# 数字电路与数字系统实验

**EX10:音频输出实验**

191220029 傅小龙

周一5-6节班

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

2020年11月29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)………………………………………………………………………………3

2.3 [建立模型](#_2.2.3建立模型)………………………………………………………………………………4

2.4 [分析/综合](#_2.4分析/综合)…………………………………………………………………………… 7

2.5 [分配引脚](#_2.6分配引脚)………………………………………………………………………………8

2.6 [全编译](#_2.7全编译)…………………………………………………………………………………10

三、[实验总结](#_实验总结)……………………………………………………………………………………10

#### 实验内容

##### 1.1实验要求

将之前实验实现的键盘与本实验的音频输出结合，实现一个简单的键盘电子琴功能。钢琴上的不同音高对应着不同的频率。我们可以根据按下的键的键值，决定播放的正弦的频率，从而实现电子琴的功能。

##### 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

在给出I2C\_Audio\_Config, I2S\_Audio, Sin\_Generator模块参考代码的基础上，用一个状态机接收键盘传送的键码，根据按下的按键数量对要输出的频率做和声处理，再将要输出的频率传给Sin\_Generator模块转换成模拟信号. 音量调整通过修改I2C\_Audio\_Config中发送给WM8731 芯片音量的数据值实现.

##### 2.2数字抽象

下图给出了各模块间的关系及其作用, 其中exp10audio为顶层文件名.

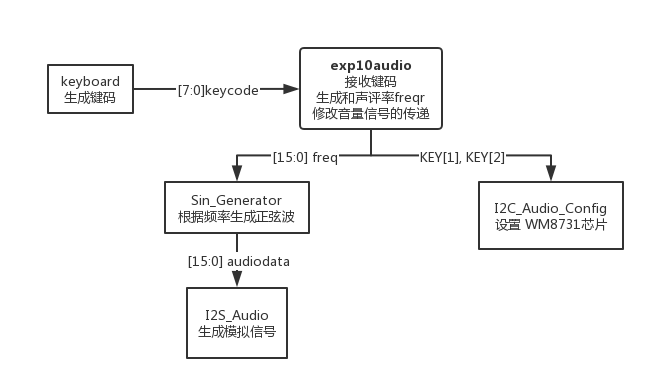


图2-2-1 模块关系示意图

##### 2.3建立模型

1. 状态机的设计

该状态机包含4个状态Q0, Q1, Q2, Q3, 分别表示有0个, 1个, 2个, 3个音的和声，并能够根据键盘模块的ready和next\_data\_n信号获取正确的键码以及根据状态数输出相应的频率.

状态机的状态转化条件以及相应的处理方式为：①当获取的键码为通码且与之前的通码不同时，状态数加一并将新获取的通码储存; ②当获取的键码为断码时，状态码减一, 并在存储通码的各变量中将松开按键的键码删除.

状态机的输出以当前状态取有效键码对应的频率取平均后参照ex10.pdf中计算递增值的方式得到应当传递给Sin\_Generator模块的频率.(对应下表中Line66开始的代码).

下面是上述状态机的相关代码：

always@(posedge CLOCK\_50) begin

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

//if(ready == 1)beginf

if(temp != keycode) neflag = 1;

else;

temp <= keycode;

next\_data\_n <= 0;

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

release\_flag <= 1;

//temp <= keycode;

case(state)

Q3: state <= Q2;

default: state <= Q0;

endcase;

end

else begin

if(release\_flag) begin

e\_out <= 0;

release\_flag <= 0;

if(key0 == keycode) begin

key0 = key1;

key1 = key2;

key2 = 0;

end

else if(key1 == keycode) begin

key1 = key2;

key2 = 0;

end

else if(key2 == keycode)begin

key2 = 0;

end

else;

end

else begin

e\_out <= 1;

if(neflag) begin

case(state)

