# 数字电路与数字系统实验

**EX08:状态机及键盘输入**

191220029 傅小龙

周一5-6节班

[1830970417@qq.com](mailto:1830970417@qq.com)

2020年11月16日

# **目录**

1. [实验内容](#_实验内容)……………………………………………………………………………………3
   1. [实验要求](#_1.1实验要求)………………………………………………………………………………3
   2. [实验工具](#_1.2实验工具)………………………………………………………………………………3
2. [实验过程](#_实验过程)……………………………………………………………………………………3

2.1 [模型概述](#_2.2.1模型概述)………………………………………………………………………………3

2.2 [数字抽象](#_2.2.2数字抽象)………………………………………………………………………………4

2.3 [建立模型](#_2.2.3建立模型)………………………………………………………………………………6

2.4 [分析/综合](#_2.4分析/综合)…………………………………………………………………………… 6

2.5 [分配引脚](#_2.6分配引脚)………………………………………………………………………………7

2.6 [全编译](#_2.7全编译)…………………………………………………………………………………9

三、[实验总结](#_实验总结)……………………………………………………………………………………9

四、附……………………………………………………………………………………………9

4.1七段数码管编码器的Verilog实现…………………………………………………9

#### 实验内容

##### 1.1实验要求

自行设计状态机，实现单个按键的 ASCII 码显示。

七段数码管低两位显示当前按键的键码，中间两位显示对应的 ASCII 码（转换可以考虑自行设计一个 ROM 并初始化）。只需完成字符和数字键的输入，不需要实现组合键和小键盘。当按键松开时，七段数码管的低四位全灭。

七段数码管的高两位显示按键的总次数。按住不放只算一次按键。只考虑

顺序按下和放开的情况，不考虑同时按多个键的情况。

支持 Shift，CTRL 等组合键，在 LED 上显示组合键是否按下的状态指示；

支持 Shift 键与字母/数字键同时按下，相互不冲突；

支持输入大写字符，显示对应的 ASCII 码；

##### 1.2实验工具

软件环境：

设计、编译、仿真：Quartus Prime Version 17.1.0 Build 590 10/25/2017 SJ Lite Edition

DE10\_Standard\_SystemBuilder

硬件环境： DE-10 Standard开发平台

FPGA芯片： Cyclone V 5CSXFC6D6F31C6

#### 实验过程

##### 2.1模型概述

在exp08.pdf表 8 4: 键盘控制器给出的Verilog代码的基础上实现对键盘输出的键码及其对应的ASCII码的显示，以及能够对按键次数进行统计.

##### 2.2数字抽象

1. 输入:

数据输入[7:0]datain: 键码输入.

状态输入shift\_state: shift按键状态.

状态输入caps\_state: capslock按键状态.

状态输入ctrl\_state: ctrl按键状态.

1. 输出:

数据输出[7:0]dataout: ASCII码输出.

下表\图给出了以上输入输出信号在DE10平台对应的信号：

|  |  |  |
| --- | --- | --- |
|  | 信号名称 | DE-10平台信号 |
| 输  入 | [7:0]datain | [7:0]temp |
| shift\_state | shift\_state |
| caps\_state | caps\_state |
| ctrl\_state | ctrl\_state |
| 输出 | [7:0]dataout | [7:0]ascii |

\*：关于表项中的DE-10平台信号的含义详见2.3节中的相关内容.

表 2-1:键盘码转ASCII码输入输出信号与DE10平台信号对应关系

##### 2.3建立模型

kbdecoder模块中，相应键码的ASCII码用类似ROM的形式存放，与键码是一一对应关系. 再根据shift, capslock键的状态对ASCII码的输出进行赋值.

