远程FPGA虚拟实验平台用SystemVerilog HDL实现彩灯控制器

原创

韭#菜菜子 于 2021-04-17 16:44:38 发布 1078 收藏 9

分类专栏： CPU实验作业 文章标签： verilog 状态机 systemverilog

本文链接：https://blog.csdn.net/SDG_29/article/details/115732932

CPU实验作业 专栏收录该内容

11 篇文章 18 订阅

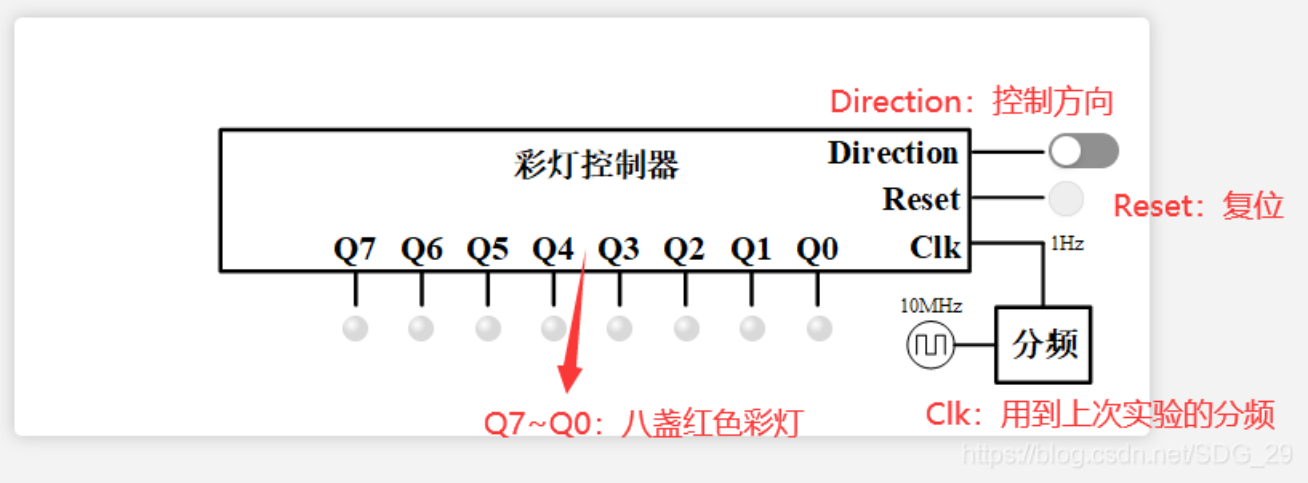
订阅专栏

远程FPGA虚拟实验平台用SystemVerilog HDL实现彩灯控制器

前言：本次实验要交一个jvp，一个rbf，一个sv，主要是靠自己画一个好看的jvp来展示实验内容，代码没什么大用处。代码我给了两套：五角星和时钟，五角星就是十个灯版本的简易彩灯，时钟是带了个进位的不标准彩灯，五角星的jvp在原理里给了，自提。

原理

实验材料



材料给的彩灯控制器是一个三段式的状态机，即：
状态转换，包括复位和其他特殊状态，对state进行控制；
状态计算，对next_state进行控制；
输出逻辑，根据state来输出。
实验要求我们写三段的。而实验材料的具体状态图还是看慕课吧。

```
wire reset =  PB[0];
wire clk;  // =  PB[1];
wire direction = S[8];
/************** The Logic of this experiment *************/
/* 对10MHz系统时钟进行分频，使用分频后的时钟作为移位寄存器的时钟。
   分频系数为10M，输出的clkout的频率为1Hz。 */
ClockDivider #(.RATIO(10000000)) divider_inst(.ClkIn(CLOCK), .Reset(reset), .ClkOut(clk));

// Finite State Machine
wire [7:0] pattern;
enum bit [3:0] {//枚举，state和next_state能有的四种状态
    STATE0   = 4'b0001,//用ont hot编码表示的状态，一串里就一个有效的一种编码，表示少还行，多了很长
    STATE1   = 4'b0010,
    STATE2   = 4'b0100,
    STATE3   = 4'b1000
} state, next_state;

always_ff @(posedge clk, posedge reset)//1.状态转换
begin
 if (reset)//复位，一个状态机得有个好的初始状态
  state <= STATE0;
 else
  state <= next_state;
end

always_comb//2.次态计算，这边用always_comb我个人觉得不太好，完全是仗着程序简单写的，最好还是指定变量
begin : set_next_state
    case (state)
    STATE0: begin
        if (direction==0)//确认是不是逆向的
            next_state = STATE1;
        else
            next_state = STATE3;
        end
    STATE1: begin
        if (direction==0)
            next_state = STATE2;
```