Q0: begin state <= Q1; key0 = keycode; end

Q1: begin state <= Q2; key1 = key0; key0 = keycode; end

Q2: begin state <= Q3; key2 = key1; key1 = key0; key0 = keycode; end

Q3: begin state <= Q3; key2 = key1; key1 = key0; key0 = keycode; end

default: state <= 0;

endcase

neflag = 0;

end

else ;

end

end

end

else begin

next\_data\_n <= 1;

end

end

assign LEDR[8:7] = state;

always @(state) begin

if(!SW[0])

freqr = 16'h0400;

else begin

case(state)

Q0: freqr = 16'h0;

Q1: freqr = rom\_freq[key0] \* a1 / a2;

Q2: freqr = (rom\_freq[key1] / 2 + rom\_freq[key0] / 2) \* a1 / a2;

Q3: freqr = (rom\_freq[key2] / 3 + rom\_freq[key1] / 3 + rom\_freq[key0] / 3) \* a1 / a2;

default:freqr = 16'h0400;

endcase

end

end

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 | Q0: state <= Q0;  Q2: state <= Q1;  Q3: state <= Q2;  default: state <= Q0;  endcase;  end  else begin  if(release\_flag) begin  e\_out <= 0;  release\_flag <= 0;  if(key0 == keycode) begin  key0 = key1;  key1 = key2;  key2 = 0;  end  else if(key1 == keycode) begin  key1 = key2;  key2 = 0;  end  else if(key2 == keycode)begin  key2 = 0;  end  else;  end  else begin  e\_out <= 1;  if(neflag) begin  case(state)  Q0: begin state <= Q1; key0 = keycode; end  Q1: begin state <= Q2; key1 = key0; key0 = keycode; end  Q2: begin state <= Q3; key2 = key1; key1 = key0; key0 = keycode; end  Q3: begin state <= Q3; key2 = key1; key1 = key0; key0 = keycode; end  default: state <= 0;  endcase  neflag = 0;  end  else ;  end  end  end  next\_data\_n <= 1;  end  end  assign LEDR[8:7] = state;  always @(state) begin  if(!SW[0])  freqr = 16'h0400;  else begin  case(state)  Q0: freqr = 16'h0;  Q1: freqr = rom\_freq[key0] \* a1 / a2;  Q2: freqr = (rom\_freq[key1] / 2 + rom\_freq[key0] / 2) \* a1 / a2;  Q3: freqr = (rom\_freq[key2] / 3 + rom\_freq[key1] / 3 + rom\_freq[key0] / 3) \* a1 / a2;  default:freqr = 16'h0400;  endcase  end  end  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 |
| 13 |  |
| 14 |  |
| 15 |  |
| 16 |  |
| 17 |  |
| 18 |  |
| 19 |  |
| 20 |  |
| 21 |  |
| 22 |  |
| 23 |  |
| 24 |  |
| 25 |  |
| 26 |  |
| 27 |  |
| 28 |  |
| 29 |  |
| 30 |  |
| 31 |  |
| 32 |  |
| 33 |  |
| 34 |  |
| 35 |  |
| 36 |  |
| 37 |  |
| 40 |  |
| 41 |  |
| 42 |  |
| 43 |  |
| 44 |  |
| 45 |  |
| 46 |  |
| 47 |  |
| 48 |  |
| 49 |  |
| 50 |  |
| 51 |  |
| 52 |  |
| 53 |  |
| 54 |  |
| 55 |  |
| 56 |  |
| 57 |  |
| 58 |  |
| 59 | else begin |
| 60 | next\_data\_n <= 1;  end  end  assign LEDR[8:7] = state;  always @(state) begin  if(!SW[0])  freqr = 16'h0400;  else begin  case(state)  Q0: freqr = 16'h0;  Q1: freqr = rom\_freq[key0] \* a1 / a2;  Q2: freqr = (rom\_freq[key1] / 2 + rom\_freq[key0] / 2) \* a1 / a2;  Q3: freqr = (rom\_freq[key2] / 3 + rom\_freq[key1] / 3 + rom\_freq[key0] / 3) \* a1 / a2;  default:freqr = 16'h0400;  endcase  end  end |
| 61 |  |
| 62 |  |
| 63 |  |
| 64 |  |
| 65 |  |
| 66 |  |
| 67 |  |
| 68 |  |
| 69 |  |
| 70 |  |
| 71 |  |
| 72 |  |
| 73 |  |
| 74 |  |
| 75 |  |
| 76 |  |
| 77 |  |
| 78 |  |
| 79 |  |
| 80 |  |

