华中科技大学
人工智能与自动化学院

多功能数字钟设计

彭杨哲

U201914634

2021 年 6 月 3 日

1 实验目的

- 掌握可编程逻辑器件的应用开发技术——设计输入、编译、仿真和器件编程。
- 熟悉 EDA 软件的使用。
- 掌握 Verilog HDL 设计方法。
- 分模块、分层次数字系统设计。

2 实验元器件

ISE 14.3 软件
FPGA 实验开发装置

Table 1: 实验元器件

3 实验原理

3.1 数字钟电路的设计思路

3.1.1 时分秒计数器的设计

分和秒计数器都是模 M=60 的计数器, 时计数器为一个 24 进制计数器.

即当数字钟运行到 23 时 59 分 59 秒时, 秒的个位计数器再输入一个秒脉冲时, 数字钟应自动显示为 00 时 00 分 00 秒

数字钟的整体设计框图如图1所示

可以将整个数字钟的设计分为几个模块:

- 时钟分频模块
- 计时, 校时模块
- 7 段数码管动态显示模块

而提高部分的内容则在基本功能的基础上进行相应的添加即可

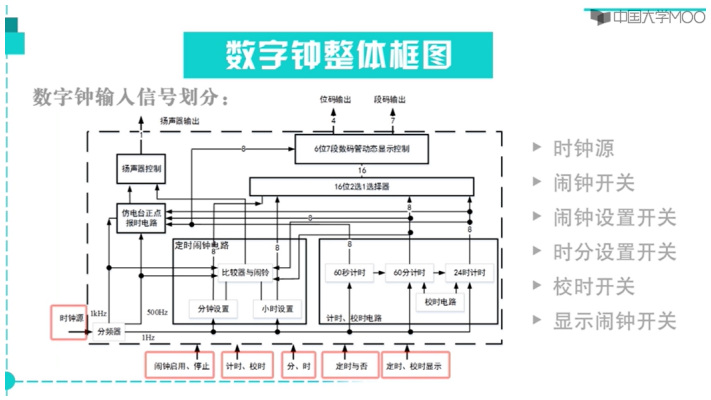


Figure 1: 数字钟整体框图

3.1.2 时钟分频模块的设计

在此模块中, 通过对 B8 管脚对应的板载 50MHz 晶振进行分频, 可以得到所需脉冲信号. 首先, 对 50MHz 信号进行 50K 次分频, 得到 1KHz 扫描信号, 此信号将用于数码管动态显示模块中进行动态显示; 然后再对得到的 1KHz 扫描信号进行 1K 次分频, 得到 1Hz 扫描信号

3.1.3 计时, 校时模块的设计

此模块的设计框图如图2所示, 其中, 秒, 分计时, 时计时均为 8 位 8421BCD 码表示的数字, 用于之后的显示模块.

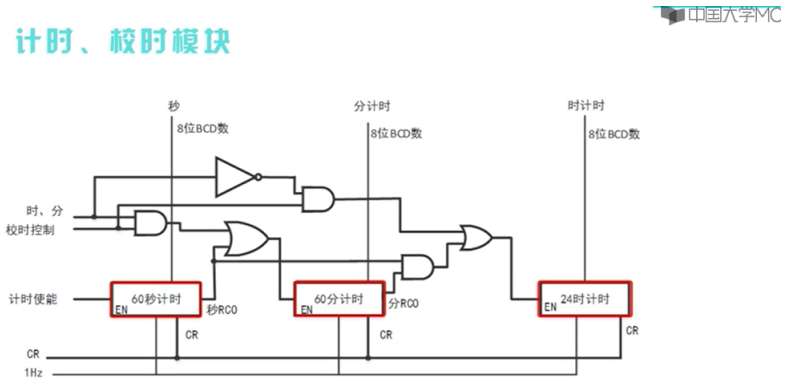


Figure 2: 计时校时模块设计框图

3.1.4 数码管动态显示模块的设计

此模块的设计框图如图3所示, 其中, 为了适应本实验中所提供的 FPGA 板载资源, 需要进行适当的调整, 将模六计数器变为模四计数器, 6 路 4 位复用器变为 4 路 4 位复用器, 3-8 译码器的输出只连接 4 个 7 段数码管.

