**华中科技大学**

**《电子线路设计、测试与实验》实验报告**

|  |  |
| --- | --- |
| 实验名称： | 多功能数字钟 |
| 院（系）： | 电子信息与通信工程学院 |
| 专业班级： | 提高1701 |
| 姓名： | 周瑞松 |
| 学号： | U201713303 |
| 时间： | 2019.4.3 |
| 地点： | 南一楼东303 |
| 实验成绩： |  |
| 指导教师： | 汪小燕教授 |

2019 年 4 月 3 日

**一、实验目的**

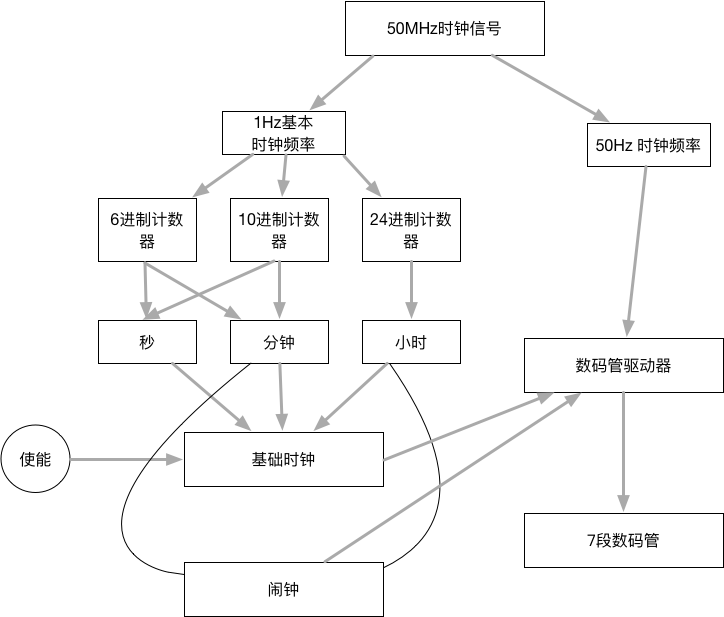
1. 熟悉verilog分层设计；
2. 熟悉功能模块的设计与重复调用；
3. 掌握数字电子钟的实现原理；
4. 实现闹钟，整点报时和时制切换。

**二、实验条件**

|  |  |
| --- | --- |
| 内容 | 型号 |
| 开发环境 | ISE 14.7 |
| 开发语言 | Verilog HDL |
| 开发板 | NEXYS4 ARTIX-7 |

**三、实验原理**

1）顶层模块



2）七段译码器

七段译码器用于将时间信号显示在数码管上。当输入D3D2D1D0接四位二进制数码十，输出低电平有效，用以驱动共阳极显示器。

3）动态扫描

动态数码扫描显示方式是利用了人眼的视觉暂留效应，把八个数码管按一定顺序（从左至右或从右至左）进行点亮，当频率足够大时，我们看到的将是全部同时显示（点亮），与传统方式得到的视觉效果完全一样。事实上，因为数码管点亮不是瞬间就可以的，它也需要一定的时间，该时间与数码管的选择有关系。为了折中这一对矛盾，实验中一般可将计数频率选择在100Hz左右可以满足上述两个要求。

4）分频器

该模块的任务是对100MHz的时钟信号进行分频，产生1Hz的秒脉冲信号，其占空比为50%，作为数字钟的计时基准。

设计一个模数为25×106的二进制递增计数器，其计数范围为0～24999999，每当计数器记到最大值时，输出信号翻转一次，即可产生1Hz的秒脉冲，其占空比为50%。

5）计数器设计

计数器分为三种计数器：模24计数器，模10计数器，模6计数器。

1. 模24计数器将八位寄存器分为两个四位二进制寄存器，分别存储时的个位和十位。个位范围从0000到1001，1001的下一位清零，同时十位进1，当十位为2且个位为3时，下一步个位十位同时清零。
2. 模10计数器使用一个四位二进制寄存器，用于存储分和秒的个位。计数范围从0000到1001，1001的下一位清零，同时控制模6计数器使能端使其进1。
3. 模6计数器使用一个四位二进制寄存器，用于存储分和秒的十位。计数范围从0000到0101，0101的下一位清零，同时控制模24计数器进1。