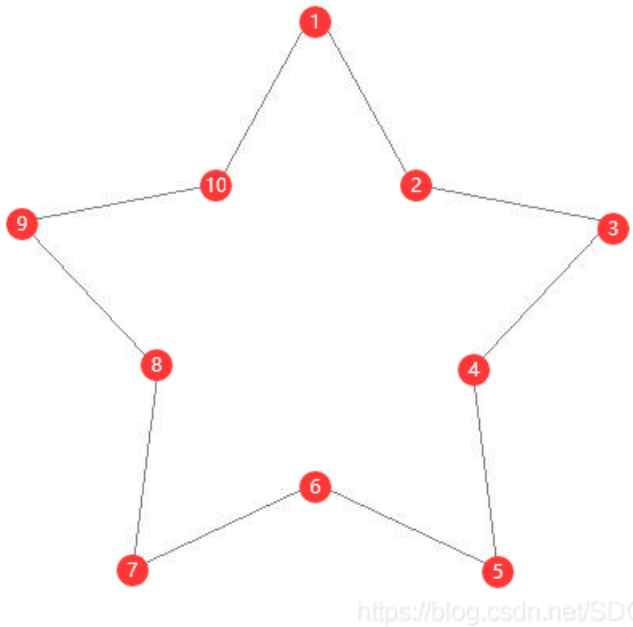
```
        else
            next_state = STATE0;
        end
    STATE2: begin
        if (direction==0)
            next_state = STATE3;
        else
            next_state = STATE1;
        end
    STATE3: begin
        if (direction==0)
            next_state = STATE0;
        else
            next_state = STATE2;
        end
    endcase
end : set_next_state

always_comb//3.输出逻辑
begin : set_outputs
    case (state)
    STATE0: begin
        pattern = 8'b1000_0001;
        end
    STATE1: begin
        pattern = 8'b0100_0010;
        end
    STATE2: begin
        pattern = 8'b0010_0100;
        end
    STATE3: begin
        pattern = 8'b0001_1000;
        end
    endcase
end : set_outputs

/****** Internal signal assignment to output port *******/
assign L[7:0] = pattern;//变量对应八盏灯
```

简易彩灯五角星

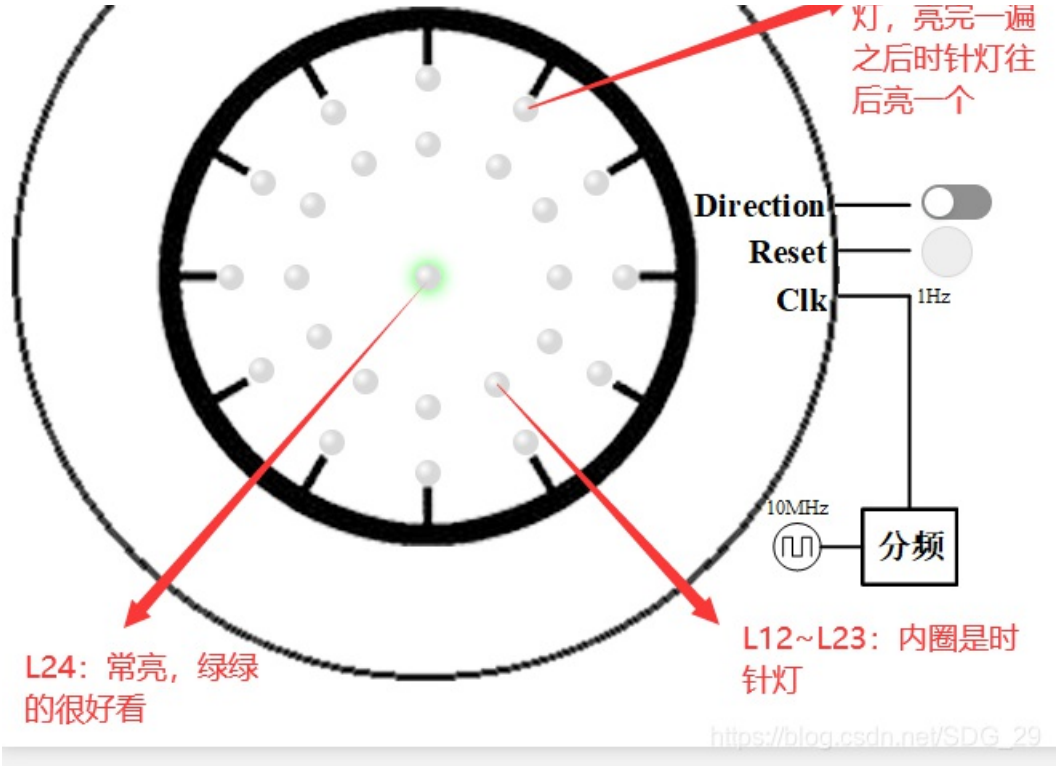
先来个实验面板，就是画图和ps结合一下画个图当背景，然后到虚拟实验平台用创新实验，载入背景，搞个jvp出来，保存一下。



