西 安 电 子 科 技 大 学

数字电路与逻辑设计实验 课程实验报告

实验名称 交通灯控制器

人工智能 学院 1920012 班

姓名 杨文韬 学号 18020100245

同作者 丁辉

实验日期 2021 年 10 月 31 日

成 绩

指导教师评语：

指导教师：

_____ 年 ____ 月 ____ 日

实验报告内容基本要求及参考格式

一、实验目的：预期达到的功能

二、实验环境

三、方案设计及理论计算：原理框图、关键部分的设计和计算

四、实验数据和仿真：仿真波形和结果

五、实验结果分析

六、实验代码

目录

交通灯控制器

1 实验目的

掌握基本的时序逻辑设计方法，学习利用计数器和状态机设计交通灯控制器。分两个方向 (1,2)，每个方向各有红 (R)、绿 (G) 两个交通灯。按下按钮 $K0 \sim K3$ 时直接进入对应序号的状态，随后即转入自动方式。在自动方式下，控制器的状态转移表如表 1 所示。

表 1: 状态转移表

状态	亮灯	停留时间
S0	R1,G2	2s
S1	R1	1s
S2	G1,R2	2s
S3	R2	1s

2 实验环境

- Quartus II 13.0: 创建项目，编写 Verilog HDL 代码，并进行编译之后生成 Test Bench 模板，输入信号进行仿真
- ModelSim ALTERA 10.4d: 进行仿真获得仿真波形
- LaTeX: 制作封面并进行文档编写排版
- Visio 2016: 用于原理框图的绘制

3 方案设计及理论计算

方案设计分为三个模块，分别为分频器模块 (clk_div)、按键防抖模块 (key_scan)、红绿灯状态转换模块 (led_trans)，它们之间的关系可用如图 1 所示的原理框图表示

3.1 分频器模块

在交通灯控制系统中，通过自动控制方式指挥交通。因此需要给电路一个稳定的时钟，板件接的时钟是 50mhz 频率的，对于以秒级的交通灯和 10ms-30ms 的按键防抖来

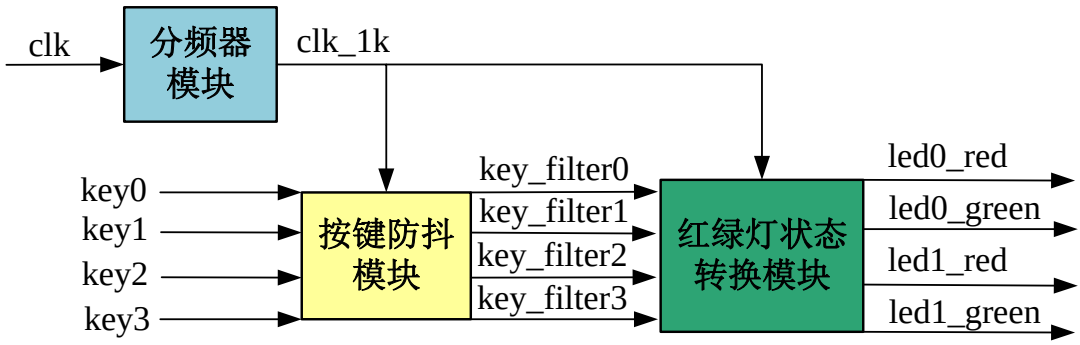

图 1: 原理框图

说，这样的频率太快了，因此我们需要设计分频器模块，将 50mhz 分频成 1khz。假设系统时钟频率为 clk_sys，期望输出时钟频率为 clk_out，则分频器的模值 M 计算公式为

$$M = \frac{clk\_sys/clk\_out}{2} - 1 = \frac{50 \times 10^3}{2} - 1 = 24999$$

信号的命名如下

- clk: 系统输入信号，由外部 50m 晶振产生
- rst: 系统复位信号，低电平有效
- clk_1k: 系统输出信号，为 1khz 频率的输出

在具体代码实现中，需要用到两个 always 模块，第一个 always 用于设计模 25000 计数器，第二个 always 用于记录 1khz 输出时钟信号当前是高电平还是低电平。

3.2 按键防抖模块

根据题意要求，需要设置 4 个按键，按下后分别置电路于 4 种状态，通过上网查找相关资料后发现，在按键过程中，由于硬件特性，会产生毛刺现象，一般我们需要选用 10ms 以上的防抖来规避，在我们的设计中，使用的是 16ms 的防抖。信号的命名如下