其中 3-8 译码器 (也可以只使用一个 2-4 译码器) 的输出作为位码, 用于选择哪一个 7 段数码管亮, 而 BCD-7 段数码管译码器的输出作为段码, 用于控制 7 段数码管如何亮.

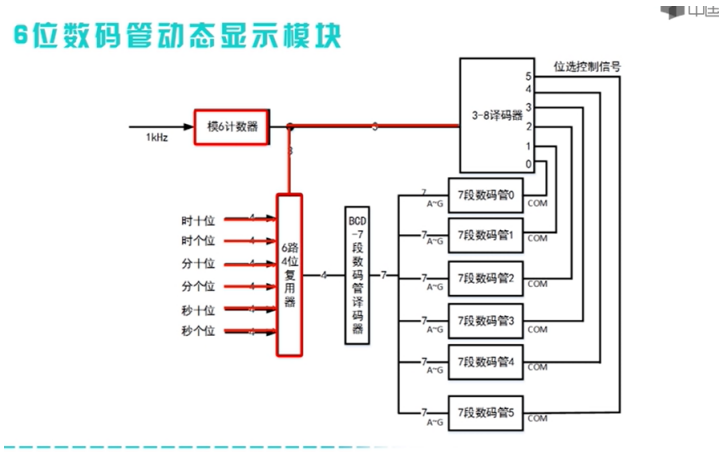


Figure 3: 动态显示模块设计框图

3.1.5 12/24 小时进制切换模块的设计

本模块的设计框图如图4所示, 此模块的输入为 bcd 码表示的小时数字和当前是否为 24 小时进制, 输出为当前是否为 12 小时进制下的早上和经过调整后的小时输出数字.

4 实验内容

1. 基本功能 (必做)
   - 显示时分秒
   - 小时为 24 进制, 分秒为 60 进制
   - 能调整小时, 分钟时间
2. 提高部分 (选做)
   小时为 12/24 进制可切换

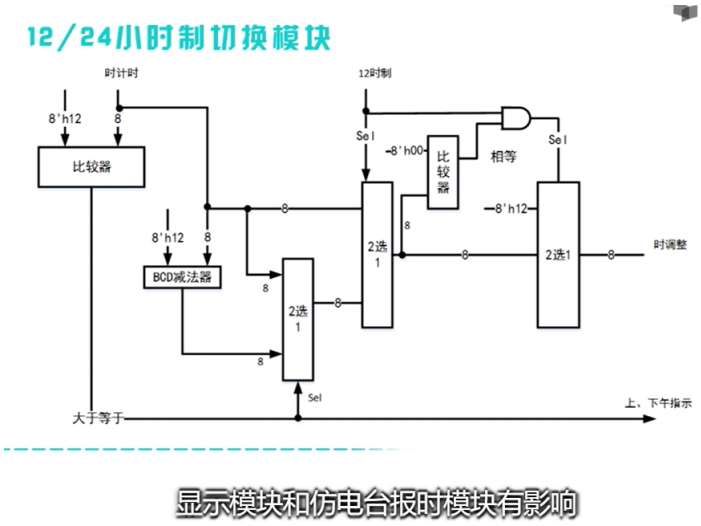


Figure 4: 12/24 小时进制切换模块设计框图

5 实验结果及分析

5.1 实验源码

5.1.1 最顶层模块

```
`timescale 1ns / 1ps
module top(
    input clk_50m,
    input cr,
    input en_clock,
    input hour_adjust,
    input min_adjust,
    input is_24_hours,
    output [3:0] pos,
    output [6:0] seg,
   output [6:0] second_out,
    output is_am
    );
  wire clk_1k,clk_1hz;
  clock_div u_clock(clk_50m,cr,clk_1k,clk_1hz);
  wire [3:0] bcd_su,bcd_st,bcd_mu,bcd_mt,bcd_hu,bcd_ht;
  clock_adjust u_clk_adjust(clk_1hz,hour_adjust,
  min_adjust,cr,bcd_su,bcd_st,bcd_mu,bcd_mt,bcd_hu,bcd_ht);
  assign second_out={bcd_st,bcd_su};
  wire [3:0] bcd_ht_disp,bcd_hu_disp;
  day24_12 u_day24_12(bcd_ht,bcd_hu,is_24_hours,
  is_am,bcd_ht_disp,bcd_hu_disp);
  scan_disp u_scan_disp(clk_1k,cr,en_clock,bcd_ht_disp,
```

```
  bcd_hu_disp,bcd_mt,bcd_mu,seg,pos);
endmodule
```

