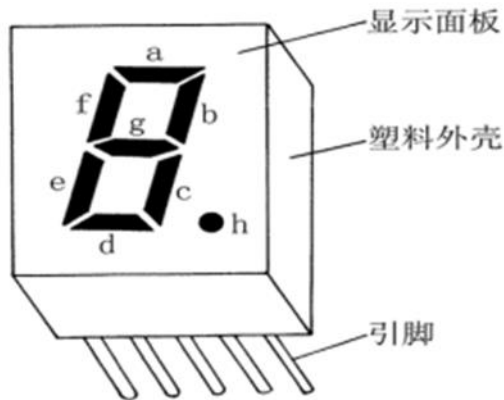


# 数码管动态扫描

## 一、项目背景

led 数码管 (LED Segment Displays) 是由多个发光二极管封装在一起组成“8”字型的器件, 引线已在内部连接完成, 只引出它们的各个笔划, 公共电极。led 数码管常用段数一般为 7 段, 如上图中的 abcdefg, 有的还会有一个小数点, 如图中的 h。



数码管要正常显示, 就要用驱动电路来驱动数码管的各个段码, 从而显示出我们要的数字。按发光二极管单元连接方式可分为共阳极数码管和共阴极数码管。共阳数码管是指将所有发光二极管的阳极接到一起形成公共阳极(COM)的数码管, 共阳数码管在应用时应将公共极 COM 接到+5V, 当某一字段发光二极管的阴极为低电平时, 相应字段就点亮, 当某一字段的阴极为高电平时, 相应字段就不亮。共阴数码管是指将所有发光二极管的阴极接到一起形成公共阴极(COM)的数码管, 共阴数码管在应用时应将公共极 COM 接到地线 GND 上, 当某一字段发光二极管的阳极为高电平时, 相应字段就点亮, 当某一字段的阳极为低电平时, 相应字段就不亮。

下表列出了要显示的数字, 以及对应的 abcdefg 的值。

显示数字	共阳 abcdefg 2 进制	共阳 abcdefg g 16 进制	共阴 abcdefg 2 进制	共阴 abcdefg 16 进制
0	7'b0000001	7'h01	7'b 1111110	7'h7e
1	7'b 1001111	7'h4f	7'b 0110000	7'h30
2	7'b 0010010	7'h12	7'b 1101101	7'h6d
3	7'b 0000110	7'h06	7'b 1111001	7'h79
4	7'b 1001100	7'h4c	7'b 0110011	7'h33
5	7'b 0100100	7'h24	7'b 1011011	7'h5b
6	7'b 0100000	7'h20	7'b 1011111	7'h3f
7	7'b 0001111	7'h0f	7'b 1110000	7'h70
8	7'b 0000000	7'h00	7'b 1111111	7'h7f
9	7'b 0000100	7'h04	7'b 1111011	7'h7b

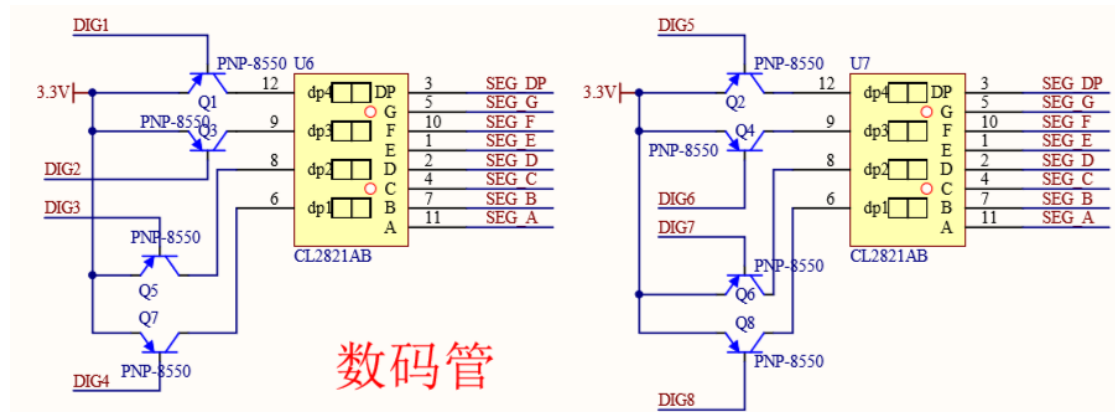
例如，共阳数码管中，*abcdefg* 的值分别是 1001111 时，也就是 *b* 和 *c* 字段亮，其他字段不亮，这时就显示了数字“1”。

如果要显示多个数码管，根据数码管的驱动方式的不同，可以分为静态式和动态式两类。

静态驱动也称直流驱动。静态驱动是指每个数码管的每一个段码都由一个单片机的 I/O 端口进行驱动，或者使用如 BCD 码二十进制译码器译码进行驱动。静态驱动的优点是编程简单，显示亮度高，缺点是占用 I/O 端口多，如驱动 5 个数码管静态显示则需要  $5 \times 8 = 40$  根 I/O 端口来驱动，要知道一个 89S51 单片机可用的 I/O 端口才 32 个，实际应用时必须增加译码驱动器进行驱动，增加了硬件电路的复杂性。

数码管动态显示接口是应用最为广泛的一种显示方式之一，动态驱动是将所有数码管的 8 个显示笔划“*a,b,c,d,e,f,g,dp*”的同名端连在一起，另外为每个数码管的公共极 COM 增加位选通控制电路，位选通由各自独立的 I/O 线控制，当要输出字形码时，所有数码管都接收到相同的字形码，但究竟是哪个数码管会显示出字形，取决于单片机对位选通 COM 端电路的控制，所以我们只要将需要显示的数码管的选通控制打开，该位就显示出字形，没有选通的数码管就不会亮。通过分时轮流控制各个数码管的 COM 端，就使各个数码管轮流受控显示，这就是动态驱动。在轮流显示过程中，每位数码管的点亮时间为 1~2ms，由于人的视觉暂留现象及发光二极管的余辉效应，尽管实际上各位数码管并非同时点亮，但只要扫描的速度足够快，给人的印象就是一组稳定的显示数据，不会有闪烁感，动态显示的效果和静态显示是一样的，能够节省大量的 I/O 端口，而且功耗更低。

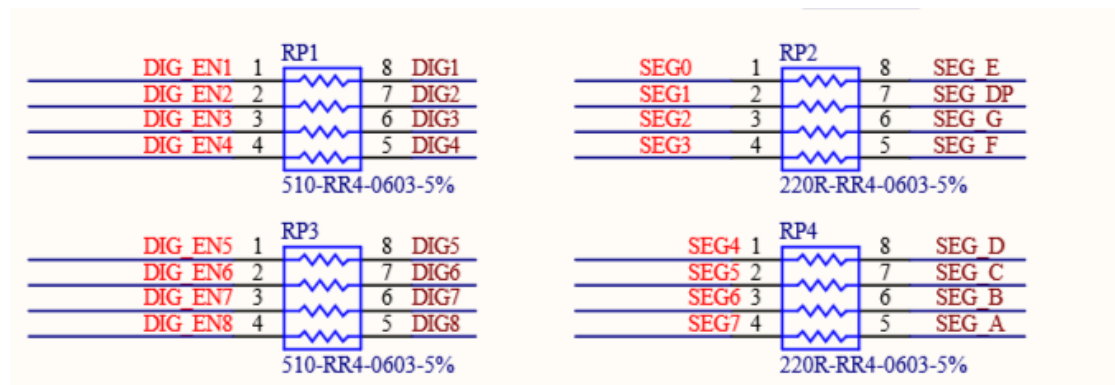
明德扬开发板上一共有 2 组 4 位的共阳数码管，也就是说一共有 8 个共阳数码管。数码管的配置电路如下。