下面是kbdecoder模块的相关代码：

module kbdecoder(datain, dataout, shift\_state, caps\_state, ctrl\_state);

input [7:0]datain;

input shift\_state; //shift键是否被按下

input caps\_state; //caps键是否被按下

input ctrl\_state; //ctrl键是否被按下

output reg [7:0]dataout;

reg [7:0] asc [255:0];

reg [7:0] ascii\_shift [255:0];

reg [7:0] ascii\_caps [255:0];

reg [7:0] temp;

initial

begin

$readmemh(".\\init\\ascii\_init.txt", asc, 0, 255);

$readmemh(".\\init\\ascii\_init\_shift.txt", ascii\_shift, 0, 255);

$readmemh(".\\init\\ascii\_init\_caps.txt", ascii\_caps, 0, 255);

end

always @(\*)

begin

if(shift\_state && !caps\_state) begin

dataout <= ascii\_shift[datain];

end

else if(caps\_state && !shift\_state) begin

dataout <= ascii\_caps[datain];

end

else dataout <= asc[datain];

end

endmodule

|  |  |
| --- | --- |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 | begin  $readmemh(".\\init\\ascii\_init.txt", asc, 0, 255);  $readmemh(".\\init\\ascii\_init\_shift.txt", ascii\_shift, 0, 255);  $readmemh(".\\init\\ascii\_init\_caps.txt", ascii\_caps, 0, 255);    end    always @(\*)  begin  if(shift\_state && !caps\_state) begin  dataout <= ascii\_shift[datain];  end  else if(caps\_state && !shift\_state) begin  dataout <= ascii\_caps[datain];  end  else dataout <= asc[datain];  end  endmodule |
| 16 |  |
| 17 |  |
| 18 |  |
| 19 |  |
| 20 |  |
| 21 |  |
| 22 |  |
| 23 |  |
| 24 |  |
| 25 |  |
| 26 |  |
| 27 |  |
| 28 |  |
| 29 |  |
| 30 |  |
| 31 |  |
| 32 |  |
| 33 |  |
| 34 |  |
| 35 |  |

初始化文件详见项目文件夹的init/文件夹中的.txt文件.

对于shift, capslock按键的状态设计和next\_data\_n信号的设置在顶层文件中给出如下实现：

always@(posedge CLOCK\_50) begin

if(ready == 1 && next\_data\_n == 1)begin

temp <= keycode;

next\_data\_n <= 0;

if(keycode == 8'hf0) begin//realse

release\_flag <= 1;

cnt <= cnt + 1;

end

else if(keycode == 8'h12 || keycode == 8'h59) begin//shift

if(release\_flag) begin

shift\_state <= 0;

e\_out <= 0;

release\_flag <= 0;

end

else begin

shift\_state <= 1;

e\_out <= 1;

end

end

ctrl\_state <= 0;

e\_out <= 0;

release\_flag <= 0;

end

else begin

ctrl\_state <= 1;

e\_out <= 1;

end

end

else begin

if(release\_flag) begin

e\_out <= 0;

release\_flag <= 0;

if(keycode == 8'h58) caps\_state <= ~caps\_state;

else;

end

else e\_out <= 1;

end

end

else begin

next\_data\_n <= 1;

end

end

|  |  |
| --- | --- |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |
| 16 |  |
| 17 |  |
| 18 |  |
| 19 |  |
| 20 |  |
| 21  else if(keycode == 8'h14) begin //ctrl  if(release\_flag) begin  ctrl\_state <= 0;  e\_out <= 0;  release\_flag <= 0;  end  else begin  ctrl\_state <= 1;  e\_out <= 1;  end  end  else begin  if(release\_flag) begin  e\_out <= 0;  release\_flag <= 0;  if(keycode == 8'h58) caps\_state <= ~caps\_state;  else;  end  else e\_out <= 1;  end  end  else begin  next\_data\_n <= 1;  end  end |  |
| 22 |  |
| 23 |  |
| 24 |  |
| 25 |  |
| 26 |  |
| 27 |  |
| 28 |  |
| 29 |  |
| 30 |  |
| 31 |  |
| 32 |  |
| 33 |  |
| 34 |  |
| 35 |  |
| 36 |  |
| 37 |  |
| 38 |  |
| 39 |  |
| 40 |  |
| 41 |  |
| 42 |  |
| 43 |  |
| 44 |  |

需要注意的是keyboard模块给出的键盘码输出并不一定是有效的，只有在ready和next\_data\_n信号都为1时才有效（Line2）.获取有效的键码输出后，需要将next\_data\_n信号置0以准备接受下一个有效信号.