5.1.2 分频模块

```
`timescale 1ns / 1ps
module clock_div(
    input clk_50m,
   input cr,
    output reg clk_1k,
    output reg clk_1hz
    );
   reg [15:0] counter_1k;
   reg [8:0] counter_1hz;

   always@(posedge clk_50m or negedge cr)
   begin
    if(!cr)
      begin
        counter_1k<=16'h0000;
        clk_1k <=1'b0;
      end
    else
      begin
        if (counter_1k==(16'd24999))
          begin
            counter_1k<=0;
            clk_1k<=~clk_1k;
          end
        else
          counter_1k<=counter_1k+1;
      end
   end

   always@(posedge clk_1k or negedge cr)
   begin
    if(!cr)
      begin
        counter_1hz<=9'd0;
        clk_1hz<=1'b0;
      end
    else
      begin
        if(counter_1hz==(9'd4999))
          begin
            counter_1hz<=0;
            clk_1hz<=~clk_1hz;
          end
        else
```

```
          counter_1hz<=counter_1hz+1;
      end
   end

endmodule
```

5.1.3 计时调时模块

```
`timescale 1ns / 1ps
module clock_adjust(
    input clk,
    input hour_adjust,
    input min_adjust,
    input cr,
    output [3:0] bcd_su,
    output [3:0] bcd_st,
    output [3:0] bcd_mu,
    output [3:0] bcd_mt,
    output [3:0] bcd_hu,
    output [3:0] bcd_ht
    );
  wire m_en,h_en,rco_s,rco_m;
  assign m_en=rco_s | min_adjust;
  assign h_en=(rco_s&rco_m) | hour_adjust;

  c60 second(clk,1'b1,cr,rco_s,bcd_st,bcd_su);
  c60 minute(clk,m_en,cr,rco_m,bcd_mt,bcd_mu);
  c24 hour(clk,h_en,cr,bcd_hu,bcd_ht);

endmodule
```

5.1.4 7 段数码管动态显示模块

```
`timescale 1ns / 1ps
module scan_disp(
    input clk,
    input cr,
    input en,
    input [3:0] ch0,
    input [3:0] ch1,
    input [3:0] ch2,
    input [3:0] ch3,
    output [6:0] seg,
    output [3:0] pos
    );
  wire rco;
  wire [3:0] bcd_sel,bcd_data;
```

```
  c4 u_c4(clk,en,cr,rco,bcd_sel);
  mux4_1 u_mux(ch0,ch1,ch2,ch3,bcd_sel,bcd_data);
  posdecode u_pos(bcd_sel,pos);
  bcd_7seg u_seg(bcd_data,seg);

endmodule
```

5.1.5 选做部分:12-24 进制切换模块

```
`timescale 1ns / 1ps
module day24_12(
    input [3:0] bcd_ht,
    input [3:0] bcd_hu,
    input is_24_hours,
    output is_am,
    output [3:0] bcd_ht_out,
    output [3:0] bcd_hu_out
    );

  wire [7:0] hour,hour_out;
  assign hour={bcd_ht,bcd_hu};
  assign hour_out={bcd_ht_out,bcd_hu_out};
  assign is_am=(hour<8'h13 & ~is_24_hours);
  assign need_substract=(hour>8'h12 & ~is_24_hours);
  assign bcd_ht_out=bcd_ht-need_substract*1;
  assign bcd_hu_out=bcd_hu-need_substract*2;

endmodule
```