图中的 SEG\_A, SEG\_B~SEG\_DP, 是段选信号，这些信号都是 8 个数码管共用的。

DIG1~DIG8 是位选信号，分别对应 8 个数码管。对应的位选信号为 0，就表示将段选信号的值赋给该数码管。例如 DIG3 为 0，表示将段选信号 SEG\_A~SEG\_DP 的值赋给数码管 3。

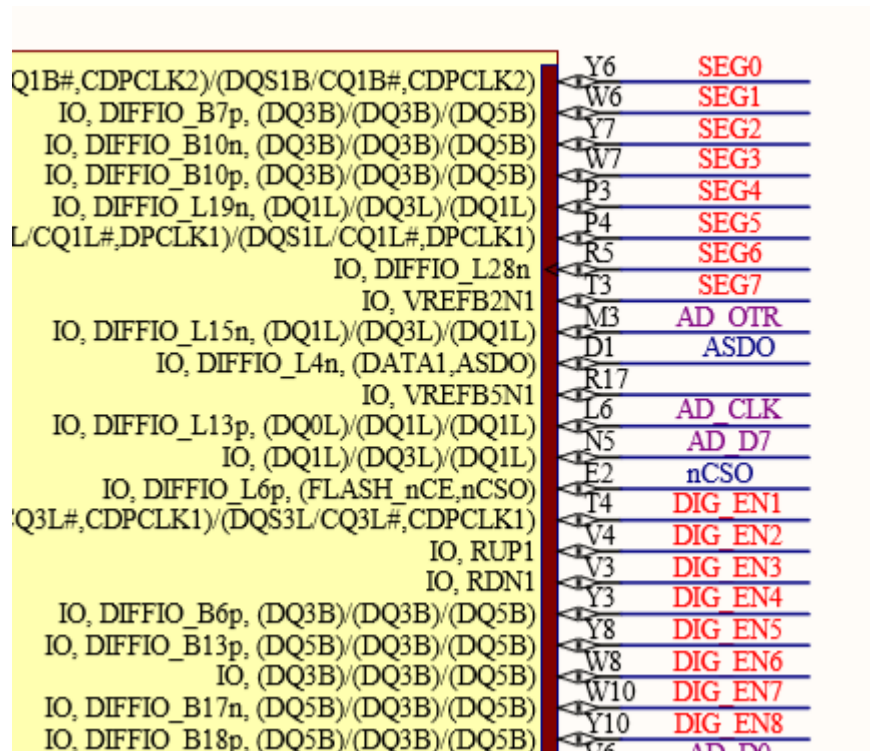
SEG\_A~SEG\_DP, DIG1~DIG8, 都是连接到电阻，如下图。



由此可见，SEG\_A~SEG\_DP 是由 SEG0~SEG7 产生的，DIG1~DIG8 是由 DIG\_EN1~DIG\_E

N8 产生的。

而 SEG0~SEG7 和 DIG\_EN1~DIG\_EN8 直接连到 FPGA 的 IO 上。



这些信号与 FPGA 管脚的对应关系如下表。

信号线	信号线	FPGA 管脚
SEG_E	SEG0	Y6
SEG_DP	SEG1	W6
SEG_G	SEG2	Y7
SEG_F	SEG3	W7
SEG_D	SEG4	P3
SEG_C	SEG5	P4
SEG_B	SEG6	R5
SEG_A	SEG7	T3
DIG1	DIG_EN1	T4
DIG2	DIG_EN2	V4
DIG3	DIG_EN3	V3
DIG4	DIG_EN4	Y3
DIG5	DIG_EN5	Y8
DIG6	DIG_EN6	W8
DIG7	DIG_EN7	W10
DIG8	DIG_EN8	Y10

也就是说，FPGA 通过控制上面中的管脚，就控制了数码管的显示。

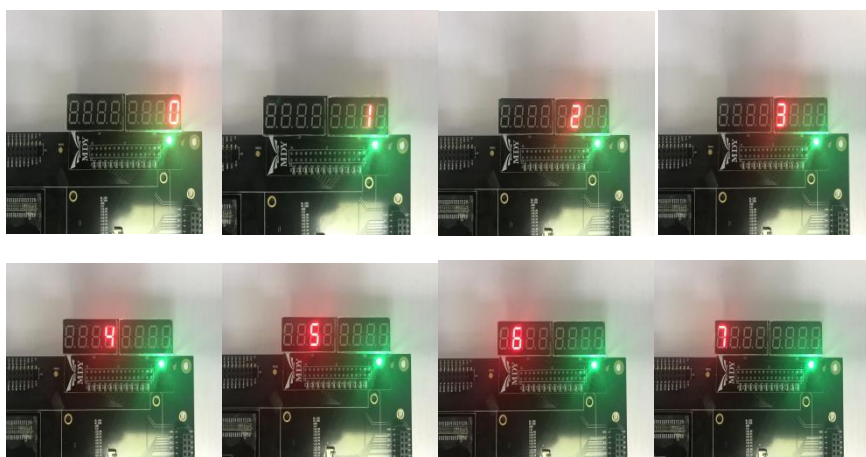
## 二、设计目标

开发板或者模块是有 8 位数数码管，本次设计需要使用 8 个数码管，实现数码管显示功能，具体要求如下：

复位后，数码管 0 显示数字 0；1 秒后，轮到数码管 1 显示数字 1；1 秒后，轮到数码管 2 显示数字 2；以此类推，每隔 1 秒变化，最后是数码管 7 显示数字 7。然后再次循环。

上板效果图如下图所示。

上板的演示效果，请登陆网址查看：[www.mdy-edu.com/xxxx](http://www.mdy-edu.com/xxxx)。



## 三、模块设计

我们要实现的功能，概括起来就是控制 8 个数码管，让数码管显示不同的数字。要控制 8 个数码管，就需要控制位选信号，即 FPGA 要输出一个 8 位的位选信号，设为 `seg_sel`，其中 `seg_sel[0]` 对应数码管 0，`seg_sel[1]` 对应数码管 1，以此类推，`seg_sel[7]` 对应数码管 7。

要显示不同的数字，就需要控制段选信号，不需要用到 DP，一共有 7 根线，即 FPGA 要输出一个 7 位的段选信号，设为 `seg_ment`，`seg_ment[6]~seg_ment[0]` 分别对应数码管的 abcdefg（注意对应顺序）。

我们还需要时钟信号和复位信号来进行工程控制。

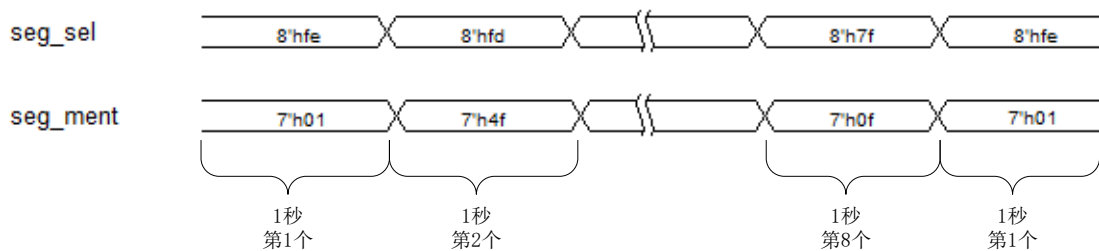
综上所述，我们这个工程需要 4 个信号，时钟 `clk`，复位 `rst_n`，输出的位选信号 `seg_sel` 和输出的段选信号 `seg_ment`。其中，`seg_sel` 和 `seg_ment` 的对应关系下如下：