对于音量的修改则通过操作开发板上的按钮KEY[1]和KEY[2]使I2C\_Audio\_Config模块给WM8731芯片传送不同的音频设置数据实现.这里共设置了6组不同音量的初始化数据, 其他参量均一致. 即只有下面这两行代码所示的左右音量不同，故这里不再全部展示.

audio\_reg1[3]= 7'h02; audio\_cmd1[3]=9'h40; //Left Volume

audio\_reg1[4]= 7'h03; audio\_cmd1[4]=9'h40; //Right Volume

在I2C\_Audio\_Config模块内设置reg型变量x以存储当前使用的初始化数据组的标号. 根据KEY[1]和KEY[2]输入改变x的值，KEY[1]和KEY[2]分别对应模块内的volume\_up, volume\_down接口. 在对变量x作出修改后，将在下一个I2C\_Audio\_Config模块的reset\_n重置信号有效时根据x的值选择相应的数据组修改WM8731芯片中的相应参数.

以下是对x变量修改的相关代码：

always @ (negedge volume\_up or negedge volume\_down) begin

if(!volume\_up) begin //set volume++

if(reg\_x < 4'h6) reg\_x <= reg\_x + 4'h1;

else ;

end

else if(!volume\_down) begin //set volume--

if(reg\_x > 4'h1) reg\_x <= reg\_x - 4'h1;

else ;

end

else ;

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 | end  else ;  end |
| 10 |  |
| 11 |  |

以下是对I2C\_Audio\_Config模块中要传送给WM8731芯片数据的部分代码的修改：

case(reg\_x)

4'h1:mi2c\_data <= {audio\_addr, audio\_reg1[cmd\_count], audio\_cmd1[cmd\_count]};

4'h2:mi2c\_data <= {audio\_addr, audio\_reg2[cmd\_count], audio\_cmd2[cmd\_count]};

4'h3:mi2c\_data <= {audio\_addr, audio\_reg3[cmd\_count], audio\_cmd3[cmd\_count]};

4'h4:mi2c\_data <= {audio\_addr, audio\_reg4[cmd\_count], audio\_cmd4[cmd\_count]};

4'h5:mi2c\_data <= {audio\_addr, audio\_reg5[cmd\_count], audio\_cmd5[cmd\_count]};

4'h6:mi2c\_data <= {audio\_addr, audio\_reg6[cmd\_count], audio\_cmd6[cmd\_count]};