- clk_1k: 系统输入信号，为 1khz 的输入时钟信号
- rst: 系统复位信号，低电平有效
- key_in: 系统输入信号，为外部按键输入
- key_delt: 信号经过防抖处理后的脉冲信号，用于状态机的转移。

在具体代码实现中，用 16 位的 `key_in_shift` 记录连续 16ms 电平值，`key_in_shift` 全为 1 时，`key_filter` 为 1，`key_in_shift` 全为 0 时，`key_filter` 为 0，其他情况下 `key_filter`

保持上一次状态，最后用按键后松开的上升沿进行边沿检测，`key_delt` 为 1 时表示检测到边沿。

3.3 红绿灯状态转换模块

这个模块主要负责红绿灯的转换，首先调用前两个模块的信号，根据题意，从 S0 到 S1 状态和从 S2 到 S3 状态，需要从 0 计数到 1999，共 $2000 \times 1\text{ms} = 2\text{s}$；从 S1 到 S2 状态和从 S3 到 S0 状态，需要从 0 计数到 999，共 $1000 \times 1\text{ms} = 1\text{s}$

4 实验数据和仿真

用 Quartus II 生成 TestBench 后，在工程目录的 simulation/modelsim 文件下找到 vt 文件进行编辑，仿真输入的含义为假设一次按键时间为 20ms，11s 时按键 0 到 S0 状态，22s 时按键 1 到 S1 状态，33s 时按键 2 到 S2 状态，44s 时按键 3 到 S3 状态。得到如图 2 所示的仿真波形图

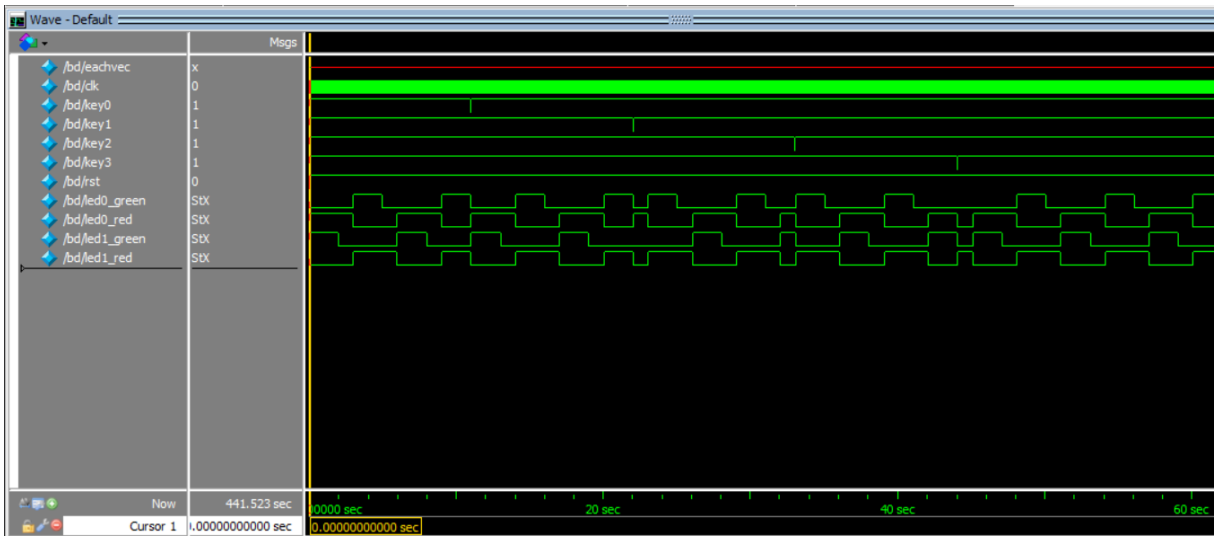

图 2: 仿真波形图

5 实验结果和分析

通过编写代码和仿真测试后，分配好管脚，管脚分配图见附录图 4 所示，我们将我们的程序烧录进开发板，S0 工作状态如图 3 所示，实验结果完全符合我们的预期，通过按按键进入对应的状态，实验不足之处在于没有添加数码管数字显示倒计时。