信号线	信号线	FPGA 管脚	内部信号
SEG_E	SEG0	Y6	<code>seg_ment[2]</code>
SEG_DP	SEG1	W6	未用到
SEG_G	SEG2	Y7	<code>seg_ment[0]</code>
SEG_F	SEG3	W7	<code>seg_ment[1]</code>
SEG_D	SEG4	P3	<code>seg_ment[3]</code>
SEG_C	SEG5	P4	<code>seg_ment[4]</code>

SEG_B	SEG6	R5	seg_ment[5]
SEG_A	SEG7	T3	seg_ment[6]
DIG1	DIG_EN1	T4	seg_sel[0]
DIG2	DIG_EN2	V4	seg_sel[1]
DIG3	DIG_EN3	V3	seg_sel[2]
DIG4	DIG_EN4	Y3	seg_sel[3]
DIG5	DIG_EN5	Y8	seg_sel[4]
DIG6	DIG_EN6	W8	seg_sel[5]
DIG7	DIG_EN7	W10	seg_sel[6]
DIG8	DIG_EN8	Y10	seg_sel[7]

我们先分析要实现的功能，数码管 0 显示数字 0，翻译成信号就是 `seg_sel` 的值为 `8'b1111_1110`，`seg_ment` 的值为 `7'b000_0001`。数码管 1 显示数字 1，也就是说 `seg_sel` 的值为 `8'b1111_1101`，`seg_ment` 的值为 `7'b100_1111`。以此类推，数码管 7 显示数字 7，就是 `seg_sel` 的值为 `8'b0111_1111`，`seg_ment` 的值为 `7'b000_1111`。

再留意下，以上都是每隔 1 秒进行变化，并且是 8 个数码管轮流显示，那么波形示意图如下图所示。



上图就是 `seg_sel` 和 `seg_seg` 信号的变化波形图。在显示第 1 个时，`seg_sel=8'hfe`，`seg_ment=7'h01` 并持续 1 秒；在第 1 个时，`seg_sel=8'hfd`，`seg_ment=7'h4f` 并持续 1 秒；以此类推，第 8 个时，`seg_sel=8'h7f`，`seg_ment=7'h0f` 并持续 1 秒。然后又再次重复。

由波形图可知，我们需要 1 个计数器用来计算 1 秒的时间。本工程的工作时钟是 50MHz，即周期为 20ns，计数器计数到  $1\_000\_000\_000/20=50\_000\_000$  个，我们就能知道 1 秒时间到了。另外，由于该计数器是不停地计数，永远不停止的，可以认为加 1 条件一直有效，可写成：`assign add_cnt==1`。综上所述，该计数器的代码如下。

```

29 always @(posedge clk or negedge rst_n)begin
30     if(!rst_n)begin
31         cnt0 <= 0;
32     end
33     else if(add_cnt0)begin
34         if(end_cnt0)
35             cnt0 <= 0;
36         else
37             cnt0 <= cnt0 + 1;
38     end
39 end
40 assign add_cnt0 = 1;
41 assign end_cnt0 = add_cnt0 && cnt0==50_000_000-1 ;

```

再次观察波形图，我们发现第 1 个，第 2 个直到第 8 个，说明这还需要另外一个计数器来表示第几个。该计数器表示第几个，自然是完成 1 秒就加 1，因为加 1 条件可为 end\_cnt0。该计数器一共要数 8 次。所以代码为：

```

45 always @(posedge clk or negedge rst_n)begin
46     if(!rst_n)begin
47         cnt1 <= 0;
48     end
49     else if(add_cnt1)begin
50         if(end_cnt1)
51             cnt1 <= 0;
52         else
53             cnt1 <= cnt1 + 1;
54     end
55 end
56 assign add_cnt1 = end_cnt0;
57 assign end_cnt1 = add_cnt1 && cnt1==8-1 ;

```

有了两个计数器，我们来思考输出信号 seg\_sel 的变化。概括起来，在第 1 次的时候输出值为 8'hfe；在第 2 次的时候输出值为 8'hfd；以此类推，在第 8 次的时候输出值为 8'h7f。我们用信号 cnt1 来代替第几次，也就是：当 cnt1==0 的时候，输出值为 8'hfe；在 cnt1==1 的时候输出值为 8'hfd；以此类推，在 cnt1==7 的时候输出值为 8'h7f。再进一步翻译成代码，就变成如下：

```

96 always @(posedge clk or negedge rst_n)begin
97     if(rst_n==1'b0)begin
98         seg_sel <= 8'hfe;
99     end
100    else if(cnt1==0)begin
101        seg_sel <= 8'hfe;
102    end
103    else if(cnt1==1)begin
104        seg_sel <= 8'hfd;
105    end
106    else if(cnt1==2)begin
107        seg_sel <= 8'hfb;
108    end
109    else if(cnt1==3)begin
110        seg_sel <= 8'hf7;
111    end
112    else if(cnt1==4)begin
113        seg_sel <= 8'hfe;
114    end
115    else if(cnt1==5)begin
116        seg_sel <= 8'hdf;
117    end
118    else if(cnt1==6)begin
119        seg_sel <= 8'hbf;
120    end
121    else if(cnt1==7)begin
122        seg_sel <= 8'h7f;
123    end
124 end

```

读者有没有发现，上面代码基本上和文字描述是一模一样的，这进一步展现了 verilog 是“硬件描述语言”。上面的代码是能正确实现 seg\_sel 功能的，从实现角度和资源角度来说，都挺好。但代码进一步概括，可以化简如下：

```

126 always @(posedge clk or negedge rst_n)begin
127     if(rst_n==1'b0)begin
128         seg_sel <= 8'hfe;
129     end
130     else begin
131         seg_sel <= ~(8'b1<<cnt1);
132     end
133 end

```

对上面代码解释一下，第 131 行是指先将 8'b1 向左移位，再取反后的值，赋给 seg\_sel。假设此时 cnt1 等于 0，那么 8'b1<<0 的结果是 8'b0000\_0001，取反的值为 8'hfe；假设 cnt1 等于 3，那么 8'b1<<3 的结果为 8'b000\_1000，取反后的结果为 8'b1111\_0111，即 8'hf7。与第一种写法的结果

都是相同的。

我们来思考输出信号 `seg_ment` 的变化。概括起来，在第 1 次的时候输出值为 `7'h01`；在第 2 次的时候输出值为 `7'h4f`；以此类推，在第 8 次的时候输出值为 `7'h0f`。我们用信号 `cnt1` 来代替第几次，也就是：当 `cnt1==0` 的时候，输出值为 `7'h01`；在 `cnt1==1` 的时候输出值为 `7'h4f`；以此类推，在 `cnt1==7` 的时候输出值为 `7'h0f`。再进一步翻译成代码，就变成如下：

```
138 always @(posedge clk or negedge rst_n)begin
139     if(rst_n==1'b0)begin
140         seg_ment <= 7'h01;
141     end
142     else if(cnt1==0)begin
143         seg_ment <= 7'h01;
144     end
145     else if(cnt1==1)begin
146         seg_ment <= 7'h4f;
147     end
148     else if(cnt1==2)begin
149         seg_ment <= 7'h12;
150     end
151     else if(cnt1==3)begin
152         seg_ment <= 7'h06;
153     end
154     else if(cnt1==4)begin
155         seg_ment <= 7'h4c;
156     end
157     else if(cnt1==5)begin
158         seg_ment <= 7'h24;
159     end
160     else if(cnt1==6)begin
161         seg_ment <= 7'h20;
162     end
163     else if(cnt1==7)begin
164         seg_ment <= 7'h0f;
165     end
166 end
```