endcase

|  |  |
| --- | --- |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |

##### 2.4分析/综合

分析/综合实验成功，如下图所示：

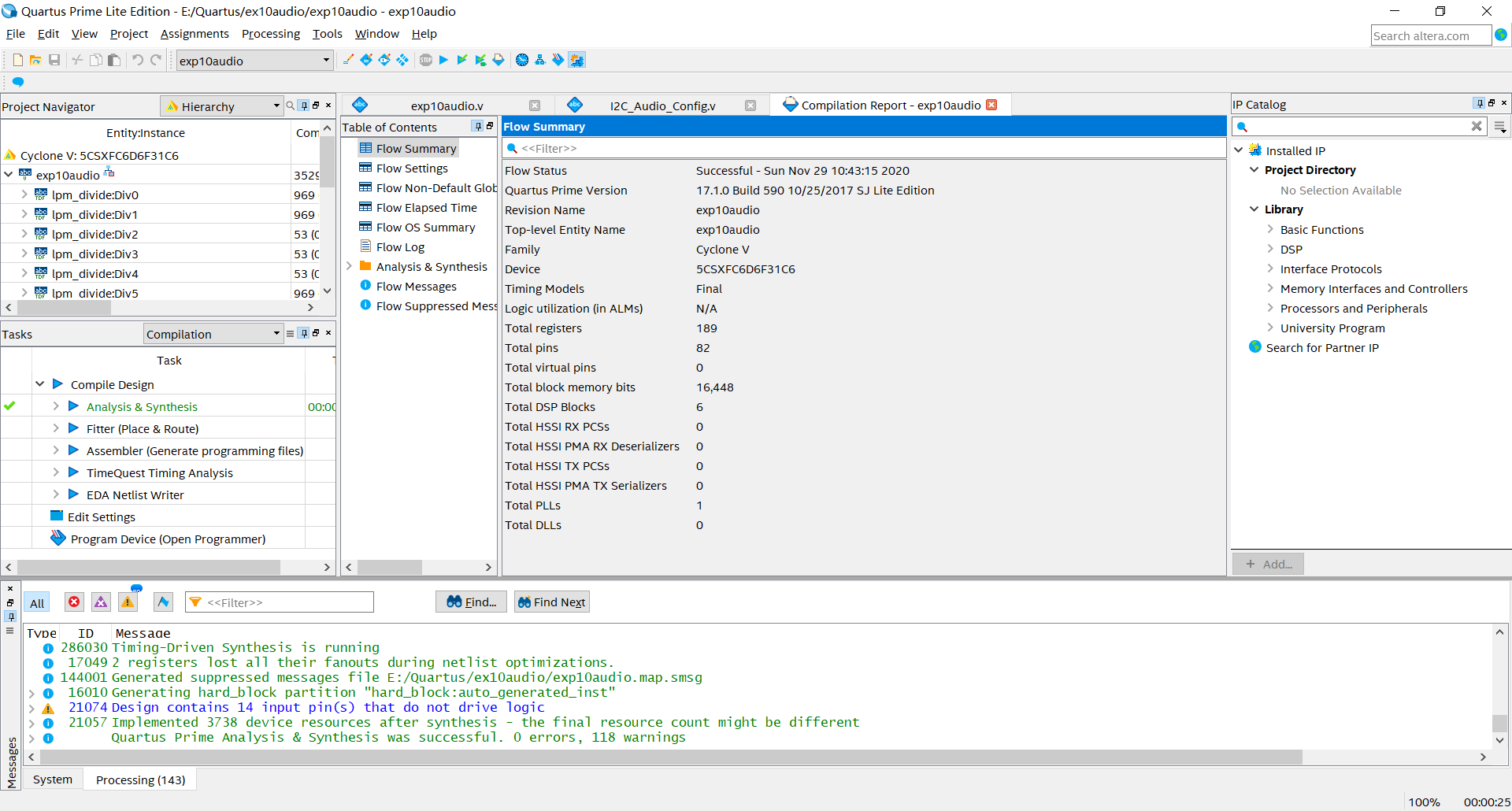


图2-4-1：分析/综合成功

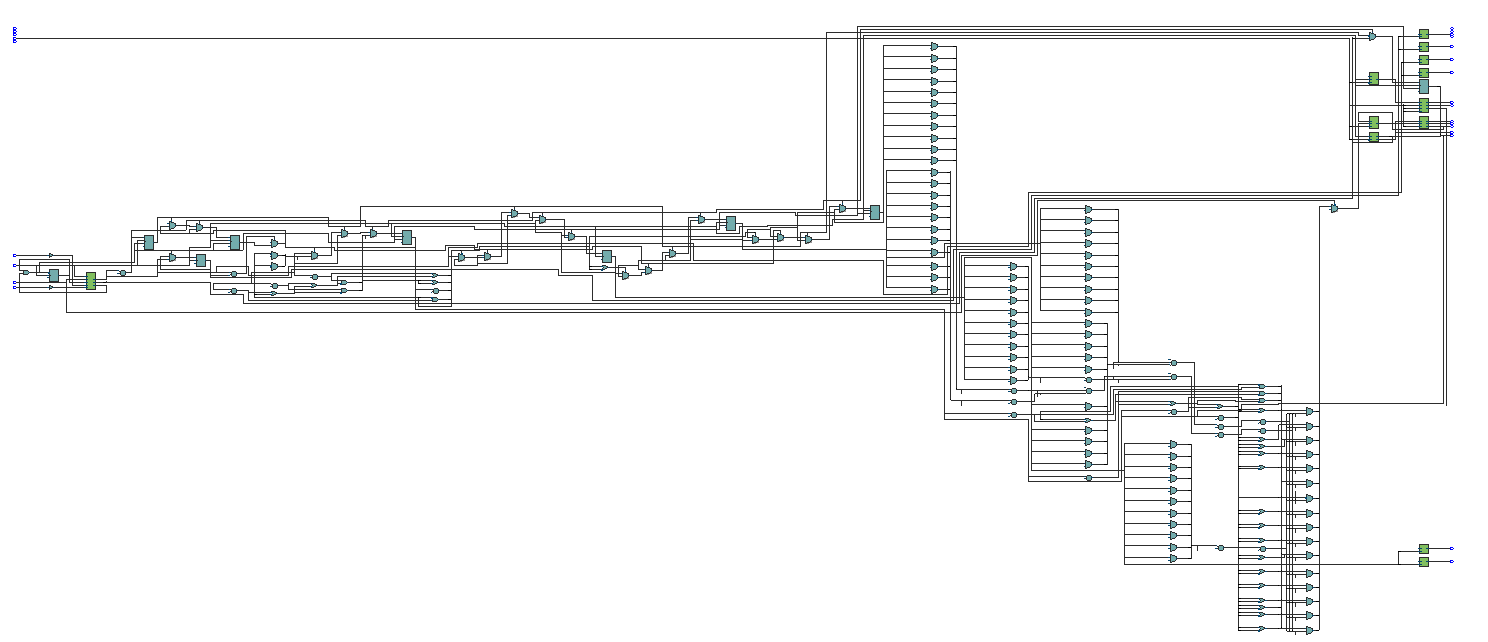