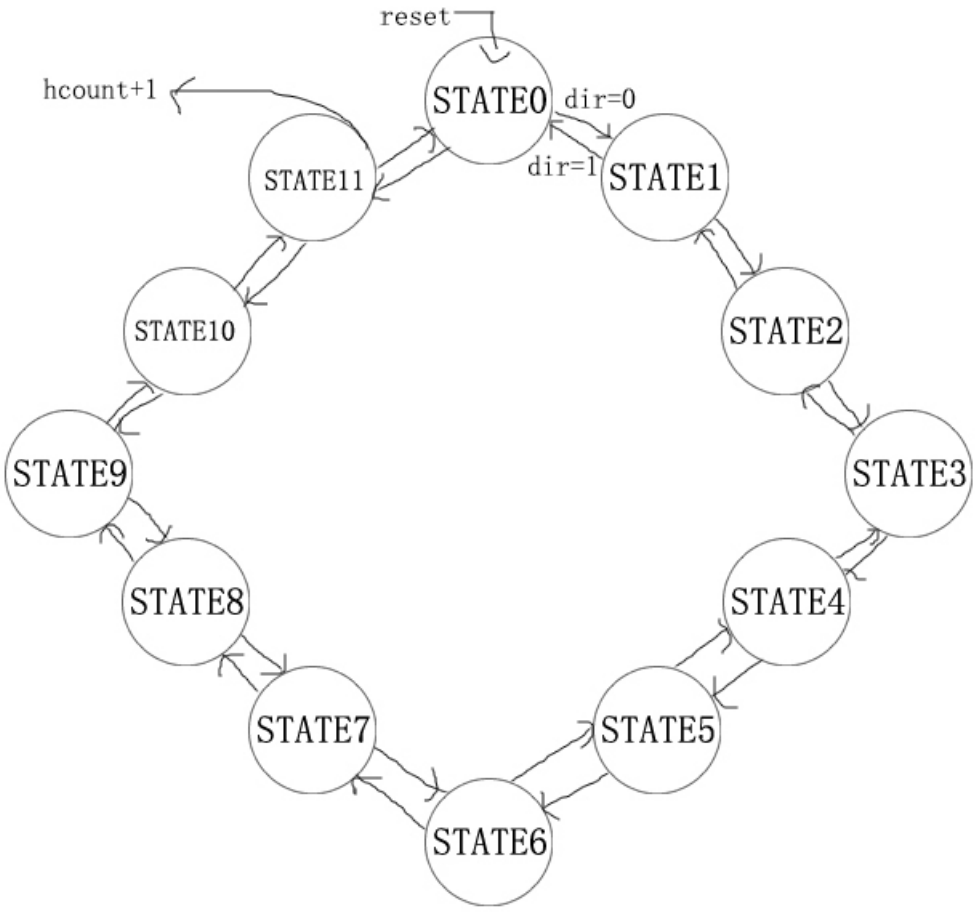
十个点一个简易五角星，不放个背景应该没人看得出来是五角星，所以建议插个背景图片。
做好了的jvp在这里 提取码：msnx

彩灯时钟

说是彩灯时钟，其实就是彩灯圆搞两遍，然后再进位一下。



画一下*不是很圆的圆形*状态图，*上帝给了我黑色的眼睛，我用它画黑暗画作*。分针和时针都用这个状态图，在状态转换时，如果检测到分针的状态走完一个轮回，就让时针也走一步。