5.1.6 管脚约束文件

```
NET "clk_50m" LOC = B8;
NET "seg[6]" LOC = M12;
NET "seg[5]" LOC = L13;
NET "seg[4]" LOC = P12;
NET "seg[3]" LOC = N11;
NET "seg[2]" LOC = N14;
NET "seg[1]" LOC = H12;
NET "seg[0]" LOC = L14;

NET "pos[3]" LOC = K14;
NET "pos[2]" LOC = M13;
NET "pos[1]" LOC = J12;
NET "pos[0]" LOC = F12;

NET "cr" LOC = E2;
NET "en_clock" LOC = F3;
```

```
NET "hour_adjust" LOC = G3;
NET "is_24_hours" LOC = K3;
NET "min_adjust" LOC = B4;
NET "second_out[6]" LOC = P4;
NET "second_out[5]" LOC = N4;
NET "second_out[4]" LOC = N5;
NET "second_out[3]" LOC = P6;
NET "second_out[2]" LOC = P7;
NET "second_out[1]" LOC = M11;
NET "second_out[0]" LOC = M5;
NET "is_am" LOC=G1;
#Created by Constraints Editor (xc3s100e-cp132-4) - 2021/05/28
NET "clk_50m" TNM_NET = "clk_50m";
TIMESPEC TS_clk_50m = PERIOD "clk_50m" 20 ns HIGH 50 %;

# PlanAhead Generated physical constraints
```

5.2 仿真测试代码

5.2.1 秒计数器仿真

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

  // Inputs
  reg clk;
  reg en;
  reg cr;

  // Outputs
  wire rco;
  wire [3:0] bcd_t;
  wire [3:0] bcd_u;

  // Instantiate the Unit Under Test (UUT)
  c60 uut (
    .clk(clk),
    .en(en),
    .cr(cr),
    .rco(rco),
    .bcd_t(bcd_t),
    .bcd_u(bcd_u)
  );
   parameter PERIOD = 10;

   always begin
      clk = 1'b0;
      #(PERIOD/2) clk = 1'b1;
      #(PERIOD/2);
```

```
    end

  initial begin
    // Initialize Inputs
    clk = 0;
    en = 0;
    cr = 0;

    // Wait 100 ns for global reset to finish
    #100;

    // Add stimulus here
    en=1;
    cr=1;
    #5000;
  end

endmodule
```

5.2.2 24 进制计数器仿真

```
`timescale 1ns / 1ps
module c24_sim;

  // Inputs
  reg clk;
  reg en;
  reg cr;

  // Outputs
  wire [3:0] bcd_u;
  wire [3:0] bcd_t;

  // Instantiate the Unit Under Test (UUT)
  c24 uut (
    .clk(clk),
    .en(en),
    .cr(cr),
    .bcd_u(bcd_u),
    .bcd_t(bcd_t)
  );
   parameter PERIOD = 10;

   always begin
      clk = 1'b0;
      #(PERIOD/2) clk = 1'b1;
      #(PERIOD/2);
   end

```

```
29    initial begin
30      // Initialize Inputs
31      clk = 0;
32      en = 0;
33      cr = 0;
34
35      // Wait 100 ns for global reset to finish
36      #100;
37
38      // Add stimulus here
39      en=1;
40      cr=1;
41      #5000;
42    end
43
44  endmodule
```

5.2.3 分频模块仿真

```
1   `timescale 1ns / 1ps
2   module clk_div_sim;
3
4     // Inputs
5     reg clk_50m;
6     reg cr;
7
8     // Outputs
9     wire clk_1k;
10    wire clk_1hz;
11
12    // Instantiate the Unit Under Test (UUT)
13    clock_div uut (
14      .clk_50m(clk_50m),
15      .cr(cr),
16      .clk_1k(clk_1k),
17      .clk_1hz(clk_1hz)
18    );
19
20     parameter PERIOD = 10;
21
22     always begin
23        clk_50m = 1'b0;
24        #(PERIOD/2) clk_50m = 1'b1;
25        #(PERIOD/2);
26     end
27
28    initial begin
29      // Initialize Inputs
30      clk_50m = 0;
```

```
    cr = 0;

    // Wait 100 ns for global reset to finish
    #100;

    // Add stimulus here
    cr=1;
    #1000;

  end

endmodule
```

5.2.4 计时，调时模块仿真

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

  // Inputs
  reg clk;
  reg hour_adjust;
  reg min_adjust;
  reg cr;

  // Outputs
  wire [3:0] bcd_su;
  wire [3:0] bcd_st;
  wire [3:0] bcd_mu;
  wire [3:0] bcd_mt;
  wire [3:0] bcd_hu;
  wire [3:0] bcd_ht;

  // Instantiate the Unit Under Test (UUT)
  clock_adjust uut (
    .clk(clk),
    .hour_adjust(hour_adjust),
    .min_adjust(min_adjust),
    .cr(cr),
    .bcd_su(bcd_su),
    .bcd_st(bcd_st),
    .bcd_mu(bcd_mu),
    .bcd_mt(bcd_mt),
    .bcd_hu(bcd_hu),
    .bcd_ht(bcd_ht)
  );
   // Note: CLK must be defined as a reg when using this method

   parameter PERIOD = 10;