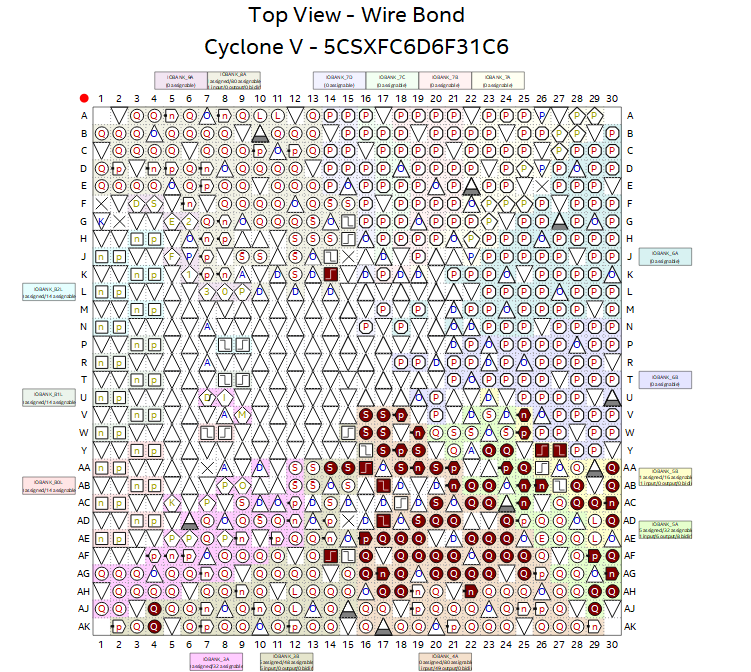
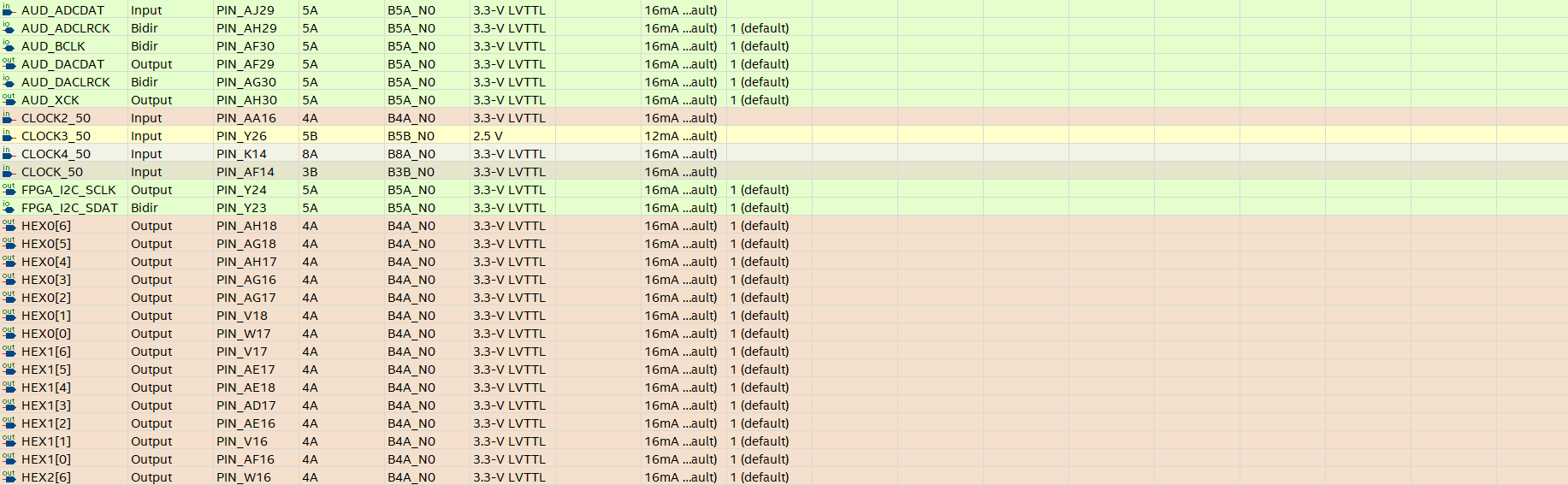