**四、实验内容**

1）拟定数字钟电路的组成框图，采用分层次、分模块的方法设计电路。

2）使用Verilog HDL设计各单元电路并进行仿真分析。

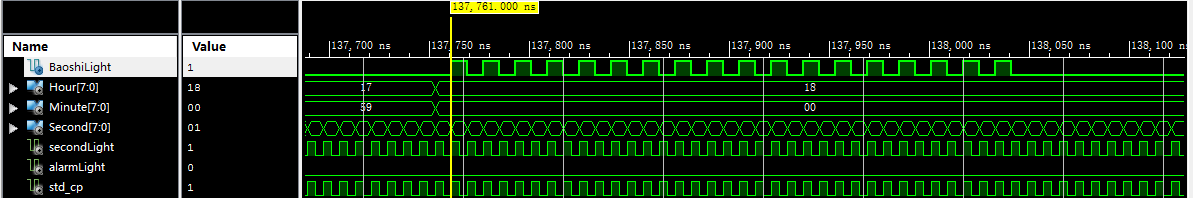
3）用FPGA开发板实现数字钟，并实际测试数字钟系统的逻辑功能。

1. 根据实验要求，实现闹钟，整点报时和时制切换三个拓展功能。

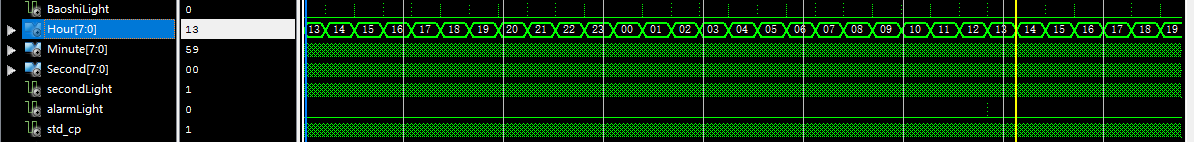
**五、仿真结果**

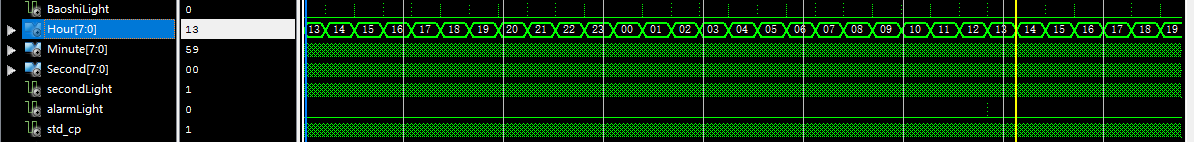
## 整点报时

当前到达整点18:00，报时灯波形如下，闪烁18次

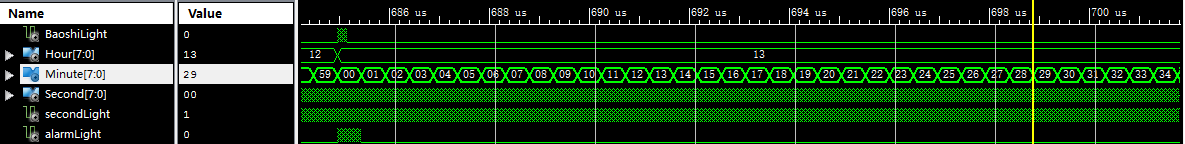


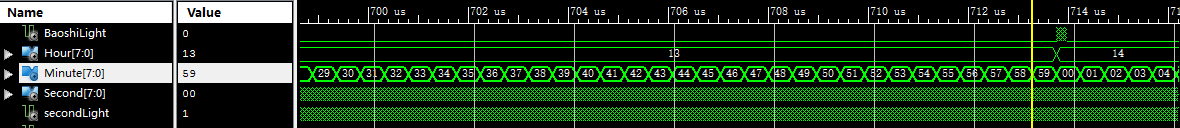
## 24小时波形图



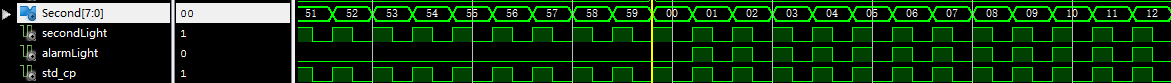


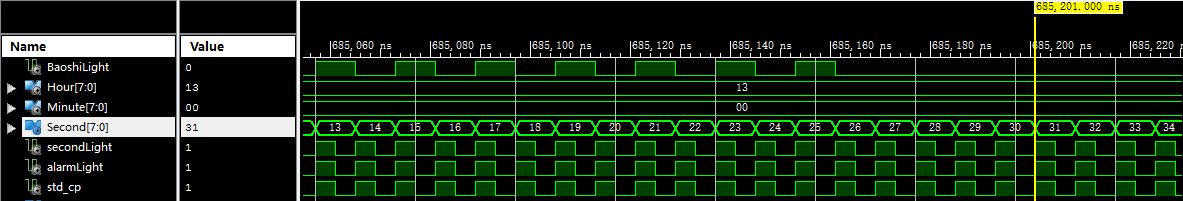
## 1分钟波形图

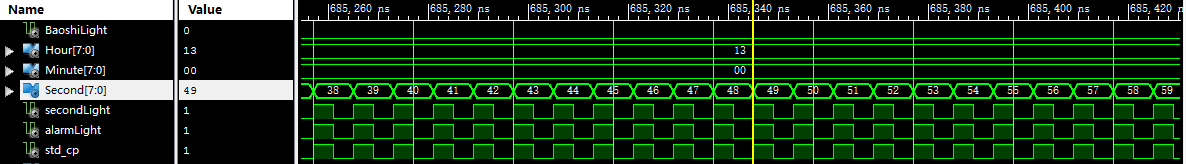




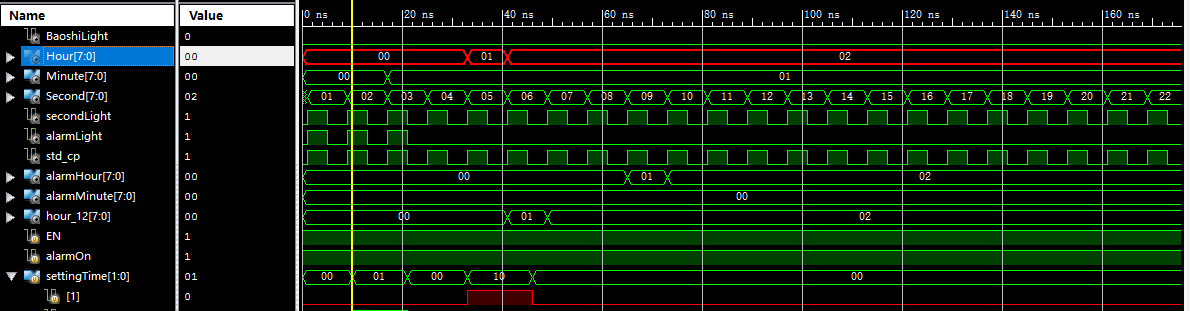
## 一秒波形图



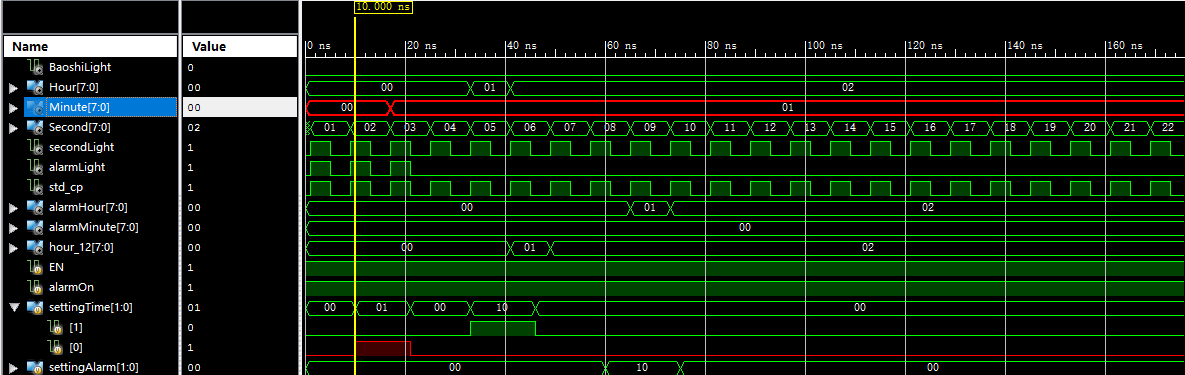




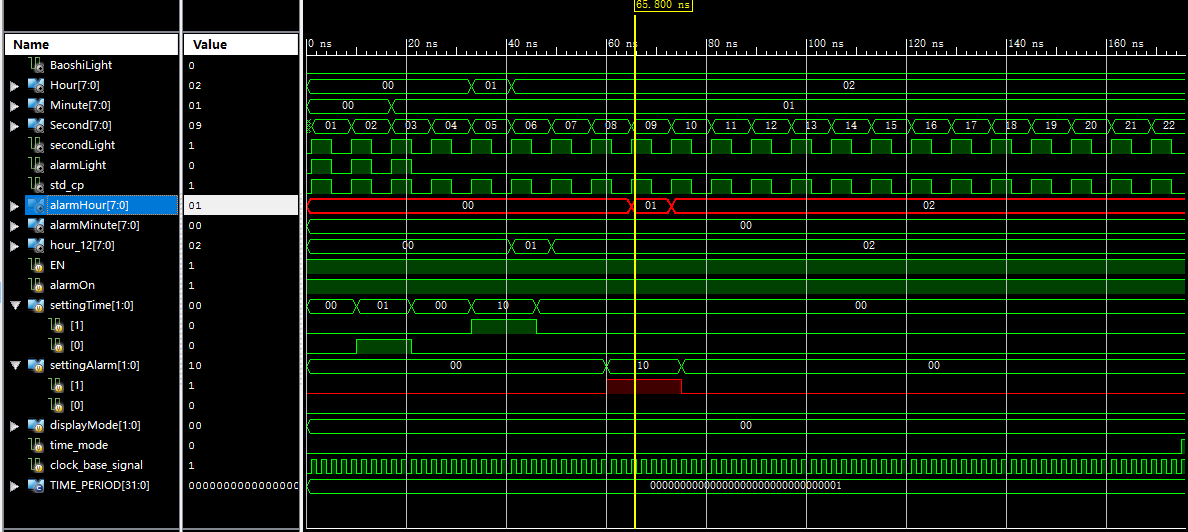