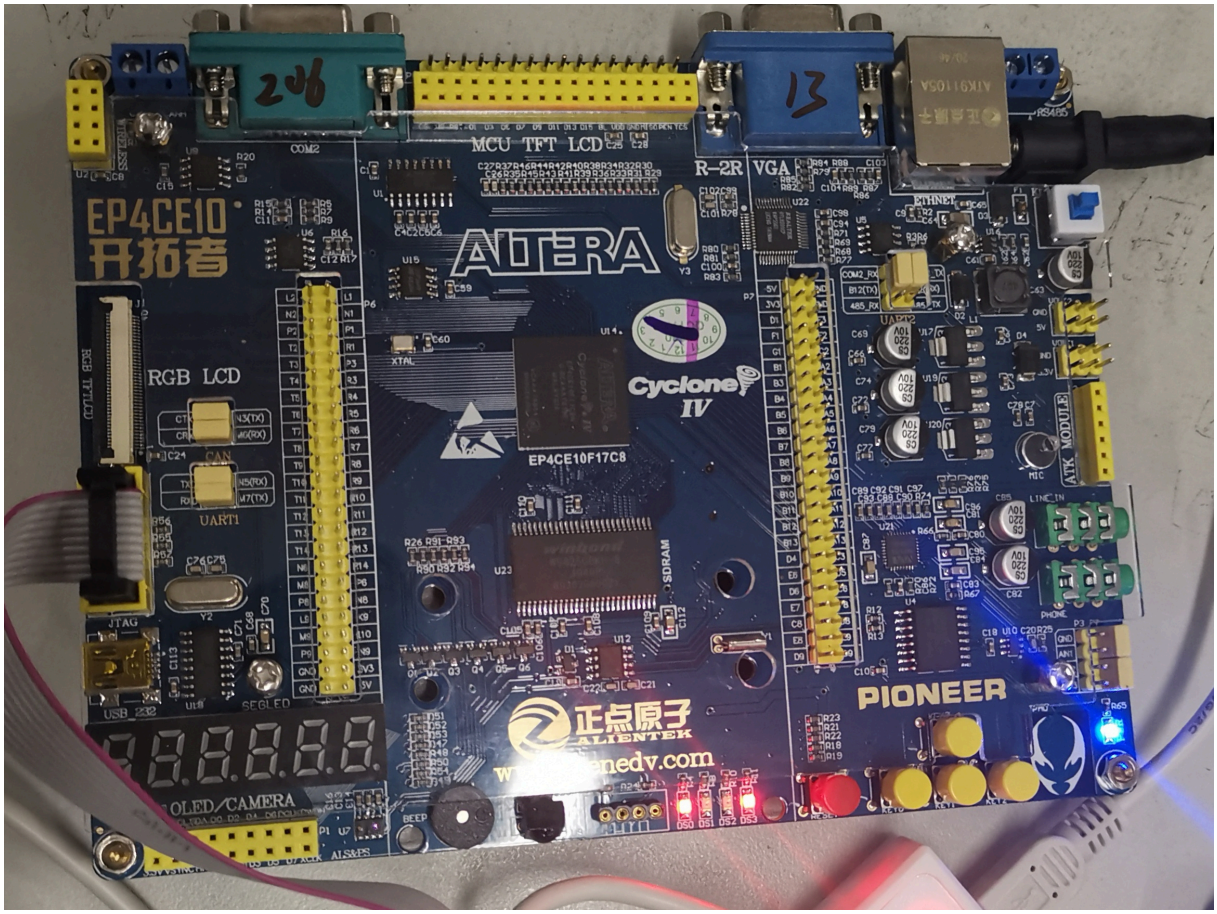

图 3: S0 工作状态示意图

6 实验代码

6.1 module 1

```
`timescale  1ns/1ns
module  clk_div(
                    clk,
                    rst,
                    clk_1k
                    );

input clk,rst;
output clk_1k;
reg clk_1k_reg=1'b0;
reg[15:0] clk_div_cnt;

always @(posedge clk or negedge rst)
begin
    if(rst==1'b0)
        clk_div_cnt<=0;
    else if(clk_div_cnt==24999)
```

```
        clk_div_cnt<=0;
    else
        clk_div_cnt<=clk_div_cnt+1;
end

always @(posedge clk or negedge rst)
begin
    if(rst==1'b0)
        clk_1k_reg<=1'b0;
    else if(clk_div_cnt==24999)
        clk_1k_reg<=~clk_1k_reg;
end

assign clk_1k=clk_1k_reg;
// assign clk_1k=clk; // 仿真时上一行注释，这一行取消注释

endmodule
```