```

```
35    always begin
36       clk = 1'b0;
37       #(PERIOD/2) clk = 1'b1;
38       #(PERIOD/2);
39    end
40
41   initial begin
42     // Initialize Inputs
43     clk = 0;
44     hour_adjust = 0;
45     min_adjust = 0;
46     cr = 0;
47
48     // Wait 100 ns for global reset to finish
49     #10;
50
51     // Add stimulus here
52     hour_adjust=0;
53     min_adjust=0;
54     cr=1;
55     #1000000;
56
57     hour_adjust=1;
58     min_adjust=0;
59     #100;
60
61     hour_adjust=0;
62     min_adjust=1;
63     #100;
64
65     hour_adjust=1;
66     min_adjust=1;
67     #100;
68
69   end
70
71 endmodule
```

5.2.5 7 段数码管动态显示

```
1  `timescale 1ns / 1ps
2  module scan_sim;
3
4    // Inputs
5    reg clk;
6    reg cr;
7    reg en;
8    reg [3:0] ch0;
9    reg [3:0] ch1;
```

```
10  reg [3:0] ch2;
11  reg [3:0] ch3;
12
13  // Outputs
14  wire [6:0] seg;
15  wire [7:0] pos;
16
17  // Instantiate the Unit Under Test (UUT)
18  scan_disp uut (
19    .clk(clk),
20    .cr(cr),
21    .en(en),
22    .ch0(ch0),
23    .ch1(ch1),
24    .ch2(ch2),
25    .ch3(ch3),
26    .seg(seg),
27    .pos(pos)
28  );
29   parameter PERIOD = 10;
30
31   always begin
32      clk = 1'b0;
33      #(PERIOD/2) clk = 1'b1;
34      #(PERIOD/2);
35   end
36
37  initial begin
38    // Initialize Inputs
39    clk = 0;
40    cr = 0;
41    en = 0;
42    ch0 = 0;
43    ch1 = 0;
44    ch2 = 0;
45    ch3 = 0;
46
47    // Wait 100 ns for global reset to finish
48    #100;
49
50    // Add stimulus here
51    en=1;
52    cr=1;
53    ch0=0;
54    ch1=1;
55    ch2=2;
56    ch3=3;
57    #10000;
58
59  end
60
```

```
endmodule
```

5.2.6 选做部分：24-12 进制切换

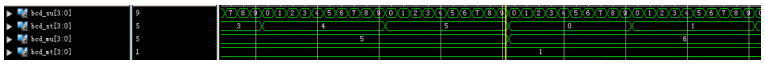
```
`timescale 1ns / 1ps
module day_24_12_sim;

  // Inputs
  reg [3:0] bcd_ht;
  reg [3:0] bcd_hu;
  reg is_24_hours;

  // Outputs
  wire is_am;
  wire [3:0] bcd_ht_out;
  wire [3:0] bcd_hu_out;

  // Instantiate the Unit Under Test (UUT)
  day24_12 uut (
    .bcd_ht(bcd_ht),
    .bcd_hu(bcd_hu),
    .is_24_hours(is_24_hours),
    .is_am(is_am),
    .bcd_ht_out(bcd_ht_out),
    .bcd_hu_out(bcd_hu_out)
  );

  initial begin
    // Initialize Inputs
    bcd_ht = 0;
    bcd_hu = 0;
    is_24_hours = 1;

    // Wait 100 ns for global reset to finish
    #100;

    // Add stimulus here
    bcd_ht=1;
    bcd_hu=1;
    #100;

    bcd_ht=1;
    bcd_hu=2;
    #100;

    bcd_ht=1;
    bcd_hu=3;
    is_24_hours=1;
    #100;
```

```
46
47      is_24_hours=0;
48      #100;
49
50    end
51
52  endmodule
```

