

# vivado 使用教程

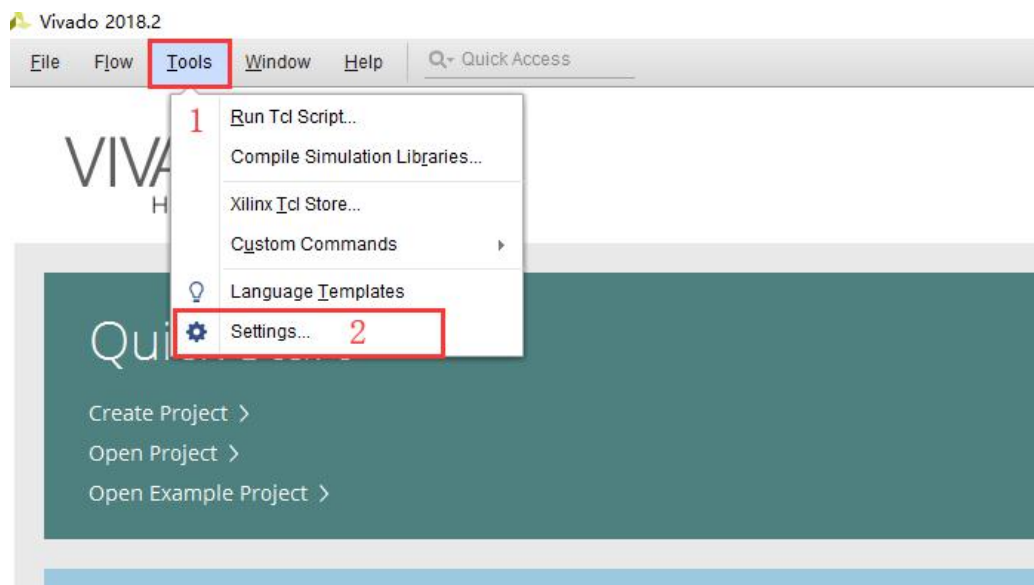
## 1. Vivado 关联第三方代码编辑器

Vivado 自带的编辑器并不是很好用。为此，很多学生都习惯于用第三方独立编辑器代替自带编辑器。支持 HDL 语言编程的第三方自带编辑器有很多，比如 Vim, Notepad++, Vscode, Sublime (收费), UltraEdit (收费) 等等。学生可以根据自己的需要选择合适的编辑器进行关联。此处以 Notepad++ 为例说明，其他选择的同学可以百度合适的方法，很容易百度到。具体方法如下：

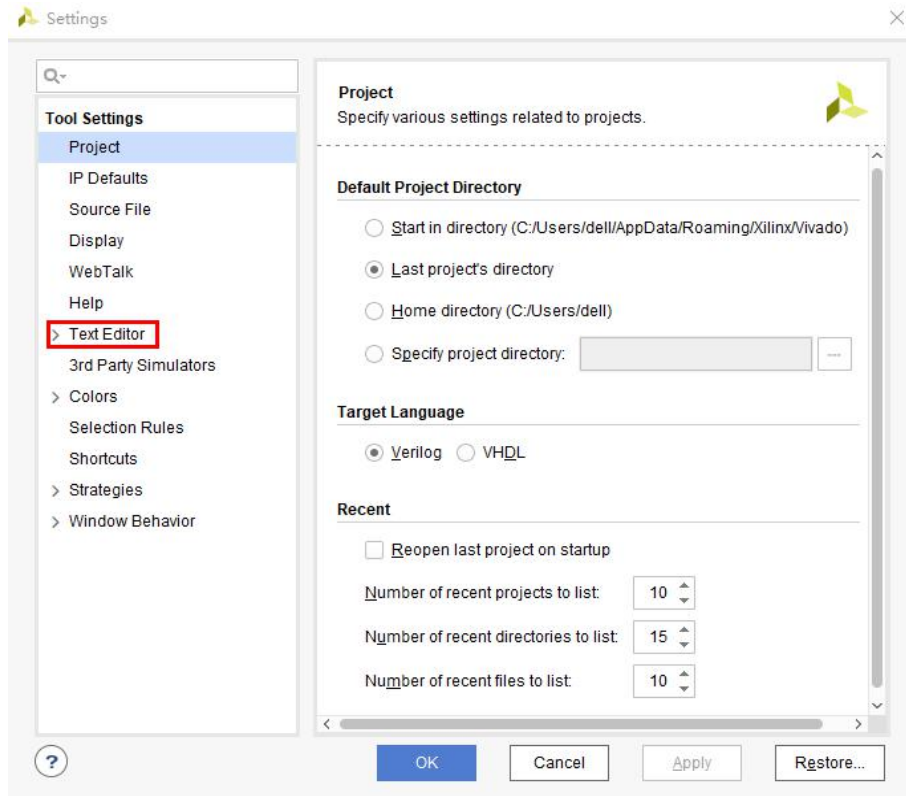
- 从桌面启动 vivado 软件。



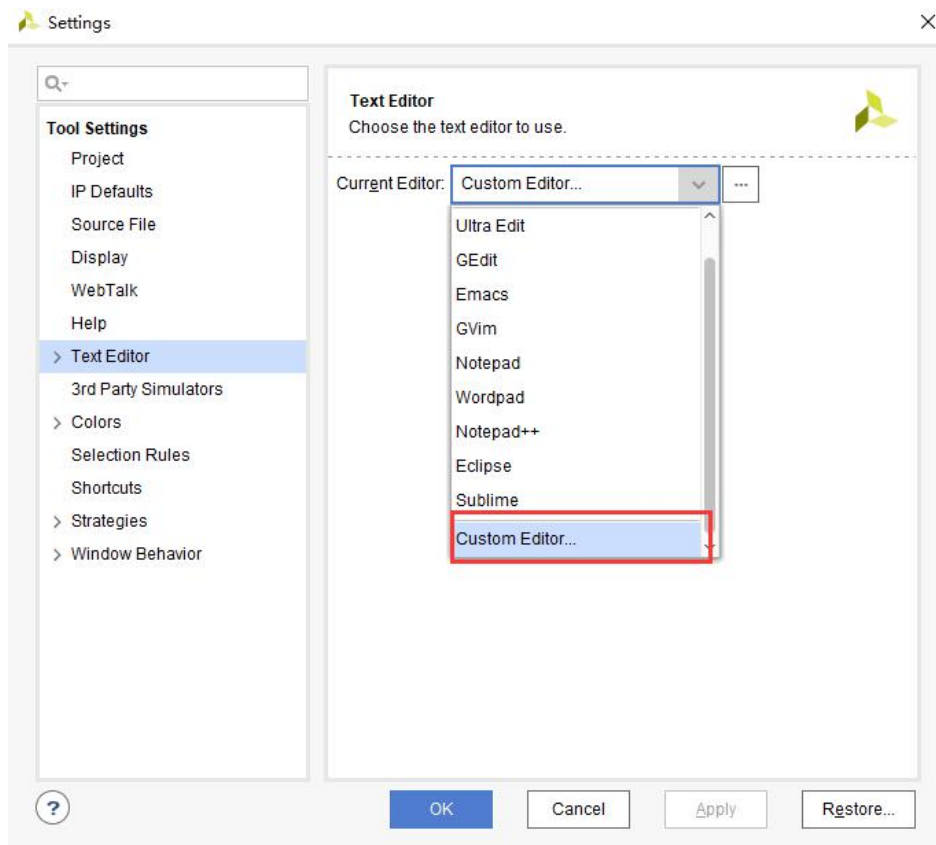
- 启动 Vivado 软件之后，按照下图中红色数字顺序在主界面的 Tools 菜单下点击 Settings 选项，进入 Settings 设置界面。



- 如下图红色框中，选择 Text Editor (编辑器修改选择) 选项。



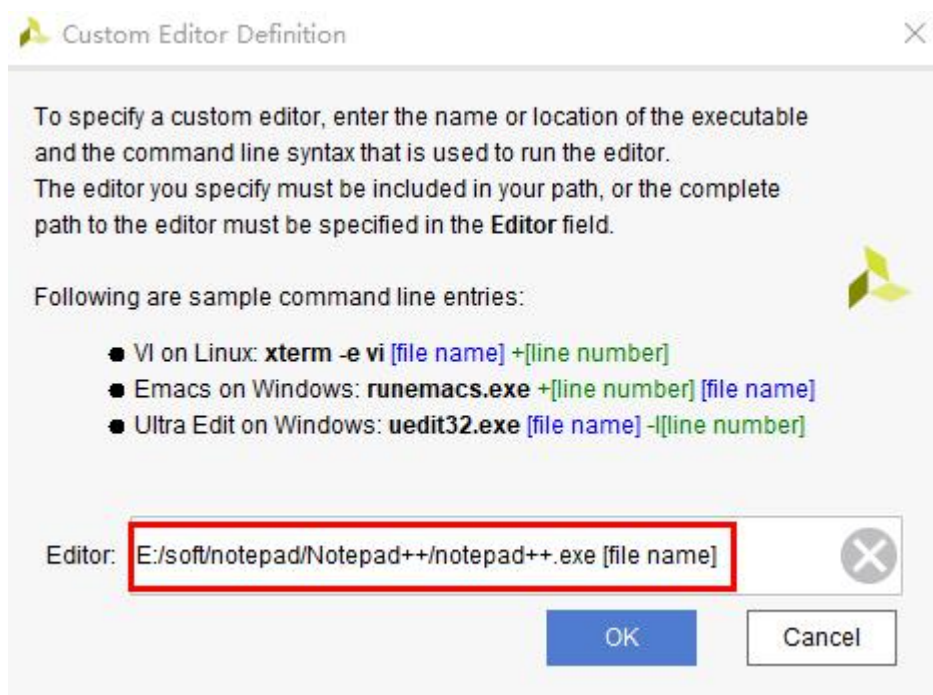
- 进入编辑器修改界面，在右边窗口 **Current Editor** 下拉框中找到 **Custom Editor** 后并选中该选项。