上面的代码正确地实现了 `seg_ment` 的功能，对于本工程说已经完美。但我们分析一下，就知道上面代码实现了类似译码的功能，将数字设成数码管显示的值，代码里只对 `0~7` 进行译码。很自然的，我先做一个通用的译码模块，将 `0~9` 都进行译码，以后就方便调用了。例如改成下面代码。



```

138 always @(posedge clk or negedge rst_n)begin
139     if(rst_n==1'b0)begin
140         seg_ment <= 7'h01;
141     end
142     else if(data==0)begin
143         seg_ment <= 7'h01;
144     end
145     else if(data==1)begin
146         seg_ment <= 7'h4f;
147     end
148     else if(data==2)begin
149         seg_ment <= 7'h12;
150     end
151     else if(data==3)begin
152         seg_ment <= 7'h06;
153     end
154     else if(data==4)begin
155         seg_ment <= 7'h4c;
156     end
157     else if(data==5)begin
158         seg_ment <= 7'h24;
159     end
160     else if(data==6)begin
161         seg_ment <= 7'h20;
162     end
163     else if(data==7)begin
164         seg_ment <= 7'h0f;
165     end
166     else if(data==8)begin
167         seg_ment <= 7'h00;
168     end
169     else if(data==9)begin
170         seg_ment <= 7'h04;
171     end
172 end

```

然后我们只要控制好 data 就能实现想要在数码管显示的数字，如下面代码。

```

174 assign data = cnt1;

```

当 cnt1=0，则数码管会显示 0。当 cnt1=1，则数码管会显示 1。

在代码的最后一行写下 endmodule

```

93 endmodule

```

至此，主体程序已经完成。接下来是将 module 补充完整。

将 module 的名称定义为 my\_seg。并且我们已经知道该模块有 4 个信号：clk、rst\_n、seg\_sel 和 seg\_ment，代码如下：

```

1 module my_seg(
2     clk      ,
3     rst_n   ,
4     seg_sel  ,
5     seg_ment
6 );
7

```

其中 clk、rst\_n 是 1 位的输入信号，seg\_sel 是 8 位的输出信号，seg\_ment 是 7 位的输出信号，根据此，补充输入输出端口定义。代码如下：

```

8 input      clk      ;
9 input      rst_n   ;
10 output[7:0] seg_sel ;
11 output[6:0] seg_ment;

```

接下来定义信号类型。

cnt0 是用 always 产生的信号，因此类型为 reg。cnt0 计数的最大值为 50\_000\_000，需要用 26 根线表示，即位宽是 26 位。add\_cnt0 和 end\_cnt0 都是用 assign 方式设计的，因此类型为 wire。并且其值是 0 或者 1，1 个线表示即可。因此代码如下：

```

15 reg [25:0] cnt0 ;
16 wire      add_cnt0 ;
17 wire      end_cnt0 ;
18

```

cnt1 是用 always 产生的信号，因此类型为 reg。cnt1 计数的最大值为 7，需要用 3 根线表示，即位宽是 3 位。add\_cnt1 和 end\_cnt1 都是用 assign 方式设计的，因此类型为 wire。并且其值是 0 或者 1，1 根线表示即可。因此代码如下：

```

19 reg [ 2:0] cnt1 ;
20 wire      add_cnt1 ;
21 wire      end_cnt1 ;
22

```

seg\_sel 是用 always 方式设计的，因此类型为 reg，其一共有 8 根线，即位宽为 8。因此代码如下：

```

13 reg [ 7:0] seg_sel ;

```

seg\_ment 是用 always 方式设计的，因此类型为 reg，其一共有 7 根线，即位宽为 7。因此代码如下：

```

15 reg [ 6:0] seg_ment ;