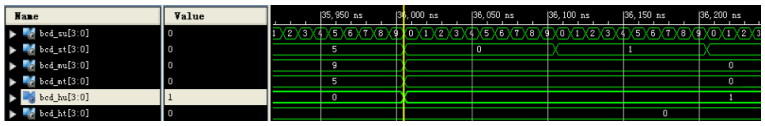
5.3 仿真结果

5.3.1 计时，调时模块仿真结果

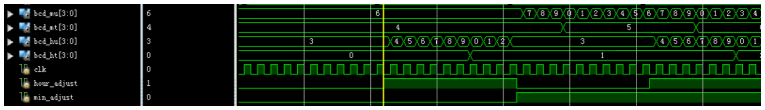
从仿真结果5(a)可以看出，当秒计数计至 59 秒时，分计数器即加一；从仿真结果5(b)可以看出，当分计数器和秒计数器计至 59 分 59 秒时，时计数器即加一；从仿真结果5(c)可以看出，当加入小时调时信号后，小时计数器开始逐秒加一，当加入分钟调时信号后，分钟计数器开始逐秒加一，当两种信号同时作用时，小时计数器和分钟计数器都同时加一



(a) 秒到分跳变



(b) 分到时跳变



(c) 调时功能仿真

Figure 5: 计时调时电路仿真结果

可见完成了调时和计时的功能.

5.3.2 动态显示模块仿真结果

从仿真结果6可以看出，位码随时间进行着模四的循环，与预期顺序相符，段码也进行着模四的循环，与各个通道对应的 bcd 码对应的 7 段数码管的显示信号相互对应，符合设计要求

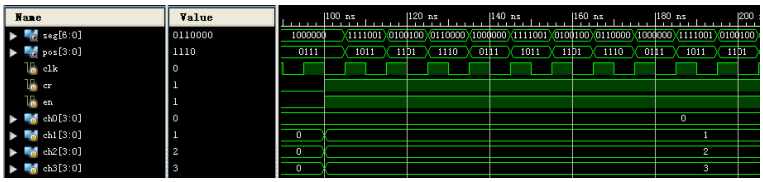


Figure 6: 动态显示模块仿真结果

5.3.3 24/12 进制切换模块仿真结果

从仿真结果7可以看出, 当选择为 24 进制时, is_am 输出始终为 0, 无论是否大于正午 12 点; 而当选择为 12 进制时, 当时间小于正午 12 点时, is_am 输出为 1, 而当时间大于正午 12 点时, is_am 输出为 0

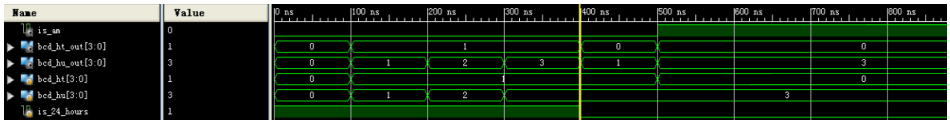


Figure 7: 24/12 时制切换结果

可见完成了 24/12 进制切换的功能.

5.3.4 top 顶层模块仿真结果

如图8所示. 从仿真结果可以看出, 当时制转换开关变化时, 对应的段码和 is_am 标志位也发生相应的变化. hour_adjust 和 min_adjust 发生相应的变化时, 相应的输出也发生相应的变化. (因为分频较多, 如果完全仿真需要持续较长时间, 具体结果可以参见附件 1 的实际烧写验证视频)

5.4 资源使用情况与系统模块架构图

5.4.1 资源使用情况

Technology Schematic 如图9所示, RTL 图如图10所示

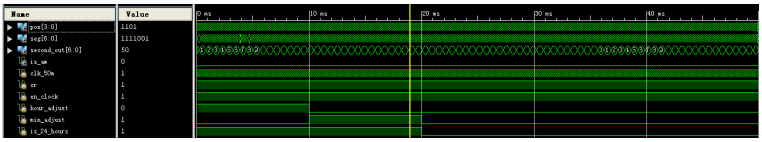


Figure 8: 顶层 top 模块仿真

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	53	1,920	2%
Number of 4 input LUTs	114	1,920	5%
Number of occupied Slices	75	960	7%
Number of Slices containing only related logic	75	75	100%
Number of Slices containing unrelated logic	0	75	0%
Total Number of 4 input LUTs	137	1,920	7%
Number used as logic	114		
Number used as a route-thru	23		
Number of bonded IOBs	25	83	30%
Number of BUFGMUXs	2	24	8%
Average Fanout of Non-Clock Nets	3.25		