对于松开按键后HEX数码管熄灭的实现：通过reg型变量release\_flag记录是否有f0键码输出. 在接收到f0键码的下一个时钟周期将数码管的使能端输入置0（Line 5-8, 33-37）.松开按键则计数加1.关于数码管译码器的设计详见附-4.1.

对于shift, ctrl按键状态的设置：若release\_flag为零，则说明是按键按下，将对应的标志信号置1，否则置0并设置数码管的使能信号为0(Line 9-31).

对于caps\_lock按键状态的设置：caps\_lock键按下、松开后caps\_lock按键的状态改变一次（Line 33-38）.

##### 2.4分析/综合

分析/综合实验成功，如下图所示：

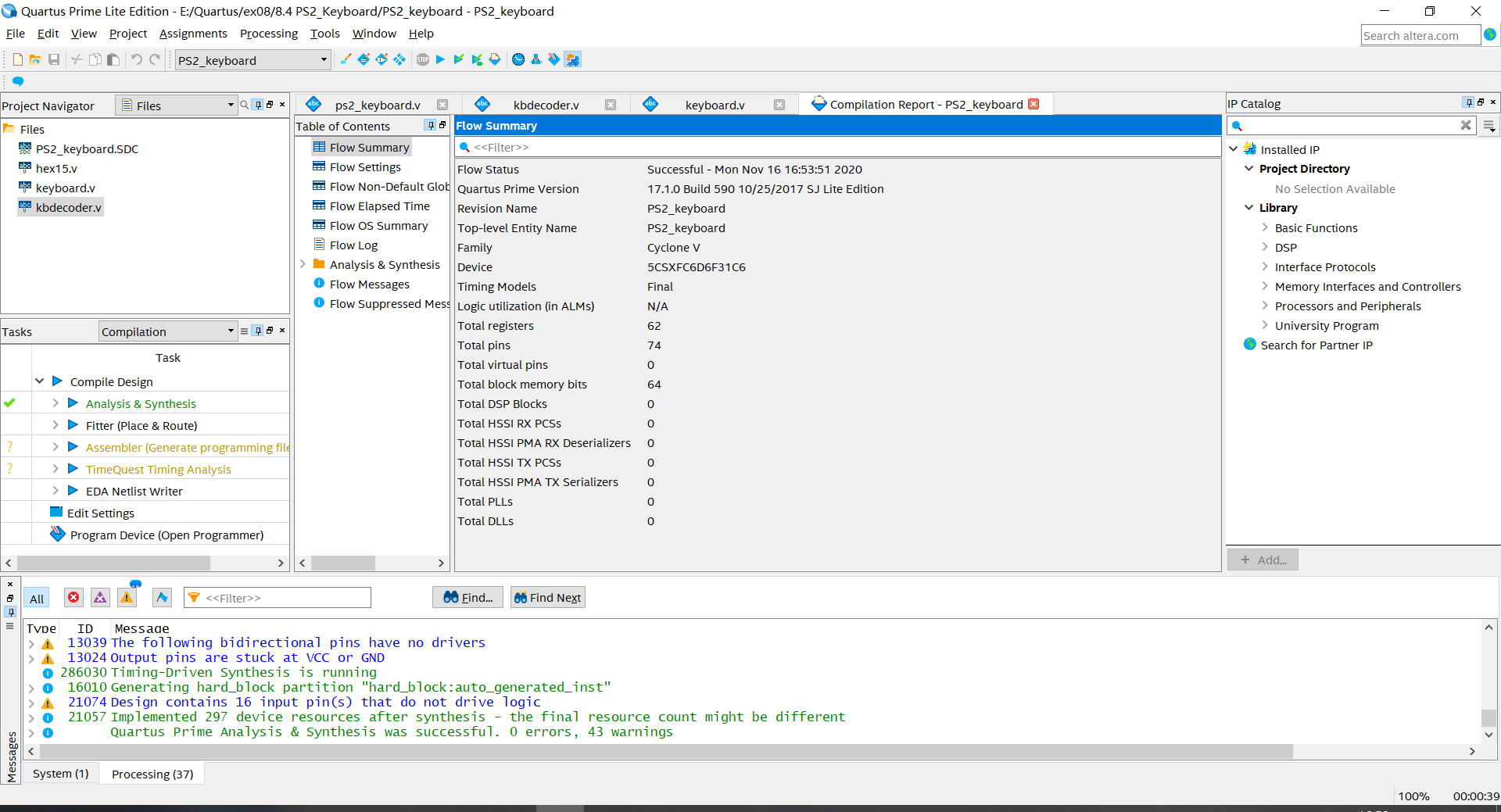


图2-4-1：分析/综合成功

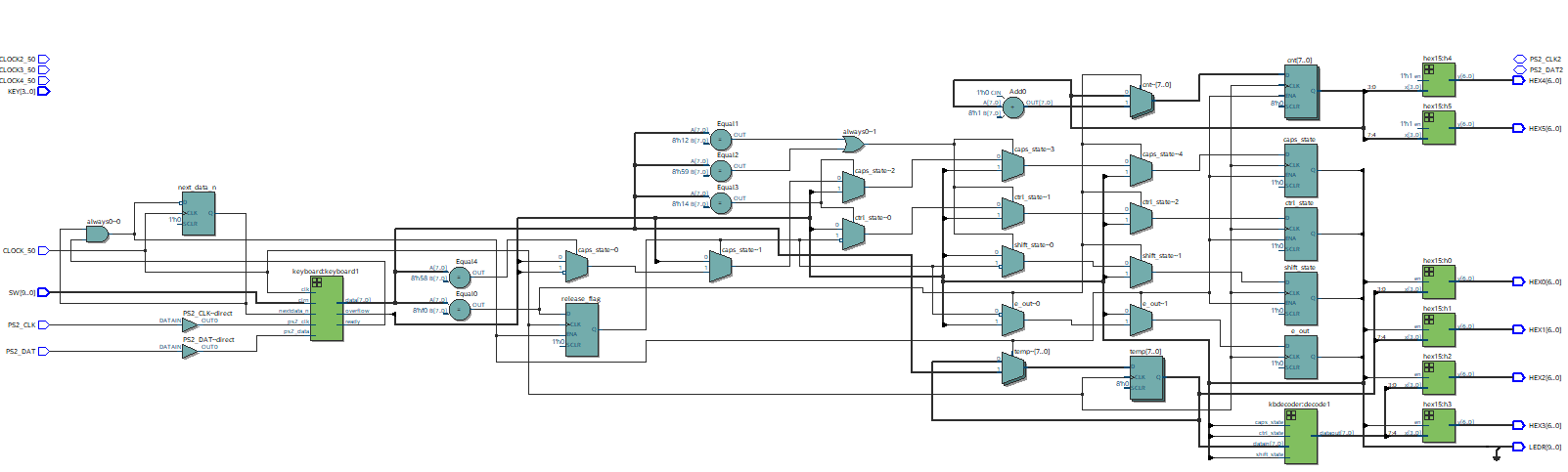
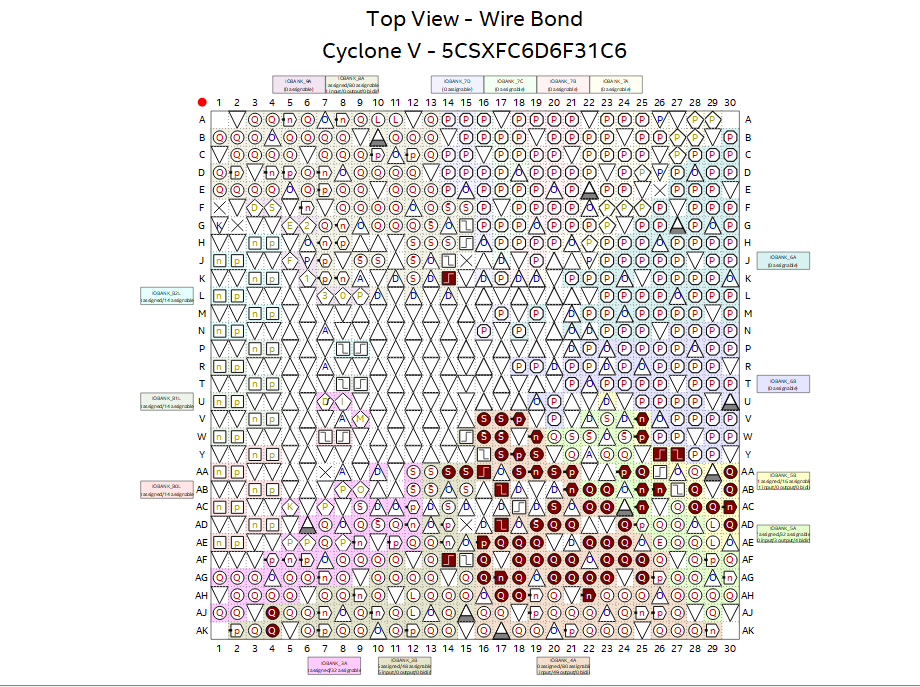
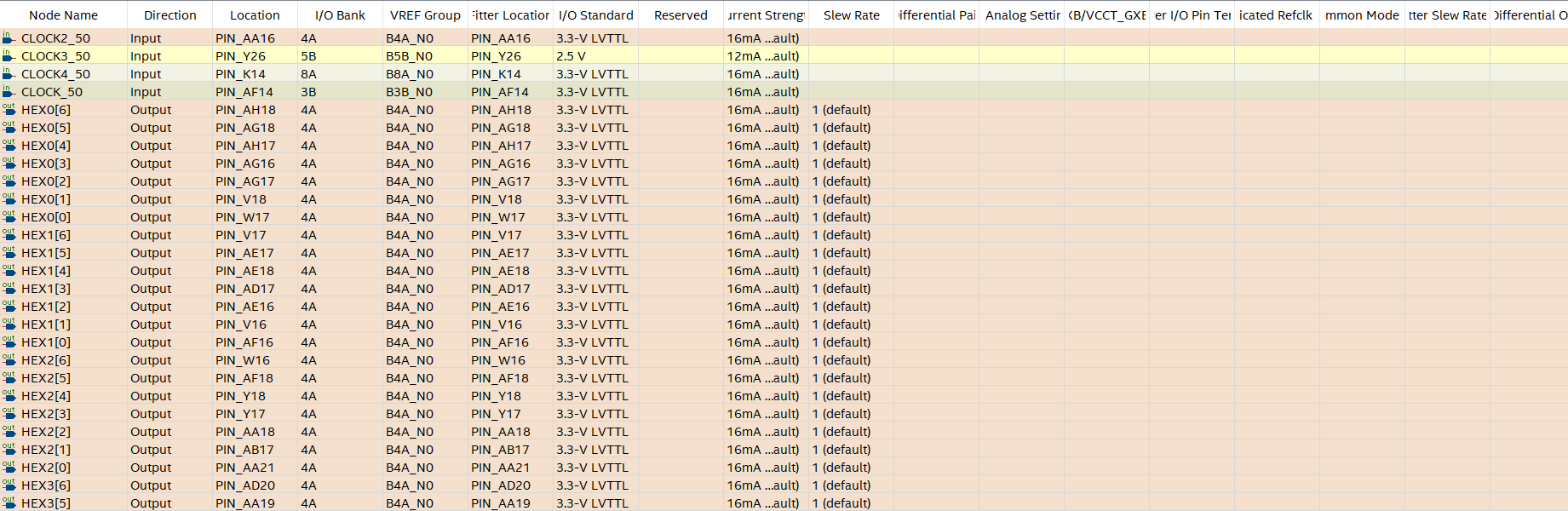