```

如果做了译码电路，即用到了 data 这个信号。那么 data 是用 assign 设计的，所以类型为 wire，

其最大值为 9，所以需要 4 位位宽。代码如下：

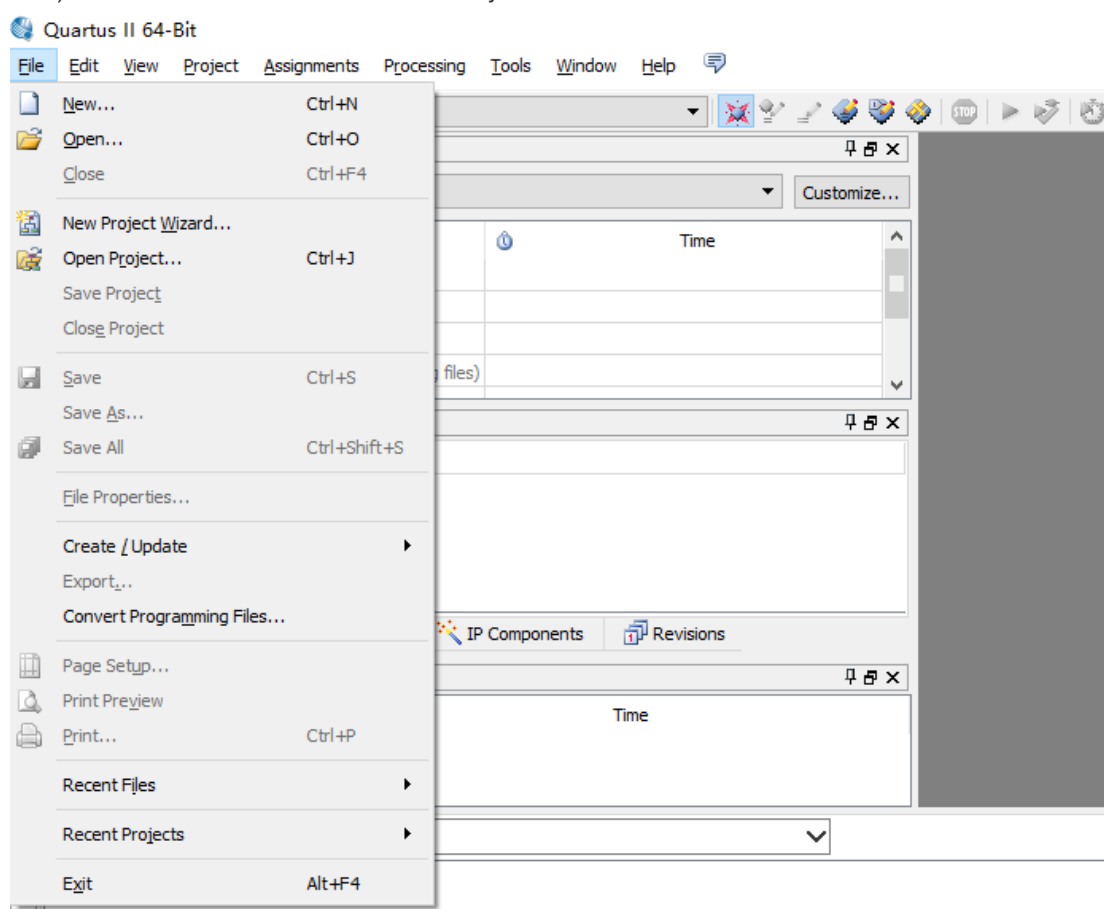
```
wire[3:0] data ;
```

至此，整个代码的设计工作已经完成。下一步是新建工程和上板查看现象。

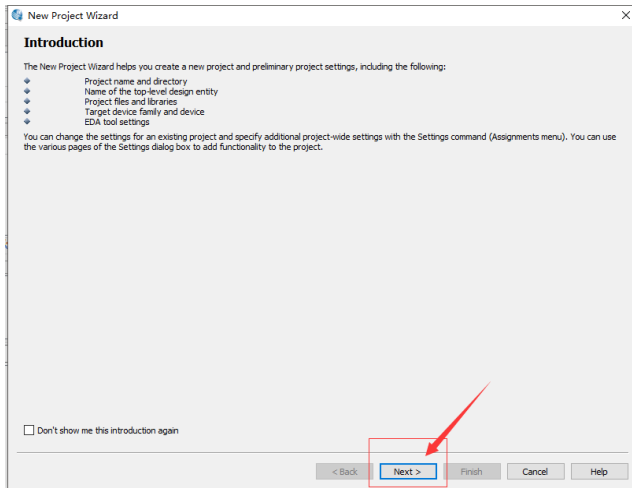
## 四、综合工程和上板

### 新建工程

(1.)点击 File 在 File 菜单中选择 New Project Wizard....




2.弹出 Introduction 界面选择 next



(2.)工程文件夹，工程名，顶层模块名设置界面

点击 next 之后进入此界面

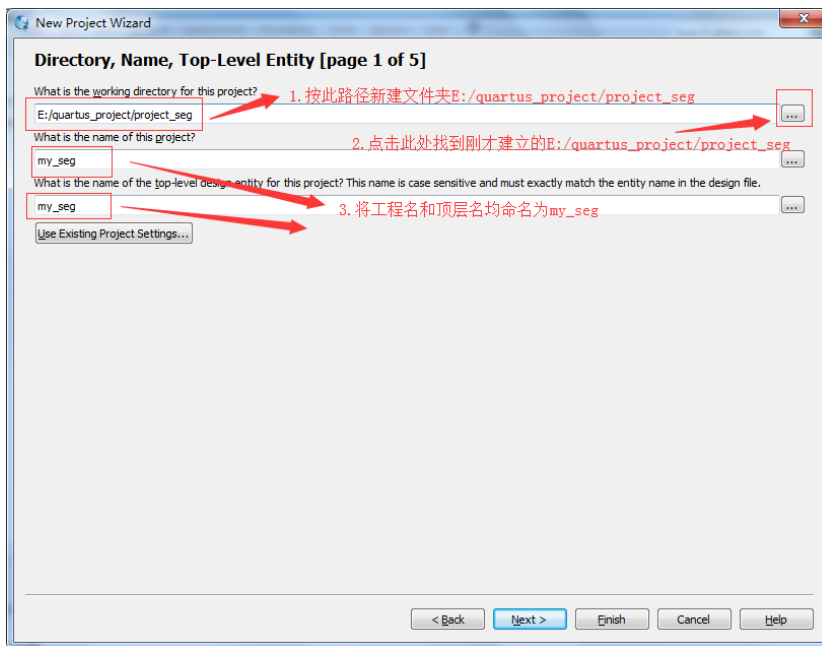
按如下路径建立文件夹 E: /quartus\_project/project\_seg

点击“选择工程文件夹 (what is the working directory for this project?)”后面的  键 找到之前建立的文件夹 E: /quartus\_project/project\_seg

将工程名命名栏 (what is the name of this project) 中输入 “my\_seg”

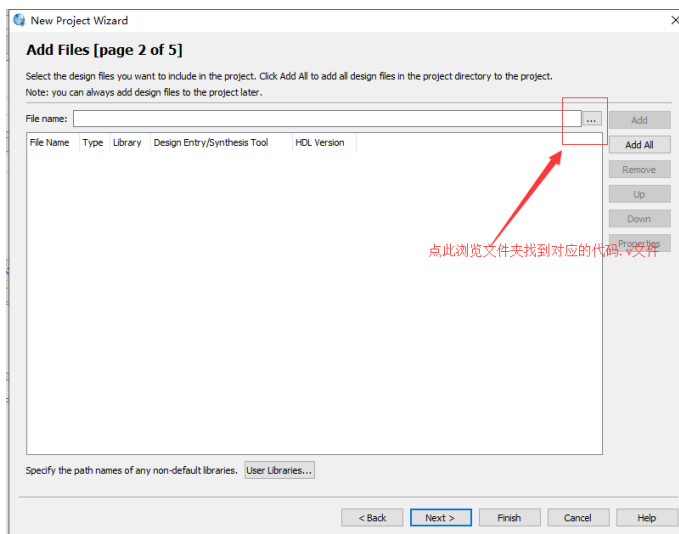
在顶层模块命名栏 (what is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file) 中输入 “my\_seg”