- 在 Custom Editor 右边会出现“...”。



- 点击“...”，弹出下面的界面，在红色框中选择自己电脑上 Notepad++ 的安装路径进行替换，格式保持不变。注意红框中的斜线为从右上到左下的。



- 设置好之后点击 OK，点击 Settings 设置界面的 OK。到此设关联第三方编辑器设置结束。在 Vivado 工程目录下，打开要编辑的文件时，会自动用 notepad++ 编辑器打开。

## 2. Vivado 关联第三方仿真软件

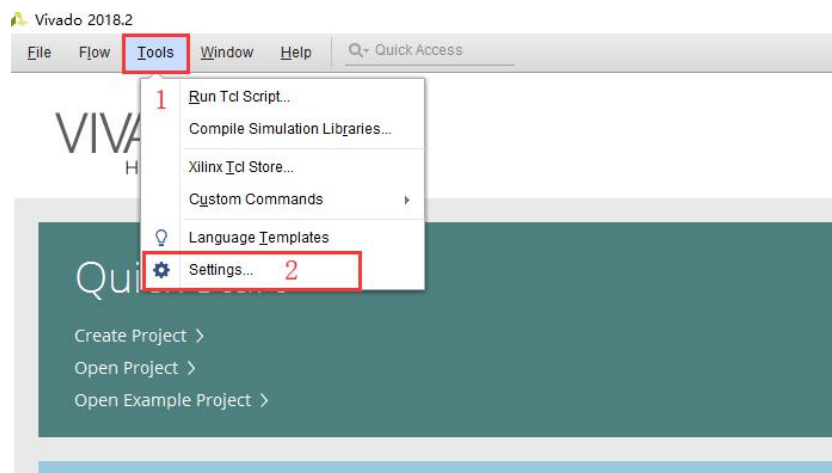
Vivado 自带自配套的仿真工具 Isim,并且该仿真软件对初学者容易上手，功能强大，完全可以满足大家工程设计的仿真需求。所以在 Vivado 平台下不建议大家使用第三方仿真工具。如果某些同学有特殊需求，必须使用第三方仿真工具则可以关联对应的仿真工具。

本接教程以第三方仿真工具 Modelsim 举例说明，如果有想使用其他仿真工具的同学请自行百度。

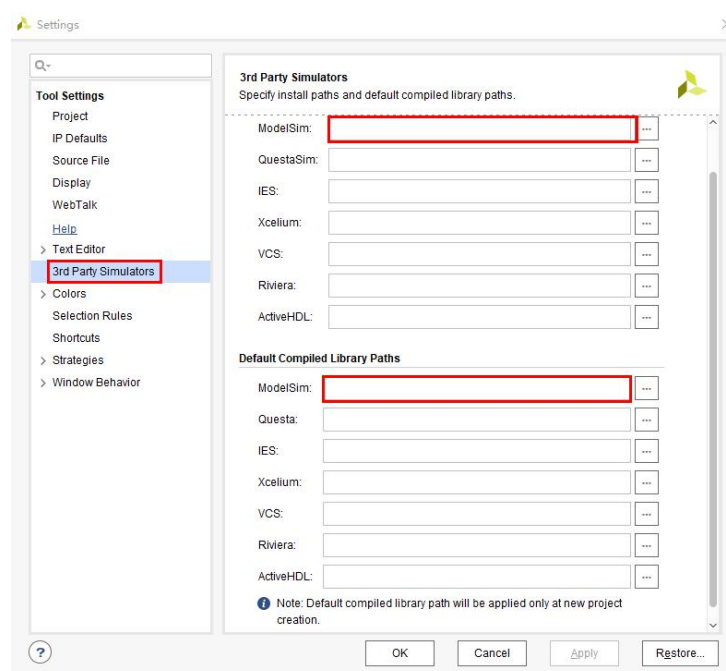
- 从桌面启动 vivado 软件。



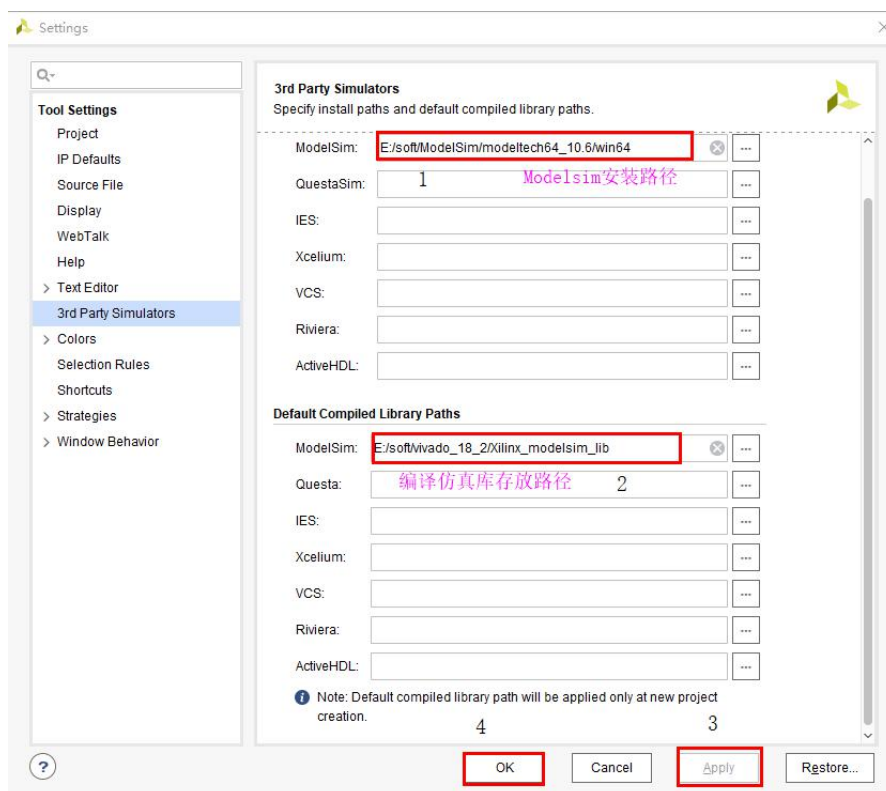
- 启动 Vivado 软件之后，按照下图中红色数字顺序在主界面的 Tools 菜单下点击 Settings 选项，进入 Settings 设置界面。



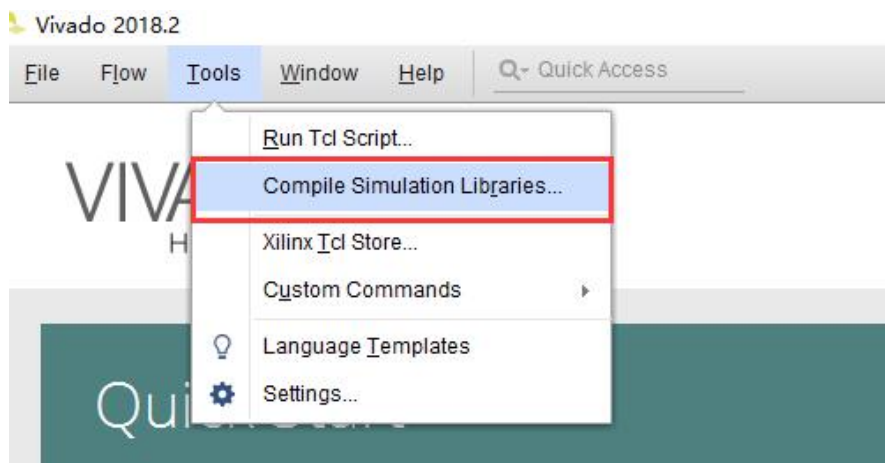
- 进入 Settings 界面，点击左侧的 3rd Party Simulators 选项，弹出仿真工具设置界面。大家可以看到有很多第三方仿真工具设置。选择红框中关于 ModelSim 的设置选项进行设置。



- 在右边 3rd Party Simulators 标签下，在 Install Paths 下的 Modelsim 通过点击右侧“...”，选择已经安装好 modelsim 的路径，在 Default Compiled Library Paths 下的 Modelsim 通过点击右侧“...”，选择设置一个英文目录文件夹存放第三方仿真工具编译好的库文件。此处，我在 Vivado 的安装路径下兴建了一个 Xilinx\_modelsim\_lib 文件夹，并将仿真库文件存放路径指定到这里，大家也可以根据自己的情况去进行设定（但必须是英文路径，不要带有空格），然后先点击 Apply,再次点击 OK。