## 时间（小时）设定仿真

****

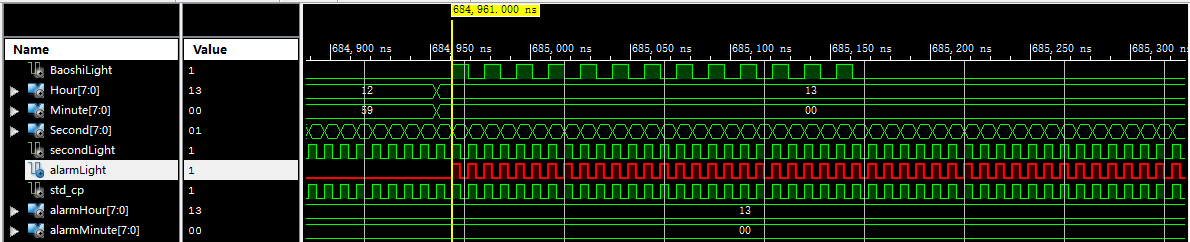
## 时间（分钟）设定仿真

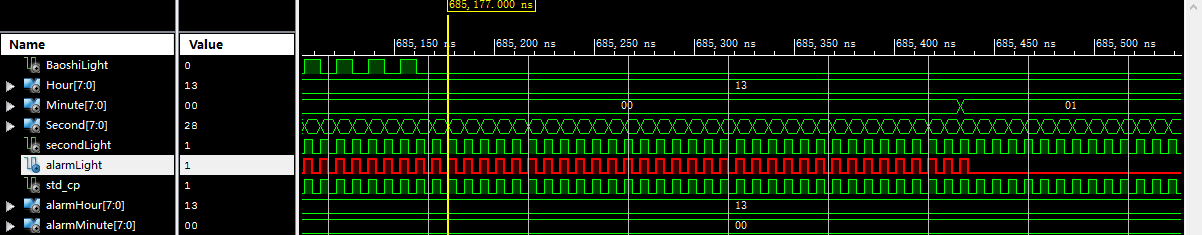
****

## 闹钟设定仿真

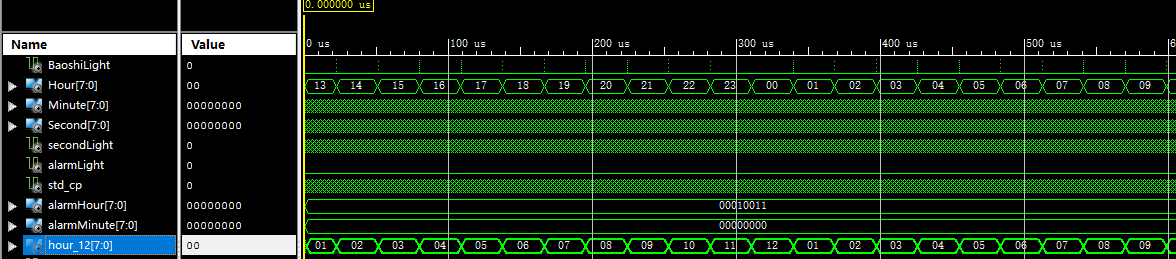
****

## 闹钟启动仿真

****

****

## 12进制／24进制仿真



**六、模块代码**

/\*

\* 顶层模块，用于将开发板上io与模块绑定及连接各底层模块

\* @input params:

\* clk: 时钟信号

\* reset: 重置

\* EN: 使能信号

\* settingButtion: 设置信号（按钮）

\* chooseButtion： 选择设置（闹钟/时钟信号） 开关

\* alarmOn: 开关闹钟

\* chooseTimeMode: 选择时间模式

\*

\* @output params:

\* //time\_mode:时间模式 12/24

\* alarm\_light: 闹钟指示灯

\* time\_light: AM / PM 指示灯

\* BaoshiLight： 报整点时数

\* seg\_out: 开发板上的数码管

\* an: 开发板上的数码管片选信号

\*/

module top\_module(

input clk,

input reset,

input EN\_siginal,

input [1:0] settingButtion,

input chooseButtion,

input alarmOn,

input chooseTimeMode,

//output time\_mode,

output alarm\_light,

output time\_light,

output BaoshiLight,

output [7:0] seg\_out,

output [7:0] an

);

wire clk\_500ms;

wire clk\_driver;

wire clk\_1s;

wire [7:0] clockHour;

wire [7:0] clockMin;

wire [7:0] clockSec;

wire [7:0] alarmHour;

wire [7:0] alarmMin;

reg [1:0] clockSetting;

reg [1:0] alarmSetting;

wire [2:0]pos;

wire [3:0]num;

//时钟分频

clk\_divider clk\_1(.clk(clk), .delay(100\_000\_000), .clk\_divd(clk\_1s));

clk\_divider clk\_2(.clk(clk), .delay(100\_000), .clk\_divd(clk\_driver));

clk\_divider clk\_3(.clk(clk), .delay(50\_000\_000), .clk\_divd(clk\_500ms));