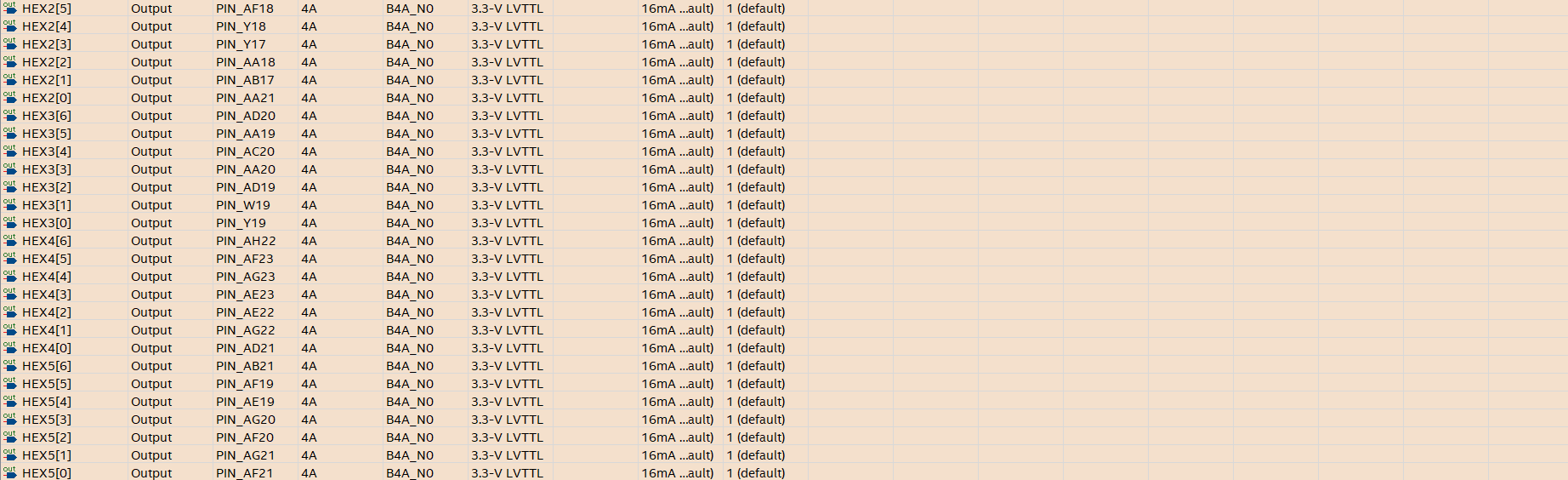
图 2-4-2:RTL视图

##### 2.5分配引脚

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







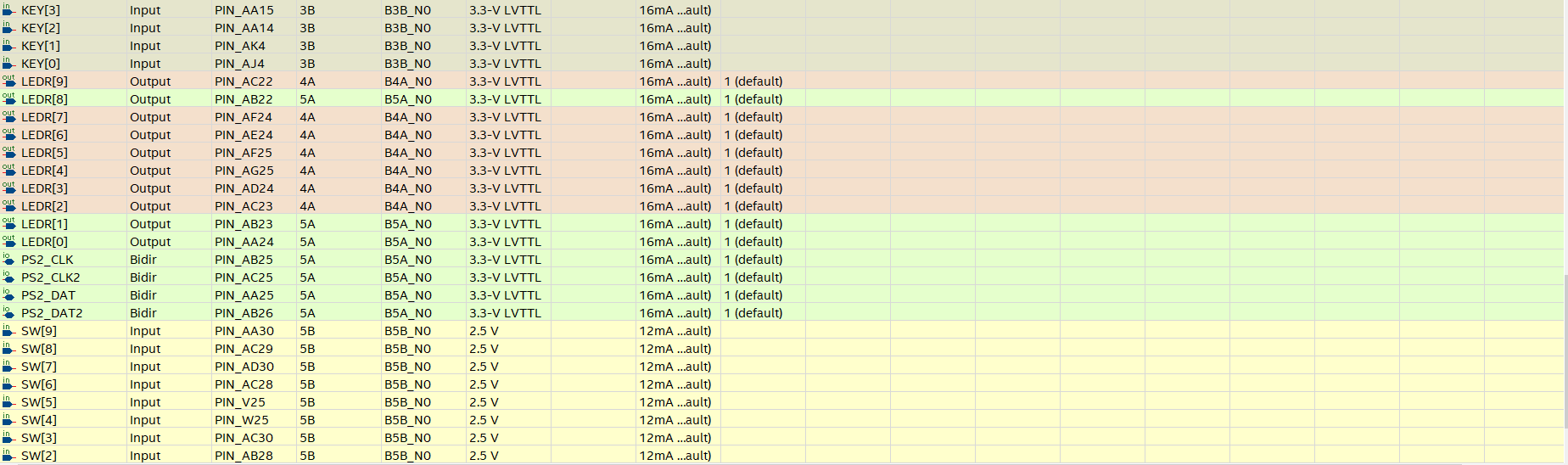




图2-5引脚分配图

##### 2.6全编译

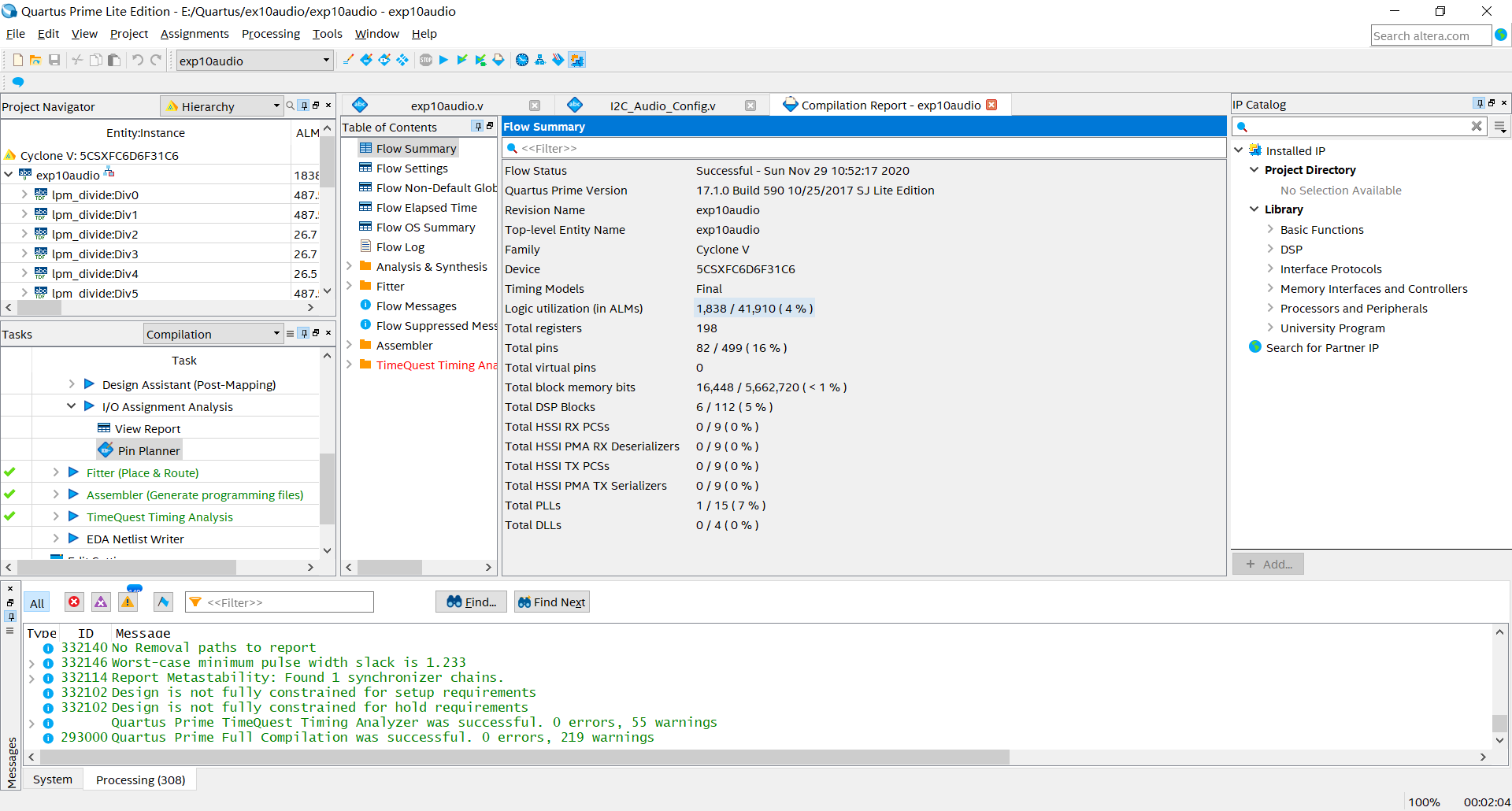


图2-6全编译成功

#### 实验总结

本次实验主要学习了音频的输出原理，并在给出的参考数字信号到模拟信号的转换和I2C音频输出芯片的初始化代码的基础上设计了能够用键盘“弹奏”的电子琴. 实验中复习并运用了之前键盘实验中学习的状态机的设计来实现和声输出，同时注意到设计过程中应注意运算的溢出和精度损失问题. 在设计音量变化功能时进一步深刻了解的RAM存储器在读写时应当遵守的时序逻辑.