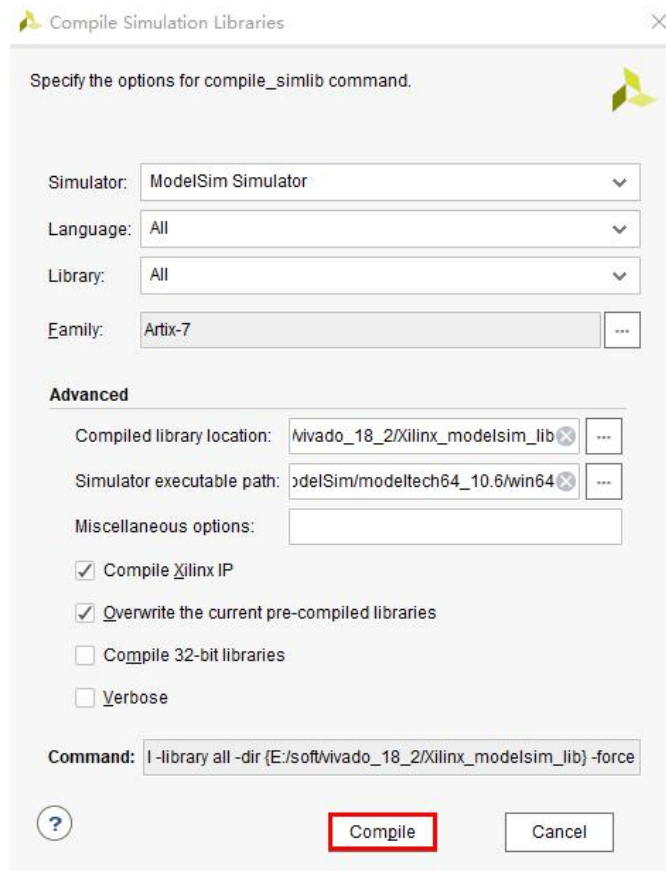


- 主界面的 Tools 菜单下点击 Compile Simulation Libraries...进行库文件编译。



- 6、进入 Compile Simulation Libraries 界面，需要进行如下一些设置，其他的全部保持默认即可。

- 我们使用的第三方仿真工具是 modelsim，在 Simulator 处选择 Modelsim Simulation;
- 器件就根据实际使用情况进行选择，我们使用的开发板是 Artix-7，可以只勾选这个就可以，这里选择的器件系列越多，Compile 生成库的时间越长，若需要选择全部的器件，建议可以选择在空闲时间进行编译。
- 设置完成之后，点击 Compile 进行编译。



- 编译开始后会出现如下进度条，此工程耗时较多，具体时间和个人电脑配置有关系。



到此，vivado 关联第三方仿真工具设置过程就结束了。

### 3. Vivado 软件的基本开发流程。

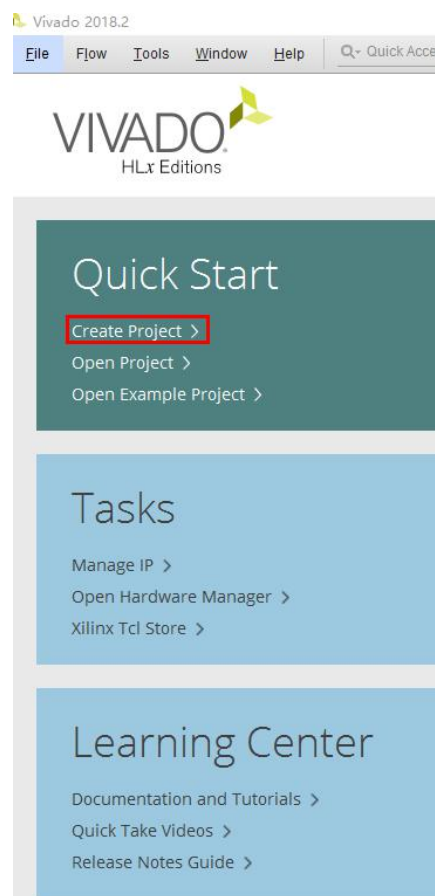
本次软件使用基本流程讲解以一个最简单的流水灯工程为设计实例说明。

- 从桌面启动 vivado 软件。

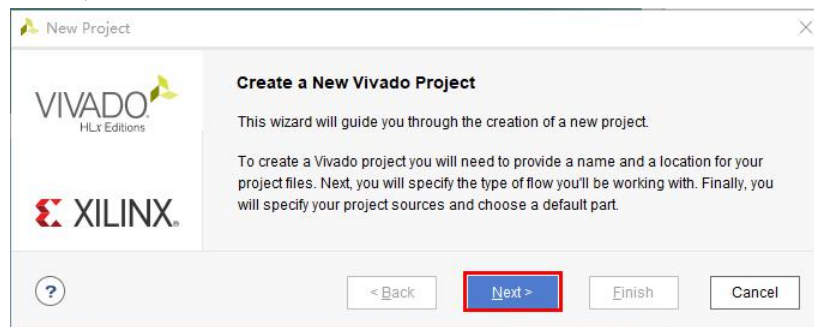


- 使用 Vivado 建立一个 FPGA 设计工程。

- File-> Project ->NEW 或者点击下面红框中的 Create Project 弹出新建工程界面

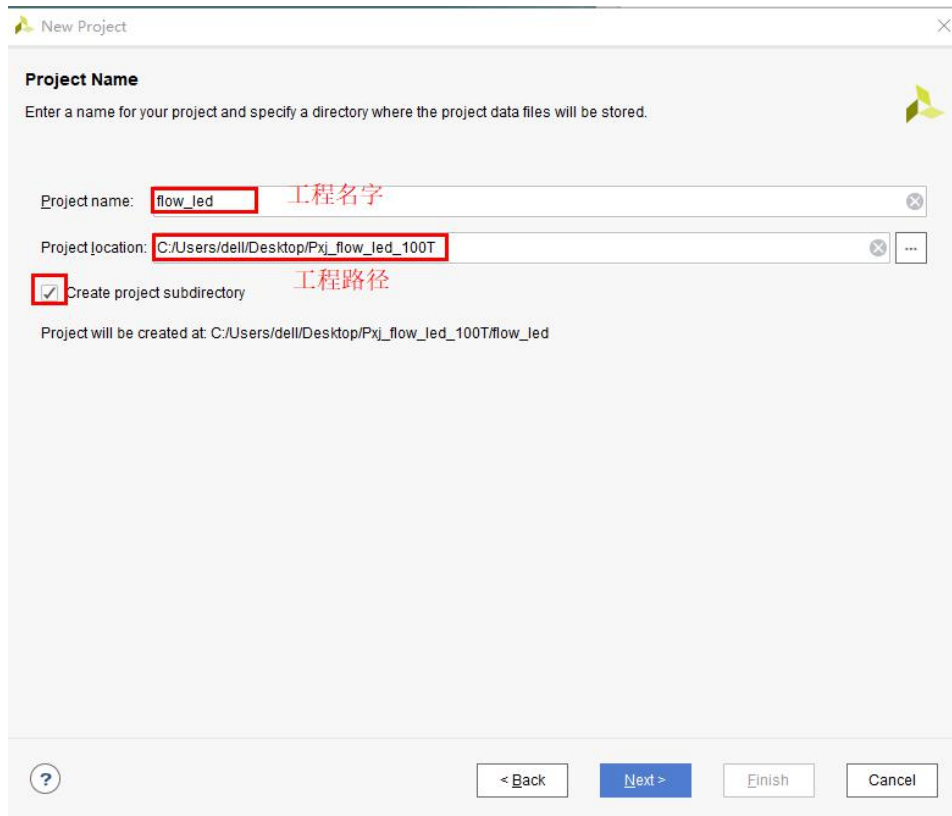


- 点击下面红框中确定





- 填写新建工程的名字和路径，如下图红色方框中,点击 Next。



The 'New Project' dialog box is shown. It has a title bar with a green icon and a close button. The main area is titled 'Project Name' and contains the instruction: 'Enter a name for your project and specify a directory where the project data files will be stored.' There are two input fields: 'Project name:' with the text 'flow\_led' and 'Project location:' with the text 'C:/Users/dell/Desktop/Pxj\_flow\_led\_100T'. Both fields are highlighted with red rectangles. Below these fields is a checkbox labeled 'Create project subdirectory' which is checked, also highlighted with a red rectangle. Below the checkbox, it says 'Project will be created at: C:/Users/dell/Desktop/Pxj\_flow\_led\_100T/flow\_led'. At the bottom, there are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'. The 'Next >' button is highlighted in blue.

Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

Project name: flow\_led

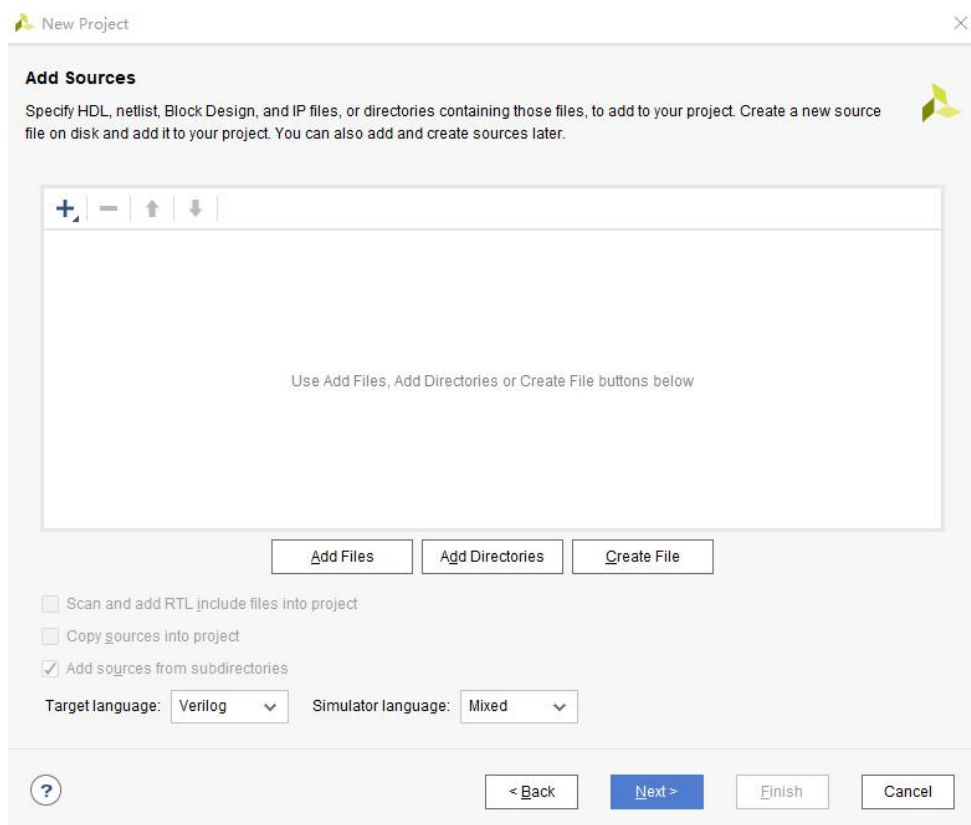
Project location: C:/Users/dell/Desktop/Pxj\_flow\_led\_100T

☒ Create project subdirectory

Project will be created at: C:/Users/dell/Desktop/Pxj\_flow\_led\_100T/flow\_led

< Back Next > Finish Cancel

- 在添加文件对话框中直接点击 Next（由于新建项目，没有可以添加的代码文件文件）



The 'Add Sources' dialog box is shown. It has a title bar with a green icon and a close button. The main area is titled 'Add Sources' and contains the instruction: 'Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.' Below this is a large empty rectangular area. At the bottom of this area, it says 'Use Add Files, Add Directories or Create File buttons below'. Below this area are three buttons: 'Add Files', 'Add Directories', and 'Create File'. Below these buttons are three checkboxes: 'Scan and add RTL include files into project', 'Copy sources into project', and 'Add sources from subdirectories'. The 'Add sources from subdirectories' checkbox is checked. Below the checkboxes are two dropdown menus: 'Target language:' with 'Verilog' selected and 'Simulator language:' with 'Mixed' selected. At the bottom, there are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'. The 'Next >' button is highlighted in blue.

Add Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.

Use Add Files, Add Directories or Create File buttons below

Add Files Add Directories Create File

☐ Scan and add RTL include files into project

☐ Copy sources into project

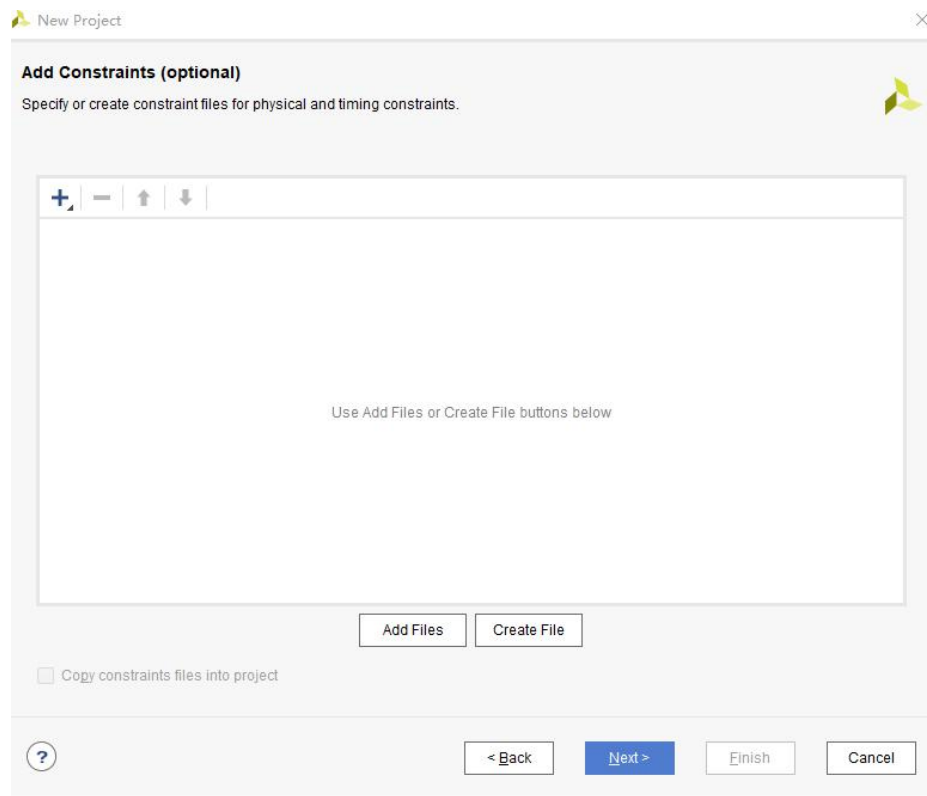
☒ Add sources from subdirectories

Target language: Verilog Simulator language: Mixed

< Back Next > Finish Cancel



- 在添加约束文件对话框中直接点击 Next（由于新建项目，没有可以添加的文件）



- 出现如下界面。在该界面选择开发板对应的器件。  
Family 选择 Artix-7,Package 选择 fgg676,Speed 选择 -2L，器件信号选择 xc7a100tfgg676-2L。之后，选择 Next。

New Project

### Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Parts | Boards

[Reset All Filters](#)

Category: All Package: All Temperature: All

Family: All Speed: All

Search: Q-

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gb Transceivers	GTPE2
xc7a50tcsq325-1L	325	150	32600	65200	75	0	120	4	4
xc7a50tfgg484-1L	484	250	32600	65200	75	0	120	4	4
xc7a50tfgg256-1L	256	170	32600	65200	75	0	120	0	0
xc7a50tcsq236-2L	236	106	32600	65200	75	0	120	2	2
xc7a50tcsq324-2L	324	210	32600	65200	75	0	120	0	0
xc7a50tcsq325-2L	325	150	32600	65200	75	0	120	4	4
xc7a50tfgg484-2L	484	250	32600	65200	75	0	120	4	4
xc7a50tfgg256-2L	256	170	32600	65200	75	0	120	0	0
xc7a75tcsq324-3	324	210	47200	94400	105	0	180	0	0

< Back Next > Finish Cancel

New Project

### Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Parts | Boards

[Reset All Filters](#)