5.命名完毕后点击 next

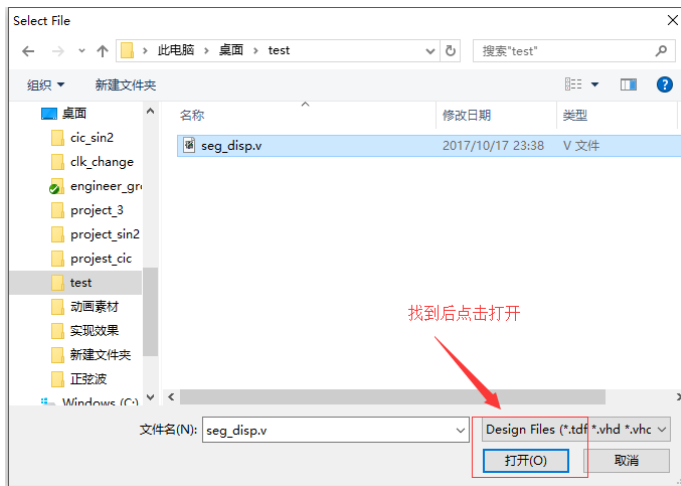


### (3.)文件添加界面

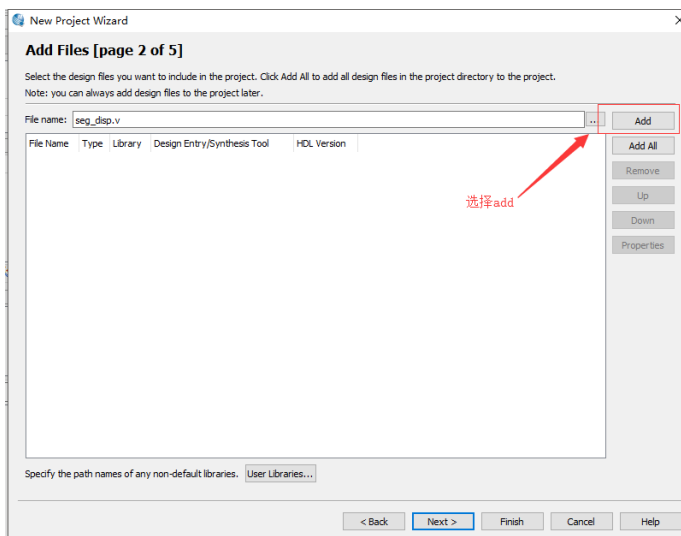
1.从上一界面进入此界面之后点击  浏览文件夹



2.找到我们之前写的.v 文件 之后选中它并点击打开

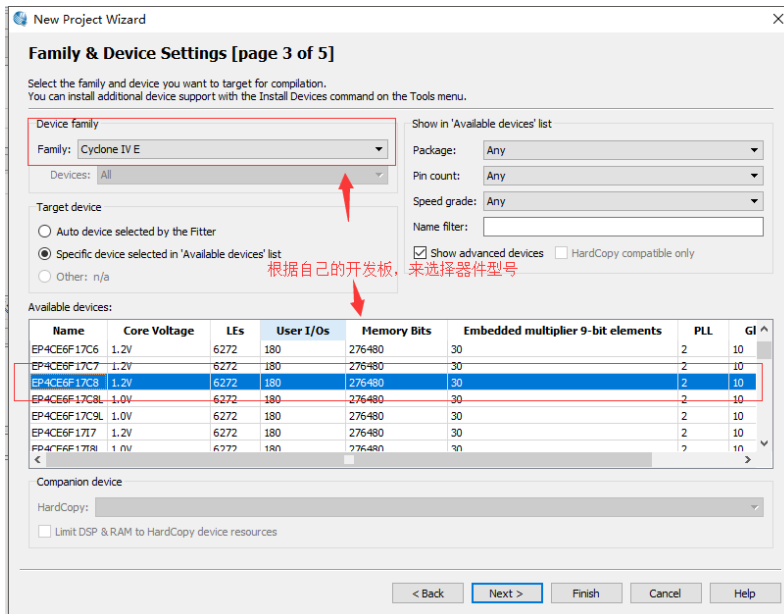


之后点击 add 添加此.v 文件



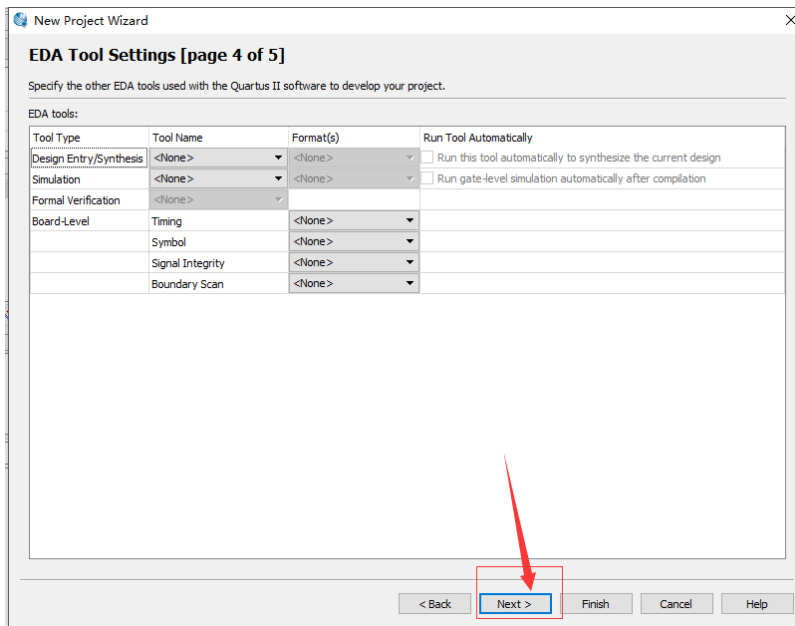
#### (4.) 器件选择界面

1. 在 Device family 这一项之中选择 Cyclone IV E
2. 在下部的 Available device 选择 EP4CE6F17C8 这一项  
完成后点击 next