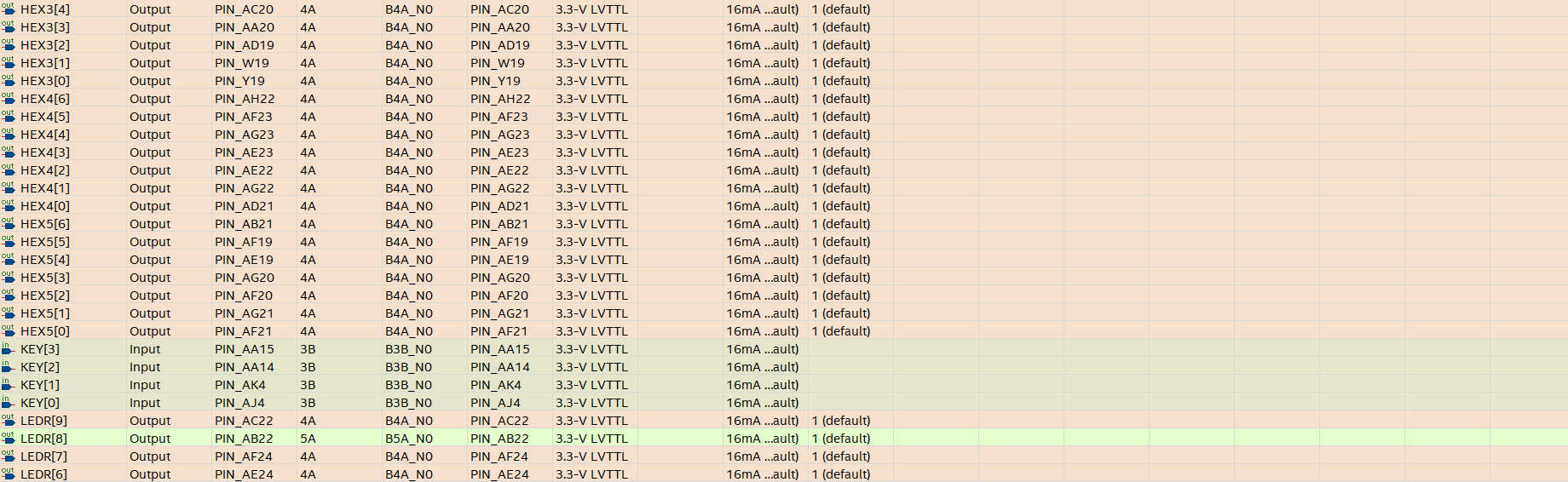
图 2-4-2:RTL视图

##### 2.5分配引脚

引脚分配使用DE10\_Standard\_SystemBuilder生成。







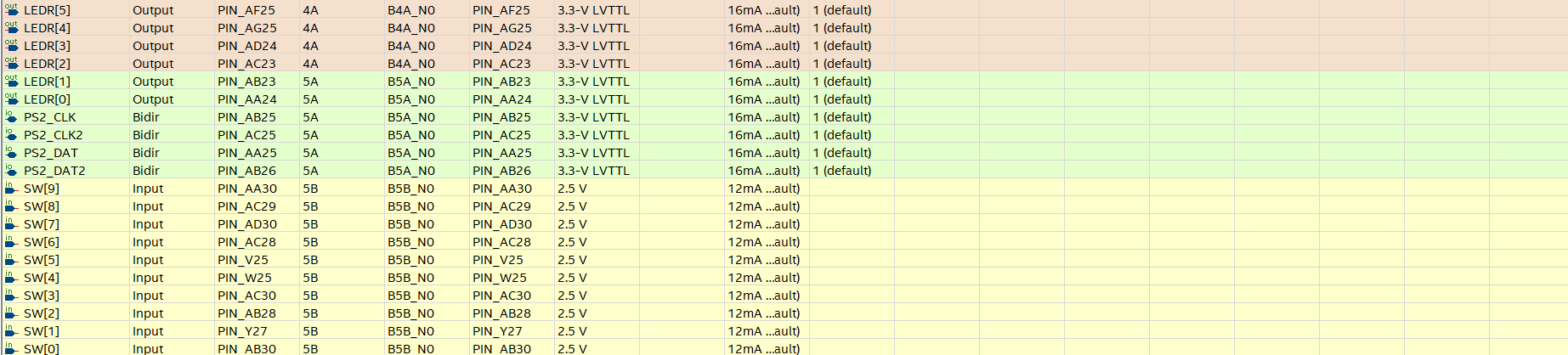


图2-5引脚分配图

##### 2.6全编译

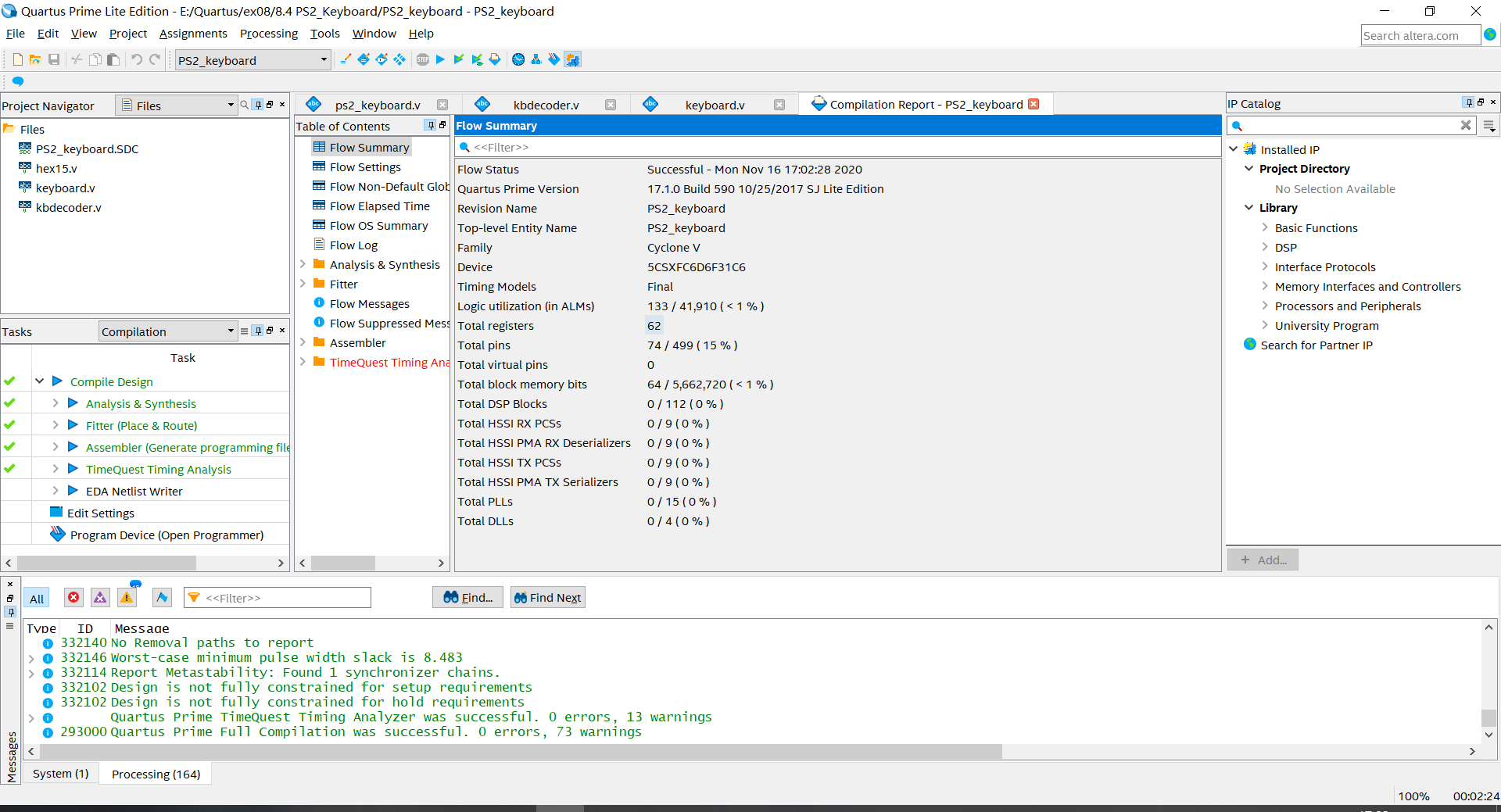


图2-6全编译成功

#### 实验总结

本次实验主要学习实践了状态机的设计、应用. 还学习了PS2键盘的工作方式并在以键盘码接收模块的基础上设计了将键盘码转为ascii码、按键次数的状态机进行了设计.需要注意的是只有正确理解并运用数据的输入输出指示信号的时序逻辑关系才能使设计的状态机正确工作.

#### 附

##### 4.1 七段数码管编码器的Verilog实现

module hex15(x, y, en);

input [3:0]x;

output reg [6:0]y;

input en;

always @(\*) begin

if(!en) y = 7'b1111111;

else case(x)