//时钟信号

clock clock\_1(.clk(clk\_1s),

.reset(reset),

.en(EN\_siginal),

.settingButton(clockSetting),

.curHour(clockHour),

.curMin(clockMin),

.curSec(clockSec) );

//选择信号

always@(posedge clk\_driver) begin

if(~chooseButtion) begin

clockSetting = settingButtion;

alarmSetting = 0;

end

else begin

clockSetting = 0;

alarmSetting = settingButtion;

end

end

alarm alarm\_1(.clk(clk\_1s),

.reset(reset),

.alarmOn(alarmOn),

.settingButton(alarmSetting[1:0]),

.curHour(clockHour),

.curMin(clockMin),

.curSec(clockSec),

.alarmLight(alarm\_light),

.outHour(alarmHour),

.outMin(alarmMin) );

/\*call time when time is at o'clock\*/

CallTime calltime(.clk(clk\_1s), .Hour(clockHour), .Minute(clockMin), .Light(BaoshiLight));

/\* display driver, output 7-segment led \*/

display\_driver ddriver( .clk(clk\_driver),

.chooseButtion(chooseButtion),

.clockHour(clockHour),

.clockMin(clockMin),

.clockSec(clockSec),

.alarmHour(alarmHour),

.alarmMin(alarmMin),

.chooseMode(chooseTimeMode),

//.time\_mode(time\_mode),

.pos(pos),

.num(num) );

/\* 3-8 decoder module, control an \*/

decoder\_38 decd38(.pos(pos), .sel(an));

/\* seven segment driver module \*/

seven\_seg\_driver ssd(.num(num), .point(0), .seg(seg\_out));

time\_set setting(.clk(clk\_divider), .curHour(clockHour), .time\_light(time\_light));

endmodule

/\*

\* 时钟模块

\* input params:

\* clk: 时钟信号输入

\* reset: 重置信号

\* EN： 使能端

\* settingButtion: 设置时钟

\*

\*

\* output params:

\* curHour: 目前的小时

\* curMin : 目前的分钟

\* curSec : 目前的秒

\*/

module clock( input clk,

input clk\_500ms,

input reset,

input en,

input [1:0] settingButton,

output reg [7:0] curHour,

output reg [7:0] curMin,

output reg [7:0] curSec

);

wire [7:0] clockHour;

wire [7:0] clockMinute;

wire [7:0] clockSec;

reg min\_clk;

reg hour\_clk;

initial begin

min\_clk = 0;

hour\_clk = 0;

end

//分钟和小时的分频

always @(posedge clk\_500ms) begin

if(settingButton[0]) begin

min\_clk <= 1;

min\_clk <= 0;

end

if(settingButton[1]) begin

hour\_clk <= 1;

hour\_clk <= 0;

end

if(curSec[3:0] == 9 && curSec[7:4] == 5) begin

min\_clk <= 0;

min\_clk <= 1;

min\_clk <= 0;

end

if(curMin[3:0] == 9 && curMin[7:4] == 5 && curSec[3:0] == 9 && curSec[7:4] == 5) begin

min\_clk <= 0;

hour\_clk <= 1;

min\_clk <= 0;

end

end

//重置小时,分钟,秒

counter\_60 alarm\_second(.CP(clk), .reset(reset), .EN(en),.Cnt(clockSec));

counter\_60 alarm\_minute(.CP(min\_clk), .reset(reset), .EN(en),.Cnt(clockMinute));

counter\_24 alarm\_hour(.CntH(clockHour[7:4]),.CntL(clockHour[3:0]), .reset(reset),.EN(en), .CP(hour\_clk));

always @(\*) begin

curHour = clockHour;

curMin = clockMinute;

curSec = clockSec;

end

endmodule

/\*\*

content: Alarm

\*/

/\*

\* 闹钟模块

\* input params:

\* clk: 时钟输入

\* reset: 重置信号

\* alarmOn： 闹钟开关

\* settingButtion: 设置时钟

\* curHour: 目前的小时

\* curMin : 目前的分钟

\* output params:

\* alarmLight: 闹钟状态LED

\* outHour ： 闹钟设置的时间

\* outMin ： 闹钟设置的时间

\*/

module alarm( input clk,

input reset,

input alarmOn,

input [1:0] settingButton,

input [7:0] curHour,

input [7:0] curMin,

input [7:0] curSec,

output reg [7:0] outHour,

output reg [7:0] outMin,

output reg alarmLight);

wire [7:0] alarmClockHour;

wire [7:0] alarmClockMinute;

reg light; //控制闹钟闪烁

initial begin

alarmLight = 0;

light = 0;

end

//按钮作为设置闹钟的使能端

wire minL\_en, minH\_en, hour\_en;

assign minL\_en = settingButton[0];

assign minH\_en = (settingButton[0] && alarmClockMinute[3:0]==4'h9);

assign hour\_en = settingButton[1];