Category: All Package: fgg676 Temperature: All Remaining

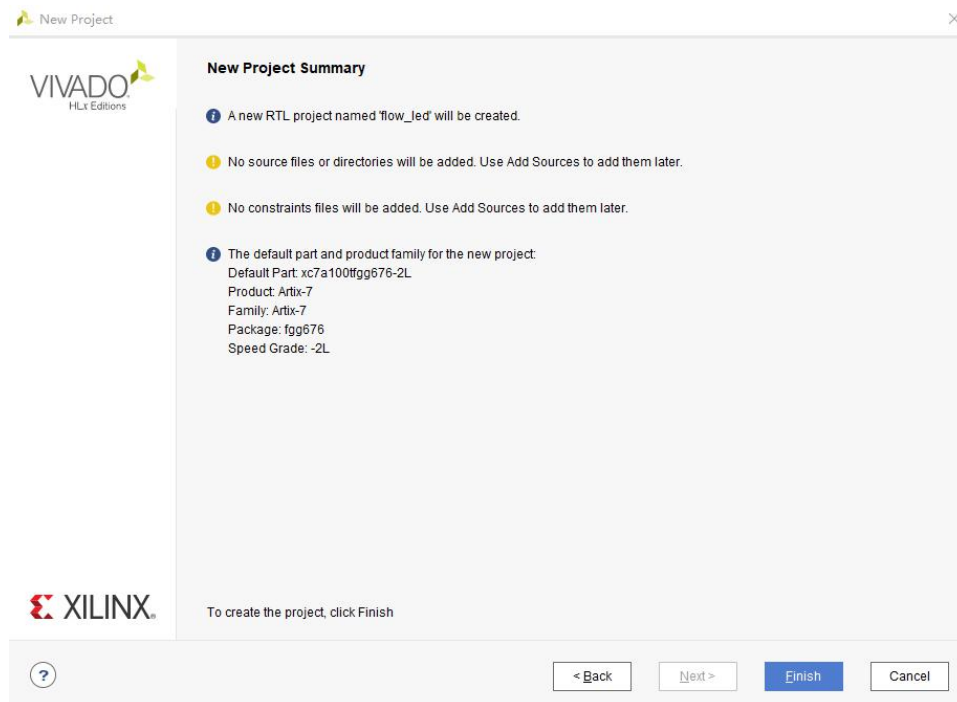
Family: Artix-7 Speed: -2L

Search: Q-

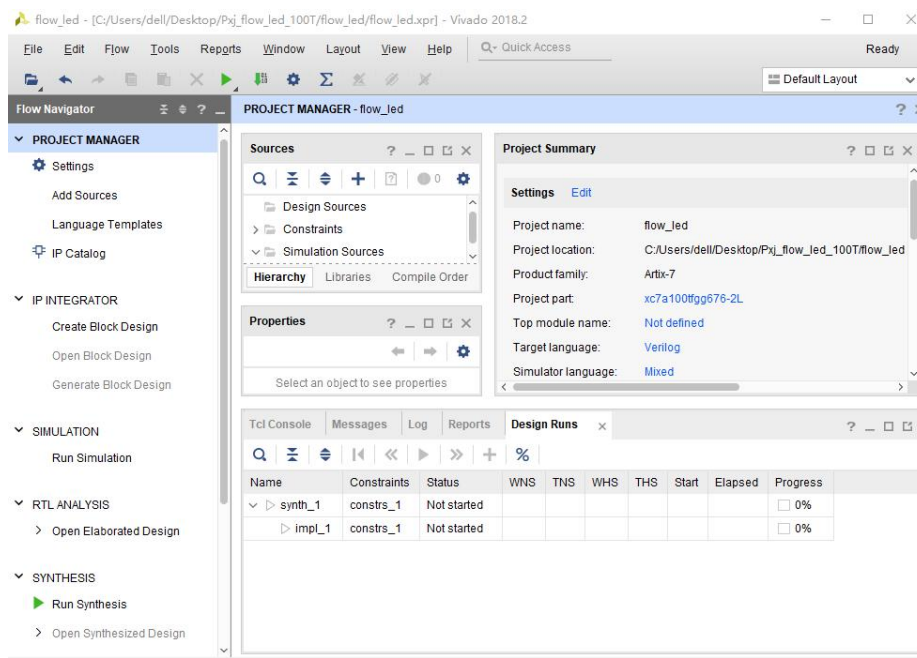
Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gb Transceivers	GTPE2 T
xc7a75tfgg676-2L	676	300	47200	94400	105	0	180	8	8
xc7a100tfgg676-2L	676	300	63400	126800	135	0	240	8	8

< Back Next > Finish Cancel

- 弹出 New Project Summary 对话框，从下面图中，可以看出整个工程的信息。单击 Finish 按钮，完成工程的创建。

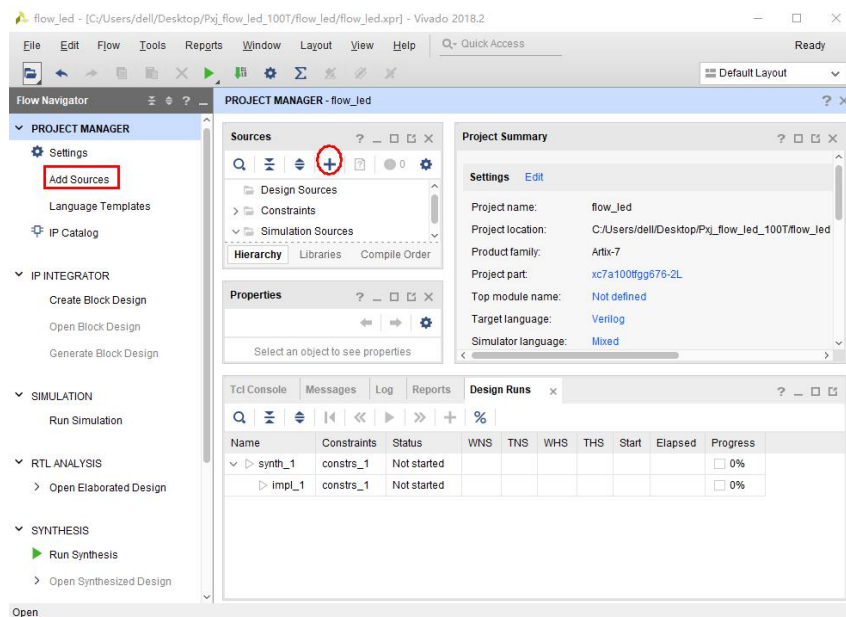


- 接下来我们会进入 Vivado 工程设计界面，如图所示。设计主界面主要包括：Flow Navigator、Project Manager、Design Runs 等模块。下面几个小节来逐渐熟悉 Vivado 工具设计界面。

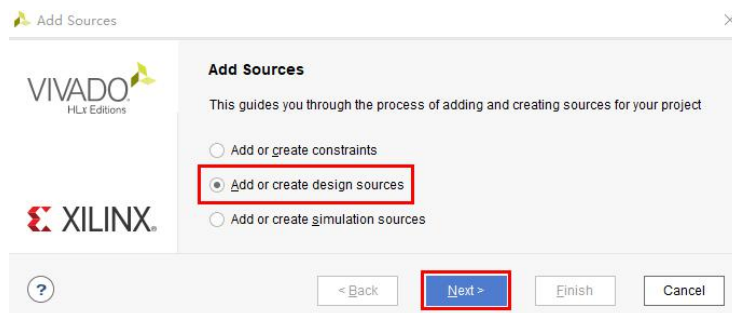


## ● 添加源文件

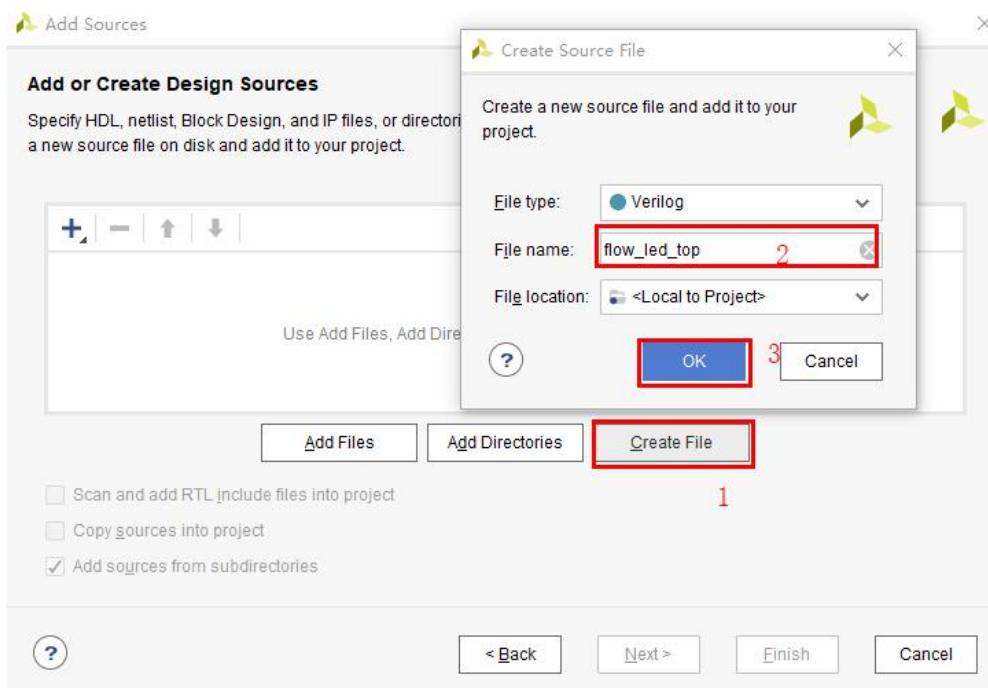
- 点击红色方框中的标签 Add Sources 或者圆圈中的加号,新添加设计源文件