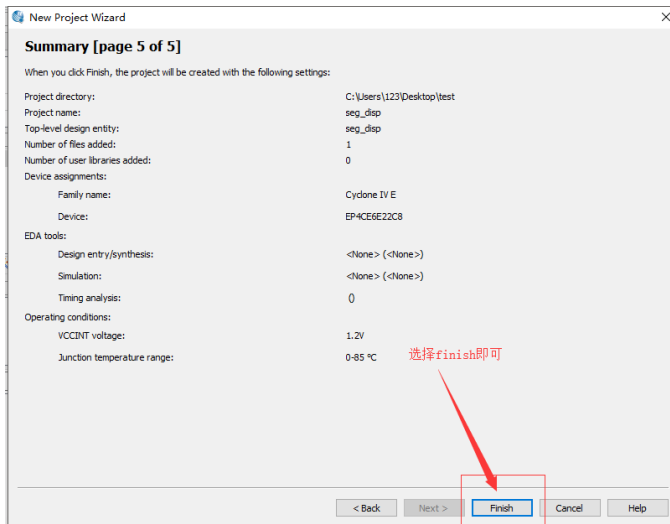
EDA 工具界面（采用默认配置即可）

点击 next



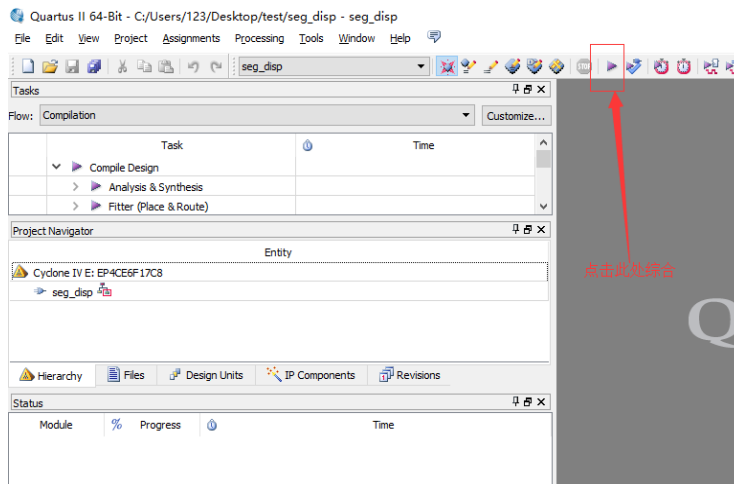
总结界面

直接点击 next 即可

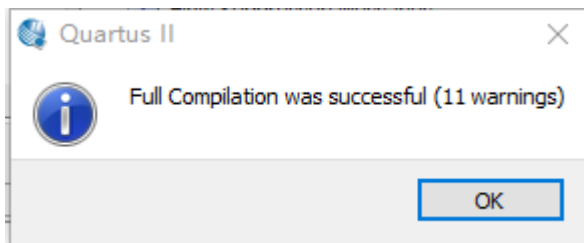


## 综合

1. 点击  此键进行编译



之后等待编译成功 点击 ok

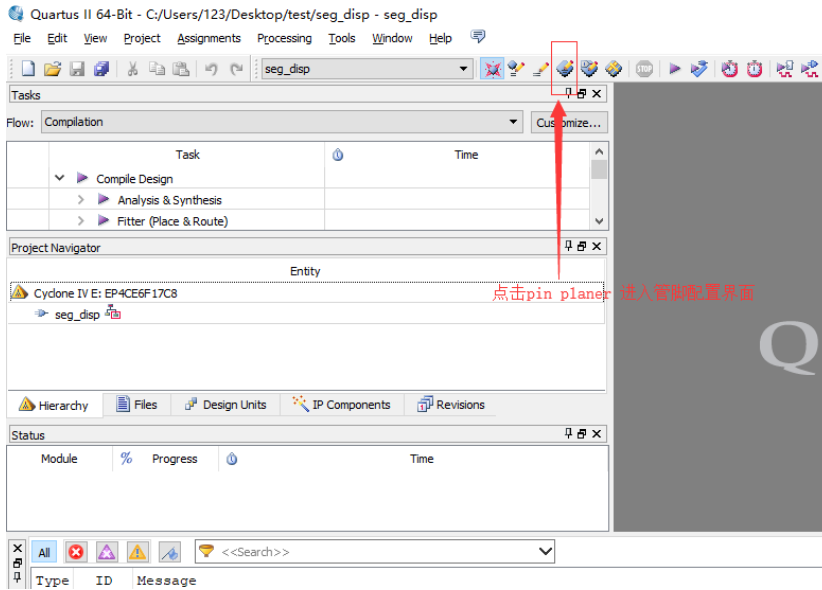


## 配置管脚

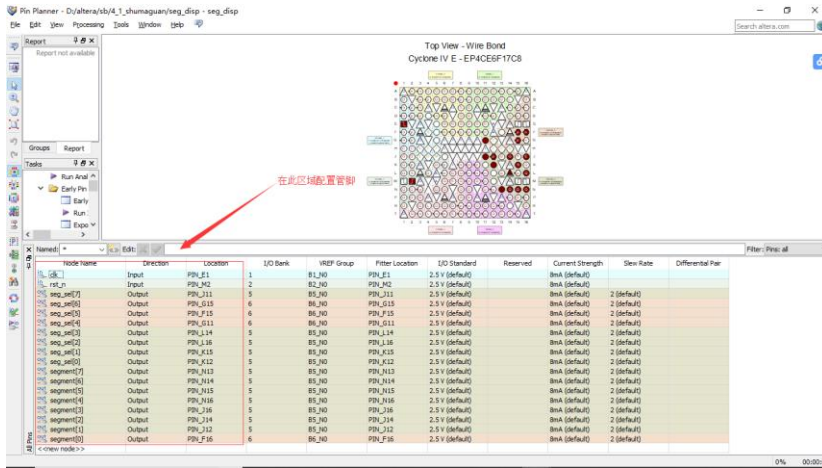
编译成功后配置管脚

点击图中位置 (pin planer 键) 进入管脚配置界面





在下图红框区域配置管脚




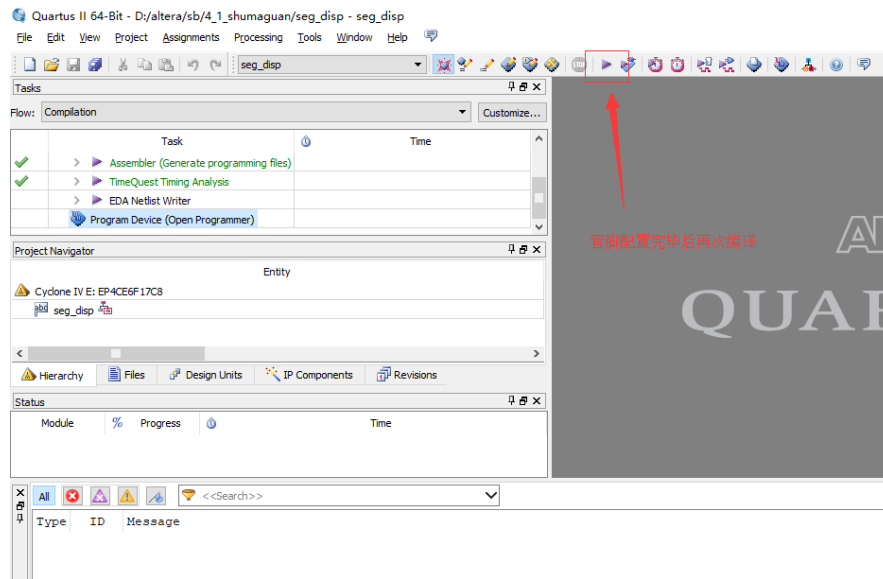
按下图内容进行管脚配置

信号线	信号线	FPGA 管脚	内部信号
SEG_E	SEG0	Y6	seg_ment[2]
SEG_DP	SEG1	W6	未用到
SEG_G	SEG2	Y7	seg_ment[0]
SEG_F	SEG3	W7	seg_ment[1]
SEG_D	SEG4	P3	seg_ment[3]
SEG_C	SEG5	P4	seg_ment[4]
SEG_B	SEG6	R5	seg_ment[5]
SEG_A	SEG7	T3	seg_ment[6]
DIG1	DIG_EN1	T4	seg_sel[0]
DIG2	DIG_EN2	V4	seg_sel[1]
DIG3	DIG_EN3	V3	seg_sel[2]
DIG4	DIG_EN4	Y3	seg_sel[3]
DIG5	DIG_EN5	Y8	seg_sel[4]
DIG6	DIG_EN6	W8	seg_sel[5]

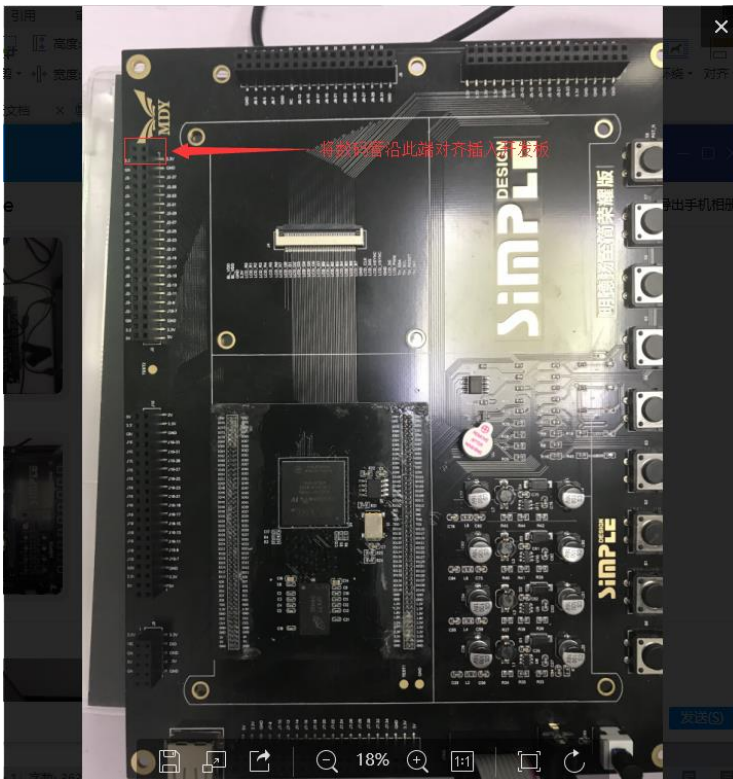
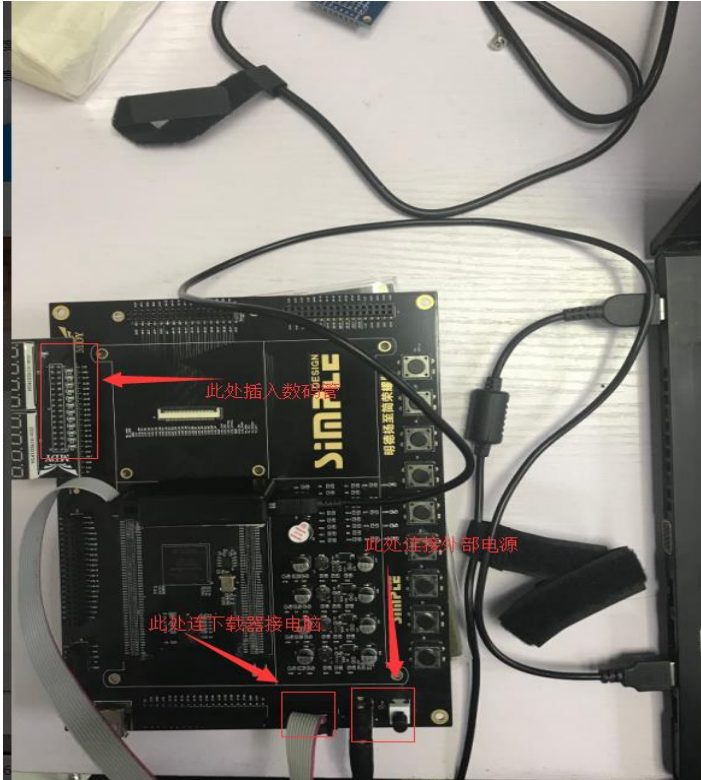
DIG7	DIG_EN7	W10	seg_sel[6]
DIG8	DIG_EN8	Y10	seg_sel[7]