//设置小时和分钟

counter\_10 alarm\_minute\_0(.Cnt(alarmClockMinute[3:0]),.reset(reset),.EN(minL\_en),.CP(clk));

counter\_6 alarm\_minute\_1(.Cnt(alarmClockMinute[7:4]),.reset(reset),.EN(minH\_en),.CP(clk));

counter\_24 alarm\_hour(.CntH(alarmClockHour[7:4]),.CntL(alarmClockHour[3:0]), .reset(reset),.EN(hour\_en), .CP(clk));

always @(posedge clk)

begin

if (alarmOn && (curMin == alarmClockMinute) && (curHour == alarmClockHour)) begin

// 此时闹钟状态为1

alarmLight = 1;

end

if (curMin != alarmClockMinute || ~alarmOn) begin

// 闹钟响了一分钟，此时闹钟状态为0

alarmLight = 0;

end

end

always @(\*) begin

outHour = alarmClockHour;

outMin = alarmClockMinute;

end

endmodule // Alarm

/\*

\* 时钟分频器模块， 分频比：1/delay

\* input params:

\* clk: 时钟输入

\* delay: 分频输入

\* output params:

\* clk\_divd: 分频后的时钟信号

\*/

module clk\_divider(

input clk,

input [31:0] delay,

output reg clk\_divd

);

reg [31:0]count;

//reg [31:0]count2;

//parameter delay = 100\_000;

initial begin

count = 0;

clk\_divd = 0;

end

always@ (posedge clk) begin

if (count >= delay / 2) begin

count = 0;

clk\_divd = ~clk\_divd;

end

count = count + 1;

end

endmodule

/\*

\* 数码管显示驱动模块， 用于将数值转化为数码管输出

\* input params:

\* clk: 时钟信号

\* clockHour, clockMin, clockSec : 时钟描述

\* alarmHour, alarmMin : 闹钟描述

\* chooseButtion : 时钟闹钟显示

\* chooseMode: 选择时间形式12/24

\* output params:

//time\_mode: 12/24小时

\* seg: 数码管显示信号

\* seg\_pos: 数码管片选信号

\*/

module display\_driver(

input clk,

input chooseButtion,

input [7:0] clockHour,

input [7:0] clockMin,

input [7:0] clockSec,

input [7:0] alarmHour,

input [7:0] alarmMin,

input chooseMode,

//output reg time\_mode,

output reg [2:0]pos,

output reg [3:0]num

);

reg [3:0] d\_num1;

reg [3:0] d\_num2;

reg [3:0] d\_num3;

reg [3:0] d\_num4;

reg [3:0] d\_num5;

reg [3:0] d\_num6;

reg [5:0] hours;

always @(posedge clk) begin

if(~chooseButtion) begin

hours = clockHour[7:4]\*10 + clockHour[3:0];

if(chooseMode == 1 && hours > 12) begin

hours = hours-12;

d\_num1 = hours / 10;

d\_num2 = hours - 10\*d\_num1;

end

else begin

d\_num1 = clockHour[7:4];

d\_num2 = clockHour[3:0];

end

d\_num3 = clockMin[7:4];

d\_num4 = clockMin[3:0];

d\_num5 = clockSec[7:4];

d\_num6 = clockSec[3:0];

end

else begin

d\_num1 = alarmHour[7:4];

d\_num2 = alarmHour[3:0];

d\_num3 = alarmMin[7:4];

d\_num4 = alarmMin[3:0];

end

end

always @(posedge clk) begin

if (pos == 6) begin

pos <= 0;

end

case (pos)

3'd0: begin

num = d\_num1;

end

3'd1: begin

num = d\_num2;

end

3'd2: begin

num = d\_num3;

end

3'd3: begin

num = d\_num4;

end

3'd4: begin

num = chooseButtion ? 8'b11111111 : d\_num5;

end

3'd5: begin

num = chooseButtion ? 8'b11111111 : d\_num6;

end

3'd6: begin

num = 8'b11111111;

end

default: num = 8'b11111111;

endcase

pos <= pos+1;

end

endmodule

/\*整点报时模块\*/

/\*

\* @params:

\* clk: 时钟信号

\* Hour, Minute: 目前的时间

\* light：信号灯,奇数秒点亮LED，偶数秒熄灭

\*/

module CallTime(

input clk,

input [7:0] Hour,

input [7:0] Minute,

output reg Light

);

reg [7:0] counter;

reg [7:0] curHour;

initial begin

Light = 0;

end

always @(posedge clk) begin

if (Minute == 8'h00) begin

curHour = Hour[7:4] \* 10 + Hour[3:0];

if (counter < curHour + curHour) begin

Light = ~Light;

counter = counter + 1;

end

else begin

Light = 0;

counter = counter + 1;

end

end

else begin

counter = 0;