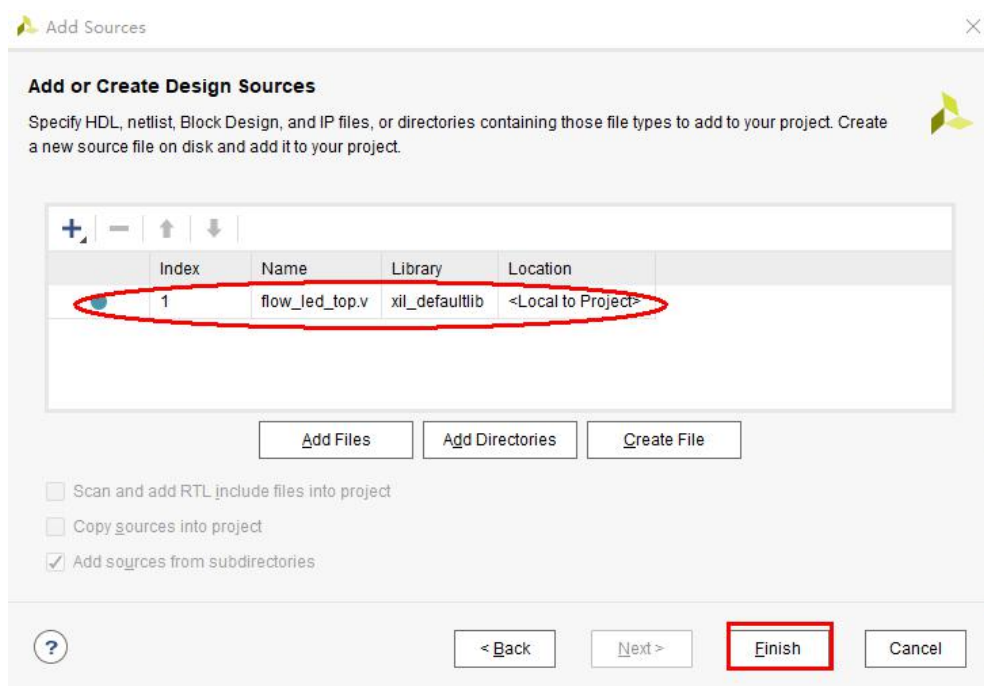
- 弹出如下图界面，选择 Add or create design sources,之后点击 Next。



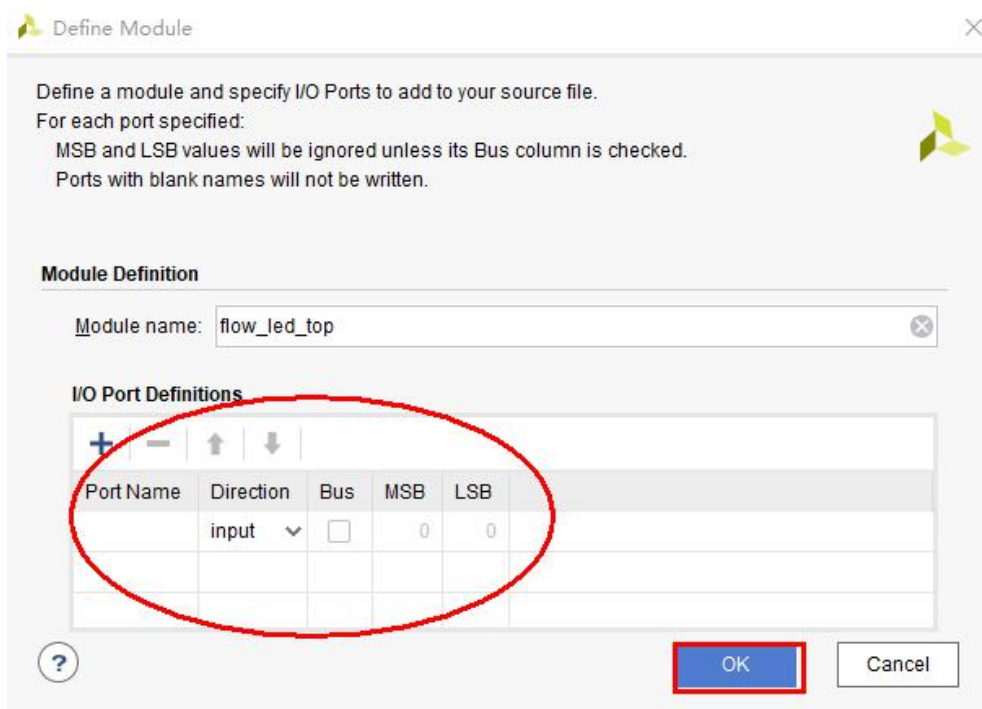
- 在弹出的界面中按照下面的红色方框顺序选择。在 1 红框中选择新建源文件，在 2 红框中添加源文件名字。



- 弹出下图，红圈中出现了新建立的 flow\_led\_top.v 源文件。点击红框中的 Finish。

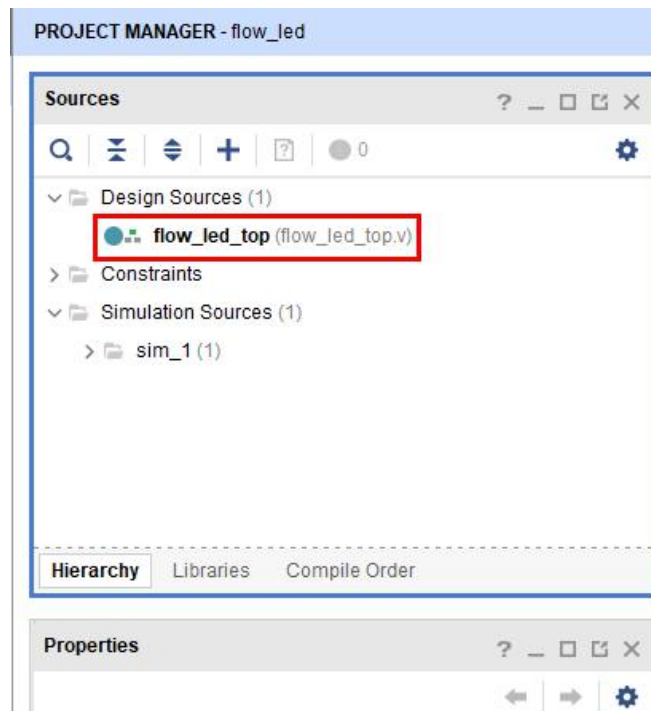


- 弹出 Define Module 对话框，可以在下图红色框中添加引输入输出的端口信息。也可以在源文件中以代码的形式输入，根据个人习惯不同选择合适的方式。此处建议直接点击 OK，接着点击 YES。



- 返回主界面到 Sources 的面板下，可以找到，我们添加的 flow\_led\_top.v 文件。

如下图所示：



- 双击打开 flow\_led\_top.v 源文件，打开结果如下图。如果代码编辑器关联了第三方代码编辑器，则会用关联的对应代码编辑器打开，如果没有关联，则用自带的编辑器打开。本系列课程均以关联 Notepad++ 编辑器为例说明。如果大家在使用中觉得该代码编辑器用的不习惯，也可以关联自己习惯的编辑器。下图中红色框中是 vivado 自动生成的一些头文件信息。我们只需要在 module 和 endmodule 之间添加我们设计的代码就行了。

```
flow_led_top.v
1  `timescale 1ns / 1ps
2  // Company:
3  // Engineer:
4  //
5  // Create Date: 2020/02/19 22:16:10
6  // Design Name:
7  // Module Name: flow_led_top
8  // Project Name:
9  // Target Devices:
10 // Tool Versions:
11 // Description:
12 //
13 // Dependencies:
14 //
15 // Revision:
16 // Revision 0.01 - File Created
17 // Additional Comments:
18 //
19 //////////////////////////////////////
20
21
22
23 module flow_led_top(
24
25 );
26 endmodule
```

➤ 添加完代码之后的结果如下图所示:

```
1 module flow_led_top
2     #(parameter end_cnt =50000000)
3     (
4     input wire      IN_CLK_50M    ,
5     input wire      PB            ,
6     output reg [7:0] LED = 8'hff
7     );
8
9     // cnt_reg define
10    reg [31:0] cnt = 0 ;
11    reg        sys_rst = 0 ;
12    //*****
13    /**      main code
14    //*****
15
16    always @ (posedge IN_CLK_50M )
17        sys_rst <= PB ;
18
19    always @ (posedge IN_CLK_50M )
20    begin
21        if (~sys_rst)
22            cnt <= 32'd0 ;
23        else
```



```

24     begin
25         if (cnt < ( end_cnt-1 ) )
26             cnt <= cnt + 1 ;
27         else
28             cnt <= 0 ;
29     end
30 end
31
32 always @ (posedge IN_CLK_50M )
33 begin
34     if ( ~sys_rst )
35         LED <= 8'hff ;
36     else
37         begin
38             if ( LED == 8'hff )
39                 LED <= 8'hfe ;
40             else
41                 begin
42                     if (cnt == end_cnt-1)
43                         LED <= { LED[6:0] , LED[7] };
44                     else
45                         LED <= LED ;
46                 end
47             end
48         end
49     end
50 endmodule

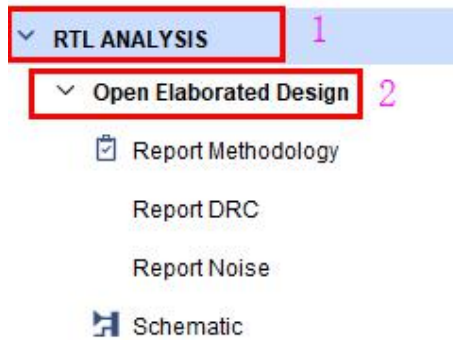
```

