3 [get_ports {hours1[1]}]
set_property PACKAGE_PIN W3 [get_ports {hours1[0]}]
set_property PACKAGE_PIN V3 [get_ports {hours2[2]}]
set_property PACKAGE_PIN V13 [get_ports {hours2[1]}]
set_property PACKAGE_PIN V14 [get_ports {hours2[0]}]
set_property PACKAGE_PIN V7 [get_ports {seg[7]}]
set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
set_property PACKAGE_PIN W4 [get_ports {sel[3]}]
set_property PACKAGE_PIN V4 [get_ports {sel[2]}]
set_property PACKAGE_PIN U4 [get_ports {sel[1]}]
set_property PACKAGE_PIN U2 [get_ports {sel[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {sel[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sel[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sel[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sel[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {hours2[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {hours2[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {hours2[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {hours1[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {hours1[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {hours1[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports rstn]
```

（7）生成比特流，打开硬件程序并打开硬件目标，在开发板上验证自己的逻辑程序。

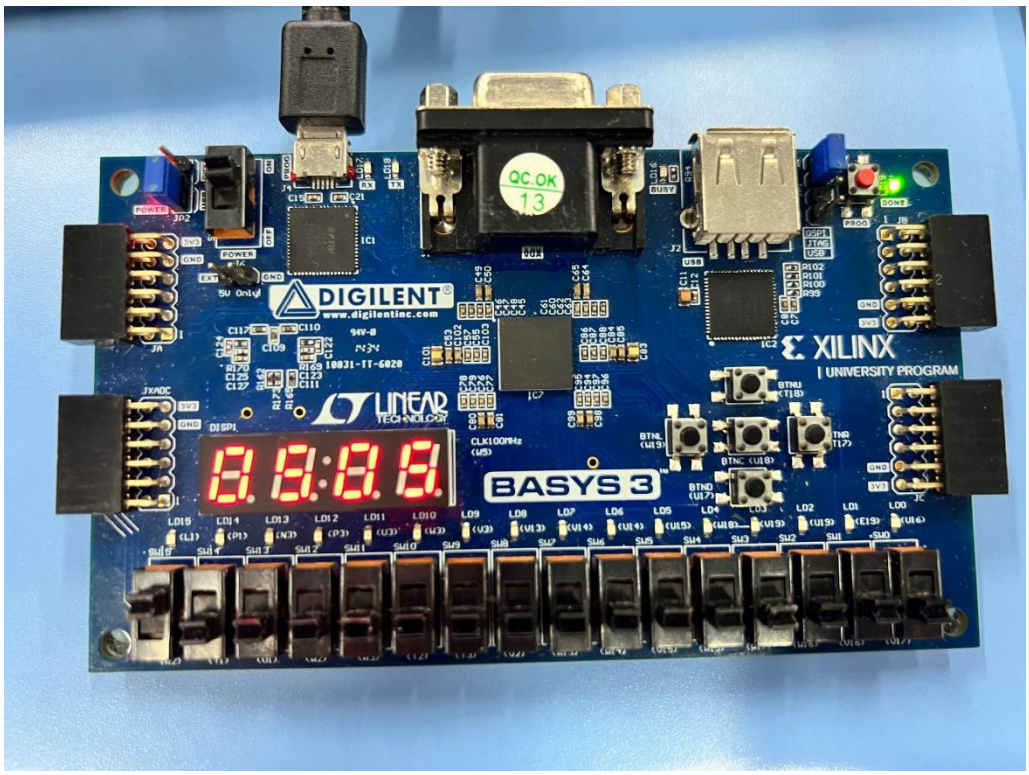
rstn 未拨上去时，时钟为置零状态：



rstn 拨上去之后，时钟开始正常工作：



时钟正常工作时的状态（显示为五分五秒）：



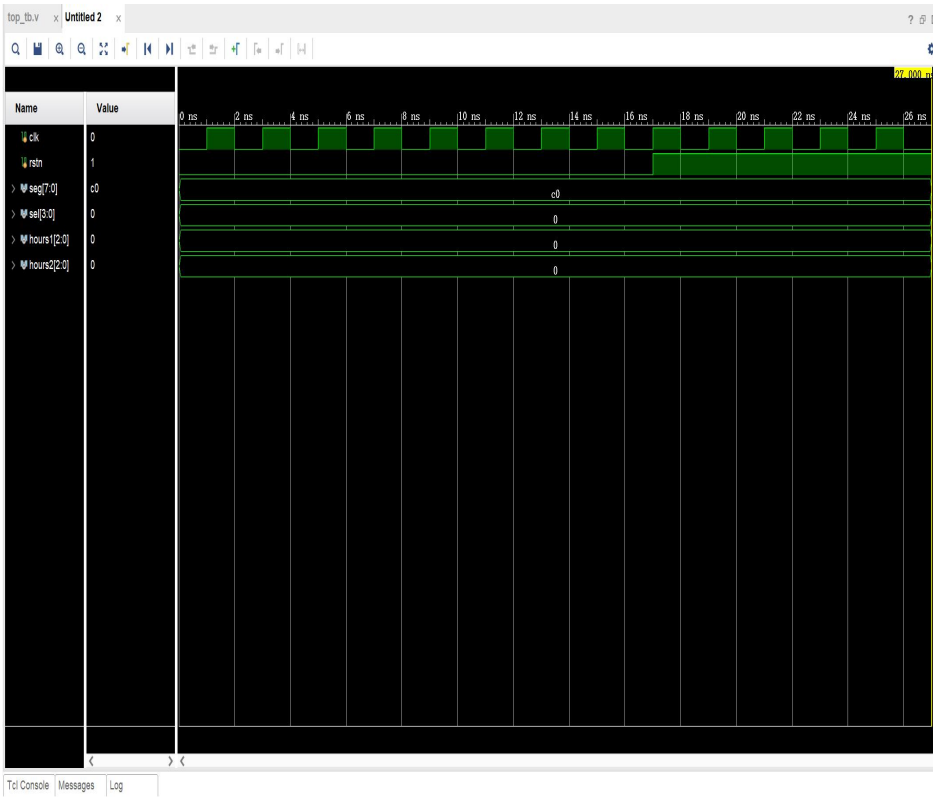
经过验证，时钟运行结果完美符合预期。