Figure 9: Technology Schematic

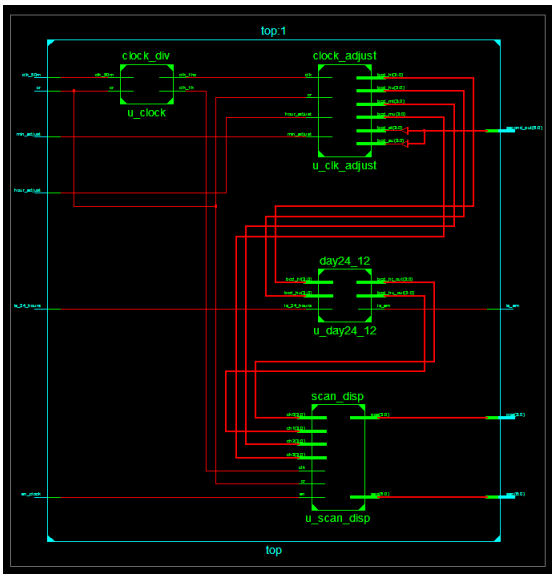


Figure 10: RTL 图

5.4.2 系统模块架构图

如图11所示

5.5 代码烧写验证，观察最终实验结果

将编写的代码烧写到 FPGA 中，进行验证，实验结果视频见附件 1.

可以看出，LED 处显示为 8 位 bcd 码，其显示的数字随秒的流逝而递增，当计数到 59 时归零，并是分加一. 当拨动时调整开关或者分调整开关时，相应的时或分就会随着秒的增加而递增，实现调时功能. 而拨动 24/12 时制调整开关后，8 位 LED 的第一个 led 便会作为是否为早上的指示灯，且对应的小时的显示也会发生相应的调整，当大于正午十二点时，早上指示灯熄灭，小时显示在相应的 24 进制的数字基础上减去 12；当小于正午十二点时，早上指示灯亮，小时显示与 24 进制的数字相同.

Figure 11: 系统模块架构图

6 小结

通过此次实验, 我加深了对 Verilog 语言的理解, 第一次亲手实践了使用 FPGA 构成中规模数字电路, 自己的仿真技能也大幅提升, 对于 n 进制计数器, 分频器, 7 段数码管电路的动态显示, Verilog 语言的各种特性都有了更深一层的理解.

7 调试步骤, 实验中出现的问题, 分析及解决方案

7.1 调试步骤

在对设计的 FPGA 电路进行调试时, 我进行了下列步骤

1. 每完成一个模块后, 便编写相应的测试代码对其进行仿真, 检验结果是

否正确.

2. 当每个模块的测试仿真结果正确后, 再编写更高层模块的代码并进行相应的仿真.

3. 当所有模块的仿真完成后, 将代码烧写到开发板上进行验证, 得到结果.

如: 我编写了几个模块的仿真文件, bcd_counter_sim, c24_sim, clk_adjust_sim, clk_div_sim, day_24_12_sim, scan_sim, second_sim, top_sim, 对各个模块进行了测试, 在测试完成后, 再将代码烧写到 FPGA 上, 可以发现, 在仿真正确后, 烧写后的验证结果也是正确的

7.2 实验中出现的问题，分析及解决方案

在本次实验过程中，我收集了一些常见的问题，小结如下：

- 当文件树中出现问号文件时, 需要重新将其从文件夹中添加到工程
- 对于 Verilog 语言, reg 类型与 wire 类型有很大区别, wire 类型可以使用 assign 进行绑定, 而 reg 类型可以在 always 语句中进行异步赋值
- Verilog 语言与一般的程序语言不同, 大部分均为异步执行, 不能在 if 语句中进行 assign 或者调用模块的操作
- 当在编写 Verilog 代码时, 应先进行仿真测试, 再将其烧写到开发板中进行实际验证, 而不要一开始就直接将所有代码全部烧入开发板中, 不然即便出现错误, 也很难排查, 而且应当分模块的进行仿真测试, 先测试底层模块, 最后再测试上层模块
- 编写较大规模代码时, 设计时应当遵循自顶向下的思想, 先设计处顶层模块, 再思考底层模块.