➤ 到此代码添加完毕。

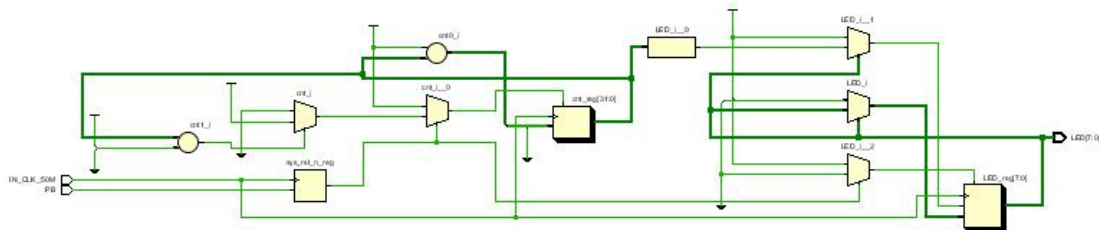
### ● RTL 代码分析

在设计完成电路结构之后，我们习惯于用硬件描述语言将所设计的电路结构以代码的形式描述呈现出来。这时由于对语法的不熟悉，或者逻辑上的错误，或者其他等原因，可能描述出来的电路结构和我们实际所设计的电路结构有些出入。这时候，就需要对所编写的源文件进行 RTL 结构，语法检查，并逐步修正，直到满足我们的设计需求。

➤ 在主界面的 Flow Navigator 面板之下，找到 RTL Analysis 选项卡如红色方框 1，展开；点击 Open Elaborated Design，如红色方框。点击 Schematic 标签。打开 RTL 原理图。

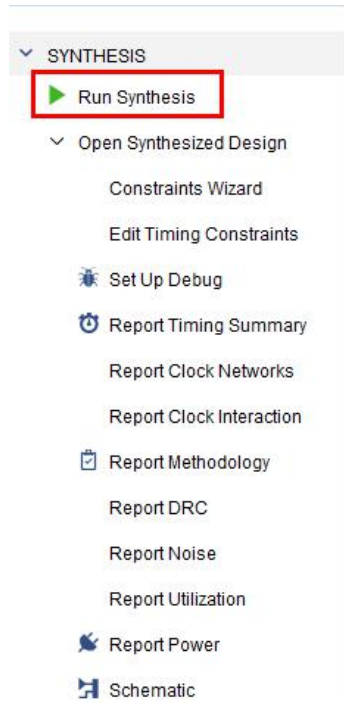


- 该 RTL 原理图就是 Vivado 软件依据我们前边所编写的 HDL 语言代码所描述生成的。从此图即可分析得出 HDL 源文件是否有问题。

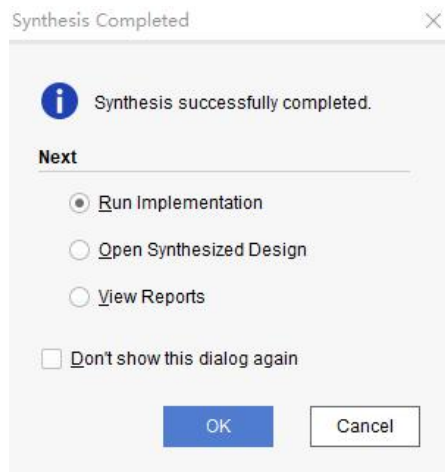


## ● 工程设计综合

- 在 Synthesis 选项卡之下点击 Run Synthesis，软件开始对工程进行综合。



- 弹出如下界面表示综合通过。



## ● 仿真测试建立

工程综合没有问题，只能说明代码在语法，逻辑方面没有问题。我们设计的电路结构是否能满足我们的功能需求，则需要进行仿真验证。

Vivado 具有 5 种仿真模式：

1、run behavioral simulation 行为级仿真（RTL 行为级仿真），也是通常说的功能仿真

2、post-synthesis function simulation 综合后的功能仿真

3、post-synthesis timing simulation 综合后带时序信息的仿真，和真实运行的时序就相差不远了。

4、post-implementation function simulation 布线后的功能仿真

5、post-implementation timing simulation（布局布线后的仿真）执行后的时序仿真最接近真实的时序波形。

几种仿真的区别如下：

RTL 行为级仿真：

在大部分设计中执行的第一个仿真将是 RTL 行为级仿真。这个阶段的仿真可以用来检查代码中的语法错误以及代码行为的正确性，其中不包括延时信息。如果没有实例化一些与器件相关的特殊底层元件的话，这个阶段的仿真也可以做到与器件无关。因此在设计的初期阶段不使用特殊底层元件即可以提高代码的可读性、可维护性，又可以提高仿真效率，且容易被重用。（绝大部分设计人员将这个阶段的仿真叫功能仿真！）。

综合后门级功能仿真（前仿真）：

一般在设计流程中的第二个仿真是综合后门级功能仿真。绝大多数的综合工具除了可以输出一个标准网表文件以外，还可以输出 Verilog 或者 VHDL 网表，其中标准网表文件是用来在各个工具之间传递设计数据的，并不能用来做仿真使用，而输出的 Verilog 或者 VHDL 网表可以用来仿真。

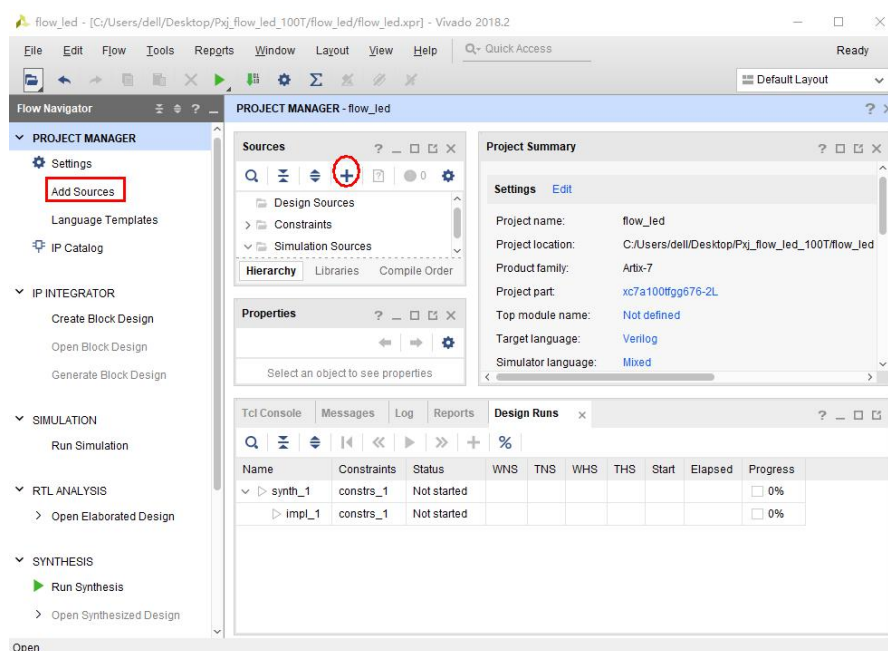
之所以叫门级仿真是因为综合工具给出的仿真网表已经是与生产厂家的器件的底层元件模型对应起来了，所以为了进行综合后仿真必须在仿真过程中加入厂家的器件库，对仿真器进行一些必要的配置，不然仿真器并不认识其中的底层元件，无法进行仿真。Xilinx 公司的集成开发环境 ISE 中并不支持综合后仿真，而是使用映射前门级仿真代替，对于 Xilinx 开发环境来说，这两个仿真之间差异很小。

时序仿真（后仿真）：

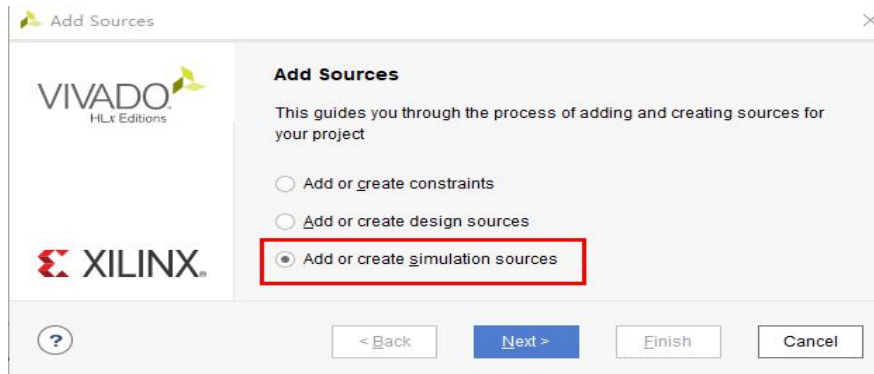
在设计流程中的最后一个仿真是时序仿真。在设计布局布线完成以后可以提供一个时序仿真模型，这种模型中也包括了器件的一些信息，同时还会提供一个 SDF 时序标注文件（Standard Delay format Timing Anotation）。SDF 时序标注最初使用在 Verilog 语言的设计中，现在 VHDL 语言的设计中也引用了这个概念。