4'b0010:y = 7'b0100100;

4'b0011:y = 7'b0110000;

4'b0100:y = 7'b0011001;

4'b0101:y = 7'b0010010;

4'b0110:y = 7'b0000010;

4'b0111:y = 7'b1111000;

4'b1000:y = 7'b0000000;

4'b1001:y = 7'b0010000;

4'b1010:y = 7'b0001000;

4'b1011:y = 7'b0000011;

4'b1100:y = 7'b1000110;

4'b1101:y = 7'b0100001;

4'b1110:y = 7'b0000110;

4'b1111:y = 7'b0001110;

default: y = 7'b1111111;

endcase

end

endmodule

|  |  |
| --- | --- |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9  4'b0000:y = 7'b1000000;  4'b0001:y = 7'b1111001;  4'b0010:y = 7'b0100100;  4'b0011:y = 7'b0110000;  4'b0100:y = 7'b0011001;  4'b0101:y = 7'b0010010;  4'b0110:y = 7'b0000010;  4'b0111:y = 7'b1111000;  4'b1000:y = 7'b0000000;  4'b1001:y = 7'b0010000;  4'b1010:y = 7'b0001000;  4'b1011:y = 7'b0000011;  4'b1100:y = 7'b1000110;  4'b1101:y = 7'b0100001;  4'b1110:y = 7'b0000110;  4'b1111:y = 7'b0001110;  default: y = 7'b1111111;  endcase  end  endmodule |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |
| 16 |  |
| 17 |  |
| 19 |  |
| 20 |  |
| 21 |  |
| 22 |  |
| 23 |  |
| 24 |  |
| 25 |  |
| 26 |  |
| 27 |  |
| 28 |  |
| 29 |  |