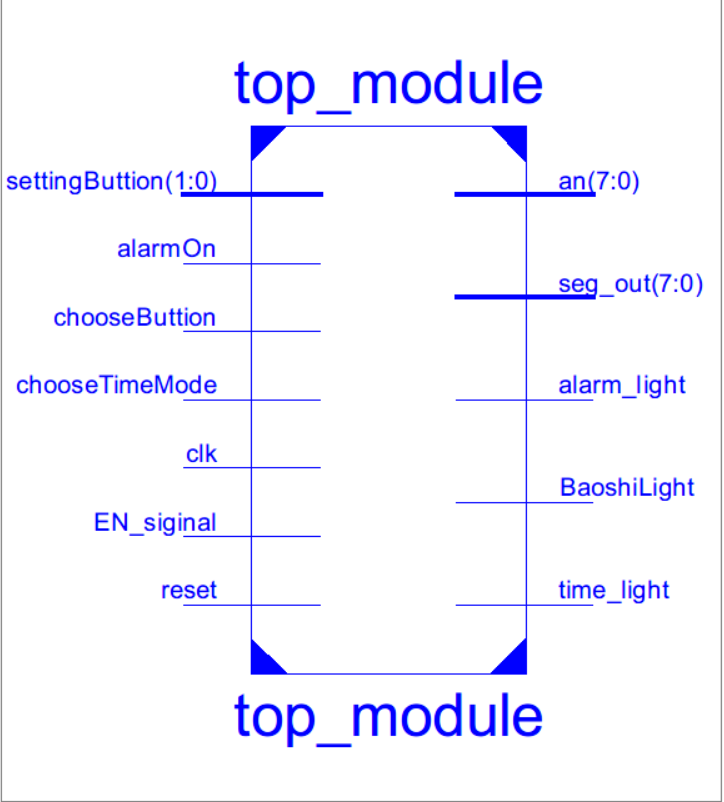
end

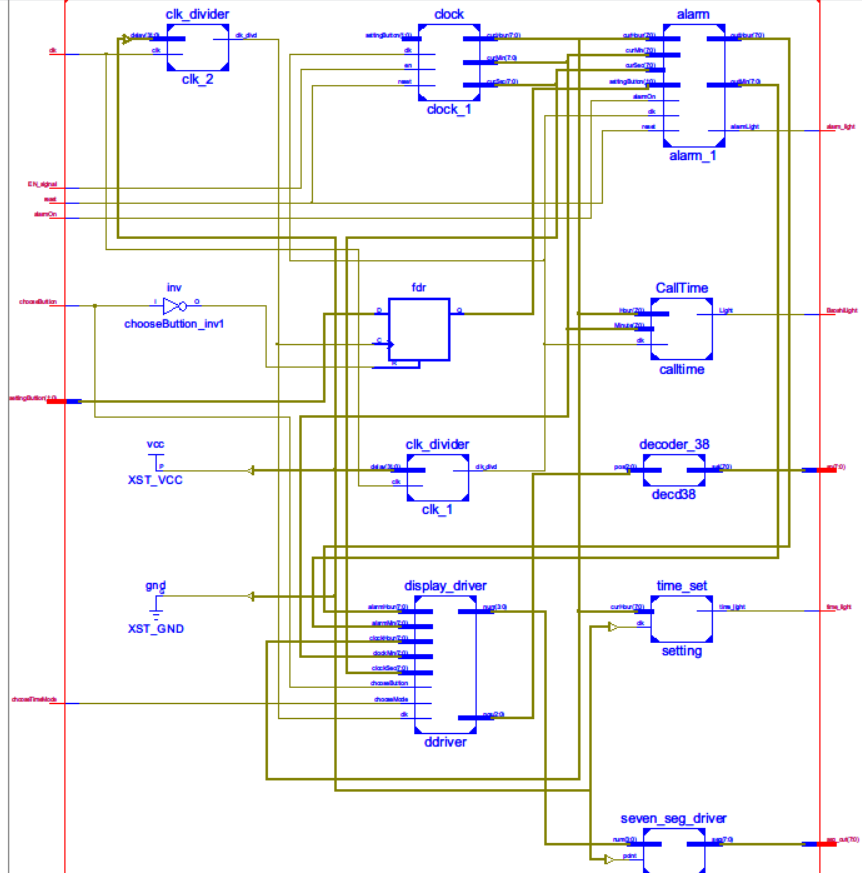
end

endmodule

**(其他代码略)**

# 七、电路图

****

****

**八、引脚分配**

|  |  |
| --- | --- |
| N17 | nCR |
| J15 | EN |
| M13 | 时制转换12/24 |
| L16 | 闹钟开关 |
| V10 | 显示闹钟 |
| P17 | 设置时间分钟 |
| M17 | 设置时间小时 |
| P17 | 设置闹钟分钟 |
| M17 | 设置时间小时 |
| H17 | 整点报时LED |
| V11 | 闹钟LED |