6.2 module 2

```
`timescale  1ns/1ns
module  key_scan(
                    clk_1k,
                    rst,
                    key_in,
                    key_delt
                    );

input clk_1k,rst;
input key_in;
output key_delt;
reg key_in_dly1,key_in_dly2;
reg[15:0] key_in_shift;
reg key_filter,key_filter_dly1;
reg[1:0] rst_cnt;

// syn key_in
always @(posedge clk_1k)
begin
    key_in_dly1<=key_in;
    key_in_dly2<=key_in_dly1;
end

always @(posedge clk_1k)
begin
```

```
    key_in_shift<={key_in_shift[14:0],key_in_dly2};
end

always @(posedge clk_1k or negedge rst)
begin
    if(rst==1'b0)
        key_filter<=1'b1;
    else if(key_in_shift==0)
        key_filter<=1'b0;
    else if(key_in_shift==16'b1111111111111111)
        key_filter<=1'b1;
    else
        key_filter<=key_filter;
end

always @(posedge clk_1k)
begin
    key_filter_dly1<=key_filter;
end

assign key_delt=(~key_filter_dly1) & key_filter;

endmodule
```

6.3 module 3

```
`timescale  1ns/1ns
module  led_trans(
                    clk,
                    rst,
                    key0,
                    key1,
                    key2,
                    key3,
                    led0_red,
                    led0_green,
                    led1_red,
                    led1_green
                    );

input clk,rst;
input key0,key1,key2,key3;
output led0_red,led0_green,led1_red,led1_green;

parameter       S0  =   4'b0000;
```

```
parameter       S1  =   4'b0001;
parameter       S2  =   4'b0011;
parameter       S3  =   4'b0111;
reg[3:0] cur_state;
reg[10:0] time_cnt;
reg led0_red_reg,led0_green_reg,led1_red_reg,led1_green_reg;
wire clk_1k;
wire key_filter0,key_filter1,key_filter2,key_filter3;


clk_div clk_div_inst(
                        .clk        (clk),
                        .rst        (rst),
                        .clk_1k     (clk_1k)
                        );

key_scan key_scan_inst0(
                        .clk_1k     (clk_1k),
                        .rst        (rst),
                        .key_in     (key0),
                        .key_delt   (key_filter0)
                        );

key_scan key_scan_inst1(
                        .clk_1k     (clk_1k),
                        .rst        (rst),
                        .key_in     (key1),
                        .key_delt   (key_filter1)
                        );

key_scan key_scan_inst2(
                        .clk_1k     (clk_1k),
                        .rst        (rst),
                        .key_in     (key2),
                        .key_delt   (key_filter2)
                        );

key_scan key_scan_inst3(
                        .clk_1k     (clk_1k),
                        .rst        (rst),
                        .key_in     (key3),
                        .key_delt   (key_filter3)
                        );

// state
always @(posedge clk_1k or negedge rst)
begin
```

```
67      if(rst==1'b0)
68          cur_state<=S0;
69      else if(key_filter0==1'b1)
70          cur_state<=S0;
71      else if(key_filter1==1'b1)
72          cur_state<=S1;
73      else if(key_filter2==1'b1)
74          cur_state<=S2;
75      else if(key_filter3==1'b1)
76          cur_state<=S3;
77      else
78          case(cur_state)
79          S0 :    if(time_cnt==1999)
80                      cur_state<=S1;
81          S1 :    if(time_cnt==999)
82                      cur_state<=S2;
83          S2 :    if(time_cnt==1999)
84                      cur_state<=S3;
85          S3 :    if(time_cnt==999)
86                      cur_state<=S0;
87          default :   cur_state<=S0;
88          endcase
89  end
90
91  // time_cnt
92  always @(posedge clk_1k or negedge rst)
93  begin
94      if(rst==1'b0)
95          time_cnt<=0;
96      else if(key_filter0==1'b1 | key_filter1==1'b1 | key_filter2==1'b1 |
            key_filter3==1'b1)
97          time_cnt<=0;
98      else
99          case(cur_state)
100         S0 :    if(time_cnt==1999)
101                     time_cnt<=0;
102                 else
103                     time_cnt<=time_cnt+1;
104         S1 :    if(time_cnt==999)
105                     time_cnt<=0;
106                 else
107                     time_cnt<=time_cnt+1;
108         S2 :    if(time_cnt==1999)
109                     time_cnt<=0;
110                 else
111                     time_cnt<=time_cnt+1;
112         S3 :    if(time_cnt==999)
```

```
                    time_cnt<=0;
                else
                    time_cnt<=time_cnt+1;
        default :   time_cnt<=0;
        endcase
end

// led0_red
always @(posedge clk_1k)
begin
    case(cur_state)
    S0 :    led0_red_reg<=1'b1;
    S1 :    led0_red_reg<=1'b1;
    S2 :    led0_red_reg<=1'b0;
    S3 :    led0_red_reg<=1'b0;
    default :   led0_red_reg<=1'b0;
    endcase
end

// led0_green
always @(posedge clk_1k)
begin
    case(cur_state)
    S0 :    led0_green_reg<=1'b0;
    S1 :    led0_green_reg<=1'b0;
    S2 :    led0_green_reg<=1'b1;
    S3 :    led0_green_reg<=1'b0;
    default :   led0_green_reg<=1'b0;
    endcase
end

// led1_red
always @(posedge clk_1k)
begin
    case(cur_state)
    S0 :    led1_red_reg<=1'b0;
    S1 :    led1_red_reg<=1'b0;
    S2 :    led1_red_reg<=1'b1;
    S3 :    led1_red_reg<=1'b1;
    default :   led1_red_reg<=1'b0;
    endcase
end

// led1_green
always @(posedge clk_1k)
begin
    case(cur_state)
```

```
    S0 :    led1_green_reg<=1'b1;
    S1 :    led1_green_reg<=1'b0;
    S2 :    led1_green_reg<=1'b0;
    S3 :    led1_green_reg<=1'b0;
    default :   led1_green_reg<=1'b0;
    endcase
end

assign led0_red=led0_red_reg;
assign led0_green=led0_green_reg;
assign led1_red=led1_red_reg;
assign led1_green=led1_green_reg;

endmodule
```