五、实验过程中遇到的问题及解决情况(主要问题及解决情况)

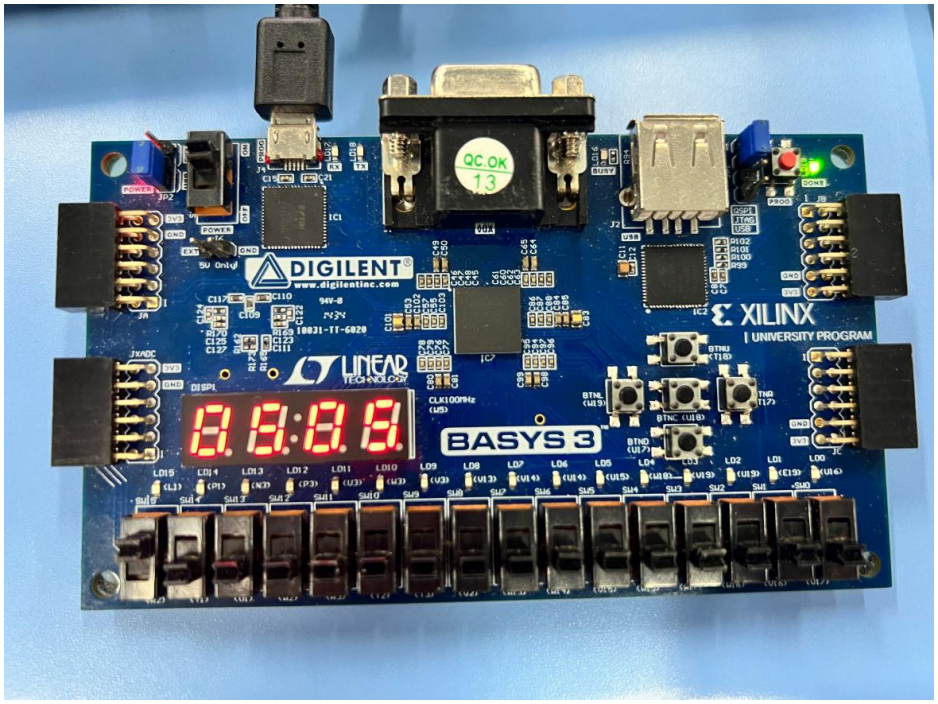
首先，我们需要找到控制数码管的时间在人眼可滞留范围的同时保证数据不会更新的具体时间，虽然给了具体的 1khz 到 60hz 但是实际操控当中仍然经常出现每个显示数字红光重合的现象，最后我们经过不断尝试把频率分频到了 500hz 成功解决了这个问题。

六、实验结果及分析和（或）源程序调试过程

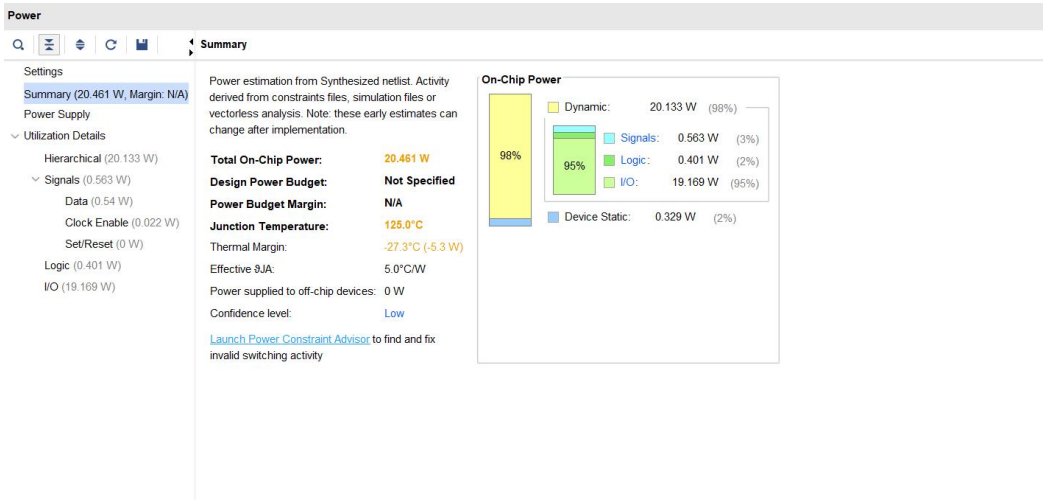
（1）通过 verilog 语言程序设计的设计文件在仿真和 最后实现的过程中得到了预期结果



（2）在仿真测试成功后，完成综合、实现、引脚分配的过程，最后成功 生成比特流并将其下载到开发板上。



通过在开发板上进一步验证自己的程序，可以得出本实验通过 verilog 语言设计的方式成功完成了数字钟。



硬件所产生的功耗（device static）为 0.062W，而由于数字钟主要是在上板验证后七段数码管发光所产生能耗，并没有太多其他产生能耗的地方，所以 I/O 功耗占比非常高，达到了 19.169W，占比 95%

七、小组分工情况说明

：编写 verilog 语言设 计的仿真代码，并进行仿真验证。共同分析在 RTL 分析、逻辑 资源占用、功耗上的问题。

：编写数字钟语言设计代码和仿真代码，并进行仿真验证。共同分析在 RTL 分析、逻辑 资源占用、功耗上的问题。

：编写通过 verilog 语言设计实现实验的设计文件代码完成引脚 绑定和上板验证。共同分析在 RTL 分析、逻辑 资源占用、功耗上的问题。