**九、实验结果及分析**

经过仿真检查，电路逻辑正确，因此编译生成bit文件写入开发板中，时钟功能运行正常。

性能分析:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Slice Logic Utilization** | **Used** | **Available** | **Utilization** | **Note(s)** |
| Number of Slice Registers | 182 | 126,800 | 1% |  |
| Number used as Flip Flops | 182 |  |  |  |
| Number used as Latches | 0 |  |  |  |
| Number used as Latch-thrus | 0 |  |  |  |
| Number used as AND/OR logics | 0 |  |  |  |
| Number of Slice LUTs | 223 | 63,400 | 1% |  |
| Number used as logic | 216 | 63,400 | 1% |  |
| Number using O6 output only | 70 |  |  |  |
| Number using O5 output only | 6 |  |  |  |
| Number using O5 and O6 | 140 |  |  |  |
| Number used as ROM | 0 |  |  |  |
| Number used as Memory | 0 | 19,000 | 0% |  |
| Number used exclusively as route-thrus | 7 |  |  |  |
| Number with same-slice register load | 4 |  |  |  |
| Number with same-slice carry load | 3 |  |  |  |
| Number with other load | 0 |  |  |  |
| Number of occupied Slices | 75 | 15,850 | 1% |  |
| Number of LUT Flip Flop pairs used | 229 |  |  |  |
| Number with an unused Flip Flop | 63 | 229 | 27% |  |
| Number with an unused LUT | 6 | 229 | 2% |  |
| Number of fully used LUT-FF pairs | 160 | 229 | 69% |  |
| Number of unique control sets | 19 |  |  |  |
| Number of slice register sites lost         to control set restrictions | 82 | 126,800 | 1% |  |
| Number of bonded [IOBs](F://FPGAcode/digital_clock/top_module_map.xrpt?&DataKey=IOBProperties) | 27 | 210 | 12% |  |
| Number of LOCed IOBs | 27 | 27 | 100% |  |
| Number of RAMB36E1/FIFO36E1s | 0 | 135 | 0% |  |
| Number of RAMB18E1/FIFO18E1s | 0 | 270 | 0% |  |
| Number of BUFG/BUFGCTRLs | 3 | 32 | 9% |  |
| Number used as BUFGs | 3 |  |  |  |
| Number used as BUFGCTRLs | 0 |  |  |  |
| Number of IDELAYE2/IDELAYE2\_FINEDELAYs | 0 | 300 | 0% |  |
| Number of ILOGICE2/ILOGICE3/ISERDESE2s | 0 | 300 | 0% |  |
| Number of ODELAYE2/ODELAYE2\_FINEDELAYs | 0 |  |  |  |
| Number of OLOGICE2/OLOGICE3/OSERDESE2s | 0 | 300 | 0% |  |
| Number of PHASER\_IN/PHASER\_IN\_PHYs | 0 | 24 | 0% |  |
| Number of PHASER\_OUT/PHASER\_OUT\_PHYs | 0 | 24 | 0% |  |
| Number of BSCANs | 0 | 4 | 0% |  |
| Number of BUFHCEs | 0 | 96 | 0% |  |
| Number of BUFRs | 0 | 24 | 0% |  |
| Number of CAPTUREs | 0 | 1 | 0% |  |
| Number of DNA\_PORTs | 0 | 1 | 0% |  |
| Number of DSP48E1s | 2 | 240 | 1% |  |
| Number of EFUSE\_USRs | 0 | 1 | 0% |  |
| Number of FRAME\_ECCs | 0 | 1 | 0% |  |
| Number of IBUFDS\_GTE2s | 0 | 4 | 0% |  |
| Number of ICAPs | 0 | 2 | 0% |  |
| Number of IDELAYCTRLs | 0 | 6 | 0% |  |
| Number of IN\_FIFOs | 0 | 24 | 0% |  |
| Number of MMCME2\_ADVs | 0 | 6 | 0% |  |
| Number of OUT\_FIFOs | 0 | 24 | 0% |  |
| Number of PCIE\_2\_1s | 0 | 1 | 0% |  |
| Number of PHASER\_REFs | 0 | 6 | 0% |  |
| Number of PHY\_CONTROLs | 0 | 6 | 0% |  |
| Number of PLLE2\_ADVs | 0 | 6 | 0% |  |
| Number of STARTUPs | 0 | 1 | 0% |  |
| Number of XADCs | 0 | 1 | 0% |  |
| Average Fanout of Non-Clock Nets | 2.68 |  |  |  |

**十、小结**

本次试验综合性较强，一开始由于没有仔细去设计构想，导致代码越写越乱，把门级电路描述和行为描述弄混了，导致程序很乱，除了很多错误。之后重新设计了一番框架模型，将每个模块要实现的功能与输入输出明细，然后重新开始构建每个模块，最后实现了所有功能。可见，在电路设计中，一定要有目的性，要有很明细的架构设计。

本次试验采用了门级描述和行为级描述的结合，在顶层模块使用门级描述，在分模块中使用更为方便的行为级描述。

最重要的是，一写报告的规范的到了训练，从实验介绍，到功能说明以及仿真报告都需要认真地完成，不得马虎。