6.4 仿真测试代码

```
// Copyright (C) 1991-2013 Altera Corporation
// Your use of Altera Corporation's design tools, logic functions
// and other software and tools, and its AMPP partner logic
// functions, and any output files from any of the foregoing
// (including device programming or simulation files), and any
// associated documentation or information are expressly subject
// to the terms and conditions of the Altera Program License
// Subscription Agreement, Altera MegaCore Function License
// Agreement, or other applicable license agreement, including,
// without limitation, that your use is for the sole purpose of
// programming logic devices manufactured by Altera and sold by
// Altera or its authorized distributors.  Please refer to the
// applicable agreement for further details.

// *********************************************************************
// This file contains a Verilog test bench template that is freely editable
    to
// suit user's needs .Comments are provided in each section to help the
    user
// fill out necessary details.
// *********************************************************************
// Generated on "10/30/2021 10:55:37"

// Verilog Test Bench template for design : led_trans
//
// Simulation tool : ModelSim-Altera (Verilog)
//

```

```
`timescale 100 us/ 100 us
module bd();
// constants
// general purpose registers
reg eachvec;
// test vector input registers
reg clk;
reg key0;
reg key1;
reg key2;
reg key3;
reg rst;
// wires
wire led0_green;
wire led0_red;
wire led1_green;
wire led1_red;

// assign statements (if any)
led_trans i1 (
// port map - connection between master ports and signals/registers
    .clk(clk),
    .key0(key0),
    .key1(key1),
    .key2(key2),
    .key3(key3),
    .led0_green(led0_green),
    .led0_red(led0_red),
    .led1_green(led1_green),
    .led1_red(led1_red),
    .rst(rst)
);
initial
begin
#0
clk=1'b0;
rst=1'b0;
key0=1'b1;
key1=1'b1;
key2=1'b1;
key3=1'b1;
#1
rst=1'b1;

#110000
key0=1'b0;
#200
```

```
74  key0=1'b1;
75
76  #110000
77  key1=1'b0;
78  #200
79  key1=1'b1;
80
81  #110000
82  key2=1'b0;
83  #200
84  key2=1'b1;
85
86  #110000
87  key3=1'b0;
88  #200
89  key3=1'b1;
90
91  //#300000000
92  //$stop;
93  // --> end
94  $display("Running␣testbench");
95  end
96  always
97  #5
98  clk=~clk;
99  begin
100 // code executes for every event on sensitivity list
101 // insert code here --> begin
102
103 // -->
104 end
105 endmodule
```