源代码

简易彩灯五角星

VirtalBoard模块

没什么技术含量，就是老师那个灯改一改。

```
`default_nettype none
module VirtualBoard (
    input  logic  CLOCK,      // 10 MHz Input Clock
    input  logic [19:0] PB,   // 20 Push Buttons, logical 1 when pressed
    input  logic [35:0] S,    // 36 Switches
    output logic [35:0] L,    // 36 LEDs, drive logical 1 to light up
    output logic  [7:0] SD7,  // 8 common anode Seven-segment Display
    output logic  [7:0] SD6,
    output logic  [7:0] SD5,
    output logic  [7:0] SD4,
    output logic  [7:0] SD3,
    output logic  [7:0] SD2,
    output logic  [7:0] SD1,
    output logic  [7:0] SD0
);

/** The input port is replaced with an internal signal **/
wire reset =  PB[0];
wire clk;  // =  PB[1];
wire direction = S[8];//注意看啊注意看这里是s8，复制老师代码的时候我没注意这里是s8,debug头都秃了才发现，我的jvp里这里也设置的8嗷，注意一下

/************** The logic of this experiment *************/
/* 对10MHz系统时钟进行分频，使用分频后的时钟作为移位寄存器的时钟。
   分频系数为10M，输出的clkout的频率为1Hz。 */
ClockDivider #(.RATIO(10000000)) divider_inst(.ClkIn(CLOCK), .Reset(reset), .ClkOut(clk));

// Finite State Machine
wire [9:0] pattern;
enum bit [3:0] {
    STATE0   = 4'b0000,
    STATE1   = 4'b0001,
    STATE2   = 4'b0010,
    STATE3   = 4'b0011,
  STATE4   = 4'b0100,
    STATE5   = 4'b0101,
    STATE6   = 4'b0110,
    STATE7   = 4'b0111,
  STATE8   = 4'b1000,
    STATE9   = 4'b1001
} state, next_state;

always_ff @(posedge clk, posedge reset)
begin
 if (reset)
  state <= STATE0;
 else
  state <= next_state;
end

always_comb
begin : set_next_state
    case (state)
    STATE0: begin
        if (direction==0)
            next_state = STATE1;
        else
            next_state = STATE9;
```

```
            next_state = STATE9;
        end
    STATE1: begin
        if (direction==0)
            next_state = STATE2;
        else
            next_state = STATE0;
        end
    STATE2: begin
        if (direction==0)
            next_state = STATE3;
        else
            next_state = STATE1;
        end
    STATE3: begin
        if (direction==0)
            next_state = STATE4;
        else
            next_state = STATE2;
        end
    STATE4: begin
        if (direction==0)
            next_state = STATE5;
        else
            next_state = STATE3;
        end
    STATE5: begin
        if (direction==0)
            next_state = STATE6;
        else
            next_state = STATE4;
        end
    STATE6: begin
        if (direction==0)
            next_state = STATE7;
        else
            next_state = STATE5;
        end
    STATE7: begin
        if (direction==0)
            next_state = STATE8;
        else
            next_state = STATE6;
        end
    STATE8: begin
        if (direction==0)
            next_state = STATE9;
        else
            next_state = STATE7;
        end
    STATE9: begin
        if (direction==0)
            next_state = STATE0;
        else
            next_state = STATE8;
        end
    endcase
end : set_next_state

always_comb
```

```
begin : set_outputs
    case (state)
    STATE0: begin
        pattern = 10'b0000000001;
        end
    STATE1: begin
        pattern = 10'b0000000010;
        end
    STATE2: begin
        pattern = 10'b0000000100;
        end
    STATE3: begin
        pattern = 10'b0000001000;
        end
    STATE4: begin
        pattern = 10'b0000010000;
        end
    STATE5: begin
        pattern = 10'b0000100000;
        end
    STATE6: begin
        pattern = 10'b0001000000;
        end
    STATE7: begin
        pattern = 10'b0010000000;
        end
    STATE8: begin
        pattern = 10'b0100000000;
        end
    STATE9: begin
        pattern = 10'b1000000000;
        end
    endcase
end : set_outputs

/****** Internal signal assignment to output port *******/
assign L[9:0] = pattern;

endmodule
```

ClockDivider模块

工程文件里别删clockdivider，或者复制粘贴计数器和分频器实验里的就好。

彩灯时钟

嗯？美少女的创意是你能白看的吗？哎，有代码，我不给你看，我就馋你一下，就是玩儿~

测试/保存/提交

总结一下这次敲代码遇到的错误和解决方法。

①always_comb construct does not infer purely combinational logic…：像always_comb其实是不好乱用的，我在写时钟的时候有两个状态机，如果用always_comb就会有死循环，最好写一个always语句块的时候，标清楚变量不要偷懒，比如

②can't resolve multiple constant drivers…：一个变量只能在一个always里被赋值，所以代码里有两个always的时候，搞了个state和next_state，分别在俩always里赋值。
这次注意要完成两个部分，一是实验平台做个jvp，二是敲代码，嗯！