## 布局布线

管脚配置完毕后再次点击  此键进行编译

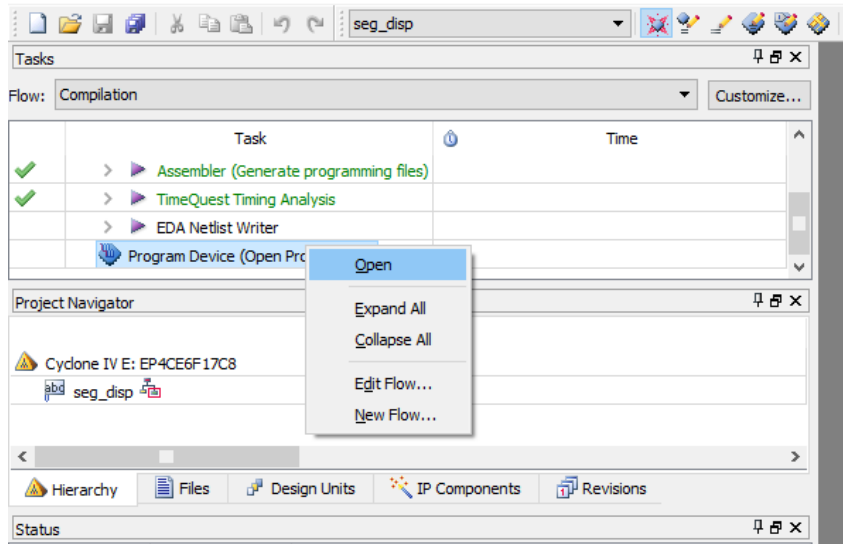


## 连接开发板

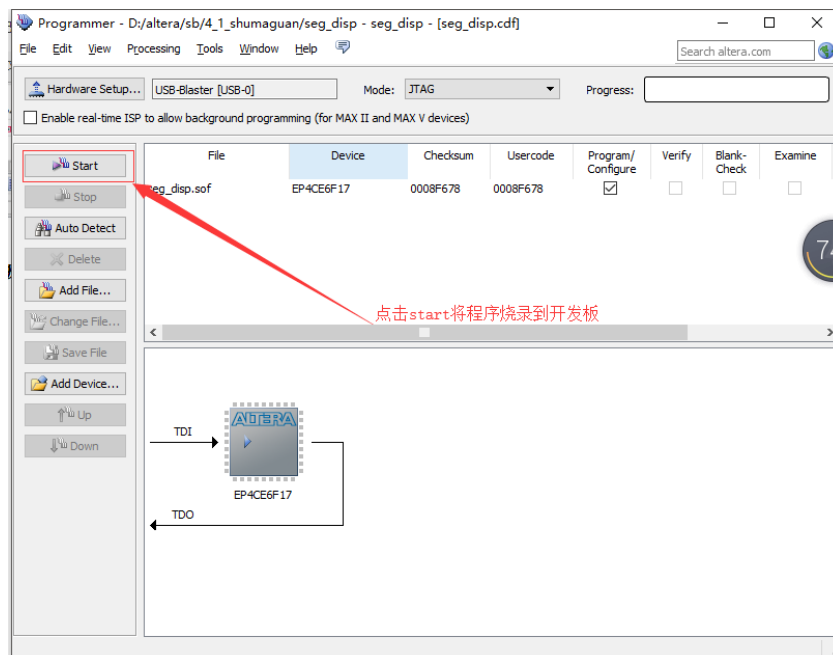


## 上板

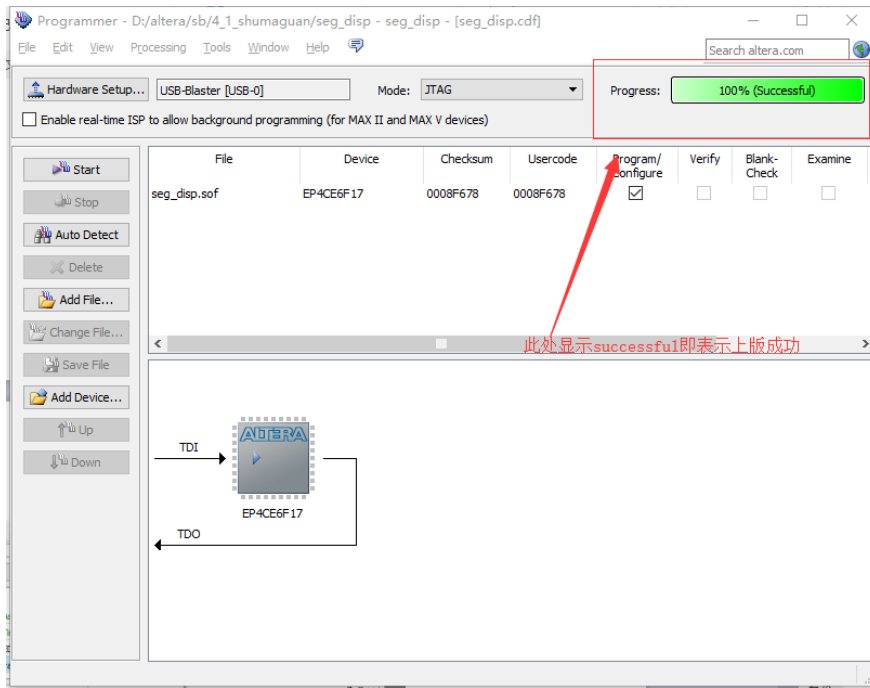
右键 Program Device 选择 Open 进入烧录界面



2. 点击 start 开始烧录程序



Progress 这一栏变为绿色显示百分之百 successful 即代表烧录成功



明德扬培训定位非常明确，坚持帮助学员实现两个目标：

具备独立项目开发能力+FPGA 就业。

我们认为，只要具备了独立开发项目能力，完成工作项目需求、导师项目要求、高薪就业这些就都不成问题。经过明德扬专业的培训，您将完全掌握到一种科学规范的 FPGA 设计方法，运用这套方法可以完成所有 FPGA 项目设计。完全具备 FPGA 工程师的能力，足以满足企业或实际项目的要求。

需要系统的学习 FPGA 请联系 Q1241003385 微信 18022857217