Appendices

附录 A　管脚分配

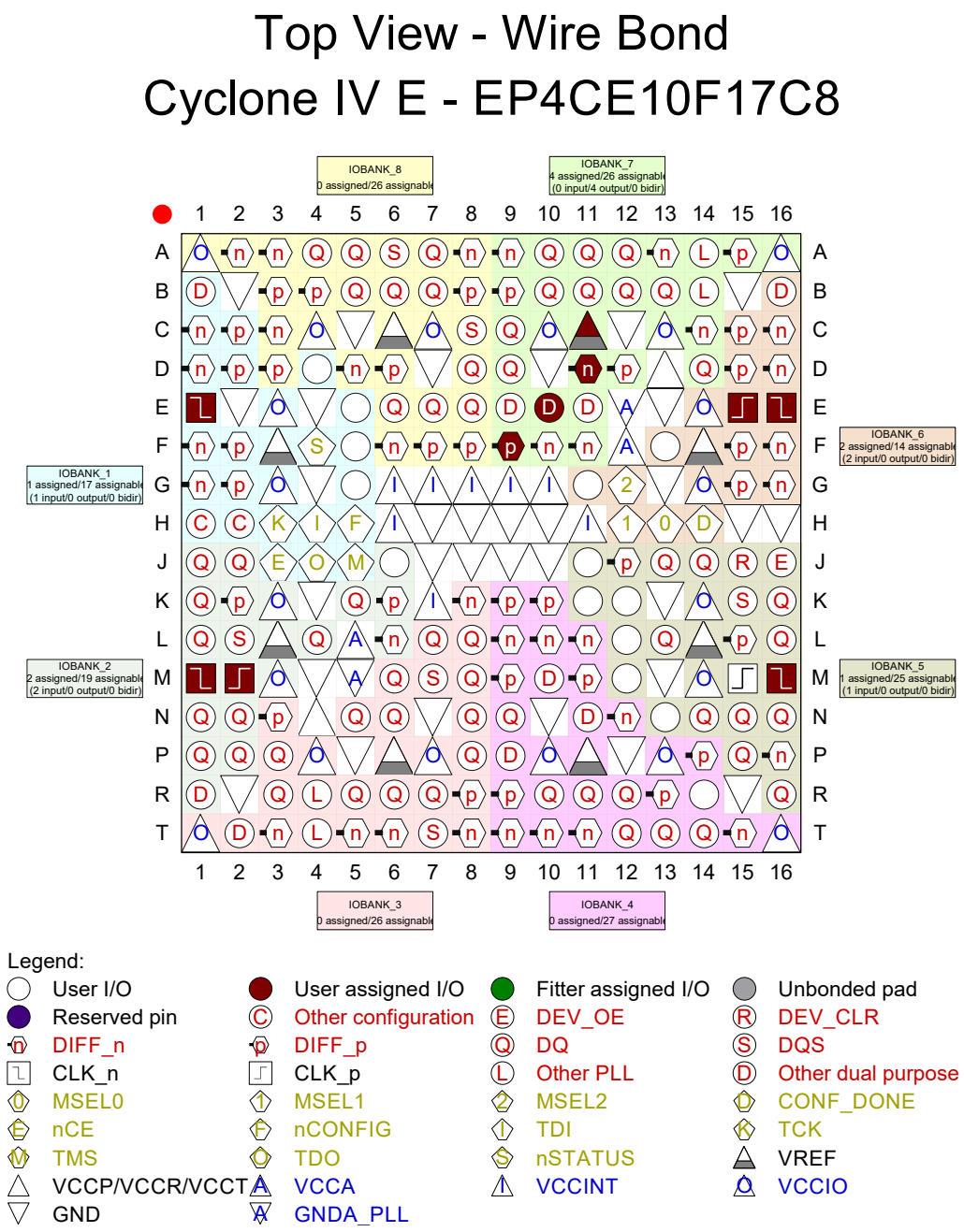

图 4: 管脚分配图