过程如下：

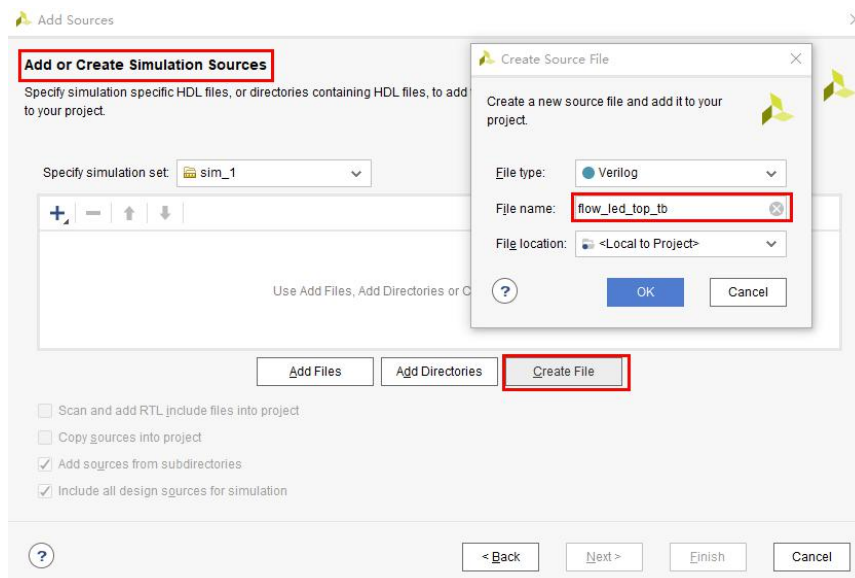
- 点击红色方框中的标签 Add Sources 或者圆圈中的加号,新添加仿真源文件。



- 弹出如下图界面，选择 Add or create simulation sources,之后点击 Next。



- 在弹出的界面中按照下面的红色方框中的信息填写，在右上红框中添加测试源文件名字，点击 OK。



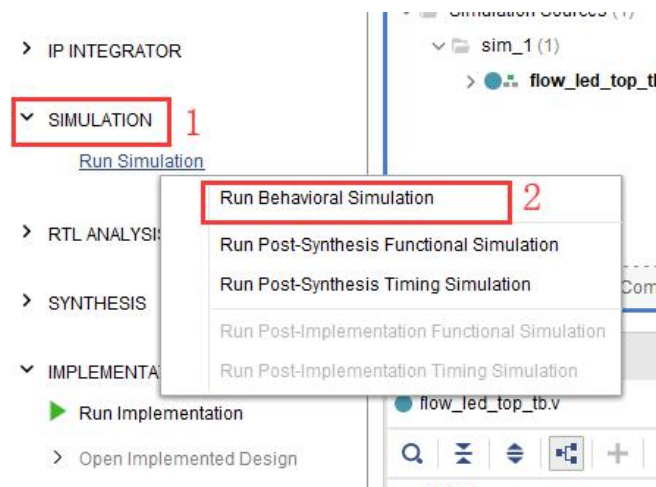
- 之后弹出的界面和前边新建设计源文件过程一样，直到返回主界面到 Sources 的面板下，可以找到，我们添加的 flow\_led\_top\_tb.v 文件。如下图所示：




- 双击打开仿真测试源文件，添加仿真测试代码。

```
1 module flow_led_tb();
2     reg      IN_CLK_50M ;
3     reg      PB          ;
4     wire [7:0] LED        ;
5
6     flow_led_top
7         #(.end_cnt (10))
8     flow_led_top_inst
9         (
10         .IN_CLK_50M (IN_CLK_50M) ,
11         .PB          (PB)          ,
12         .LED          (LED)
13         );
14
15     initial IN_CLK_50M=0 ;
16     always #10 IN_CLK_50M =~IN_CLK_50M ;
17
18     initial
19     begin
20         PB =0 ;
21         #40 ;
22         PB = 1 ;
23         #2000 ;
24         $stop ;
25     end
26 endmodule
```

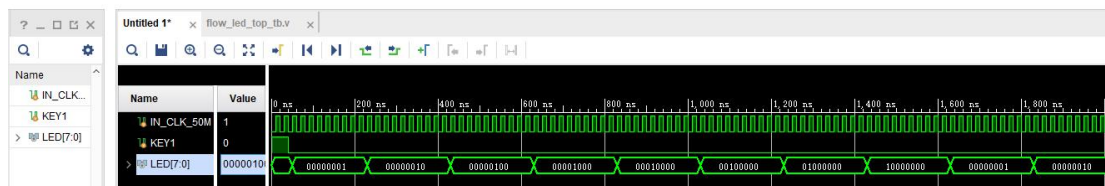
- 在 SIMULATION 选项卡下 Run Behavioral Simulation 进入仿真界面。



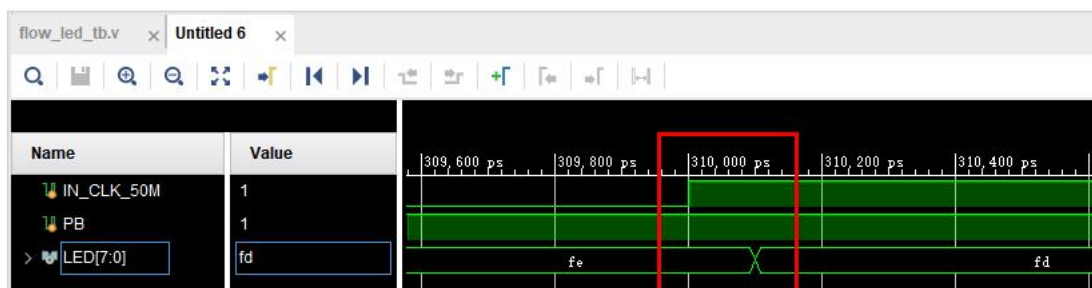
- 出现仿真波形界面，点击  启动仿真。设置 LED 数值的显示形式为二进制，

观察结果和设计目标相比较。

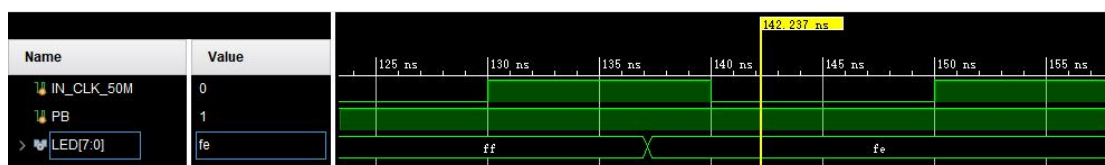
### ① RTL 行为级仿真结果图



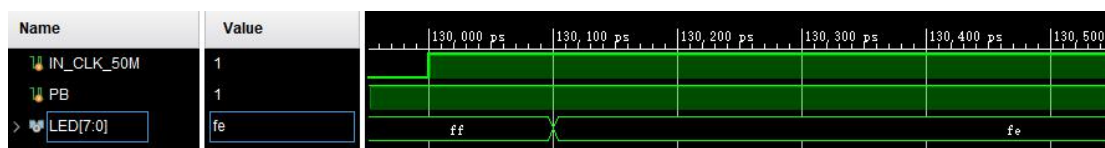
### ② 综合后的功能仿真结果图



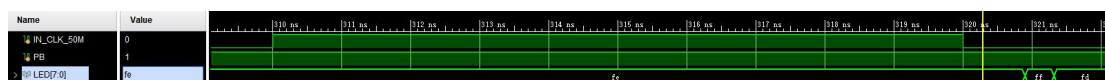
### ③ 综合后带时序信息的仿真结果图



### ④ 布线后的功能仿真结果图



### ⑤ 执行后的时序仿真结果图

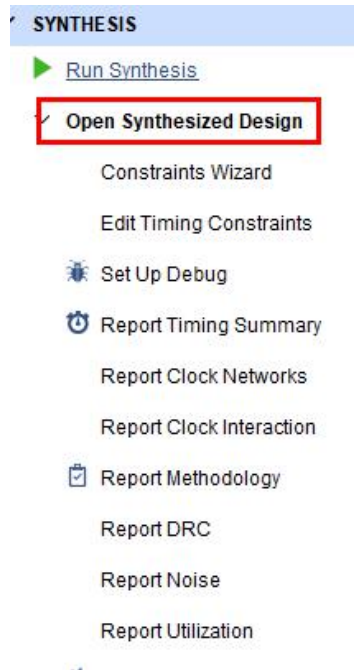


## ● 添加约束设计

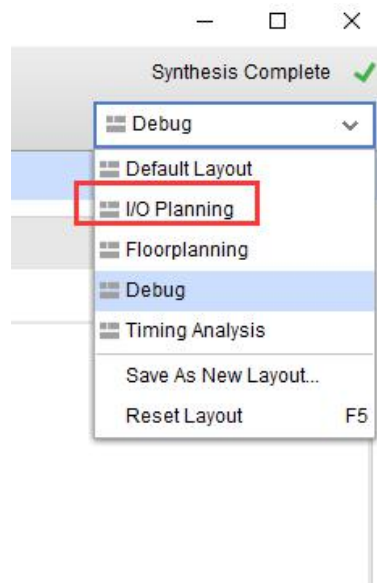
一般 FPGA 开发板上 FPGA 不会单独使用，都是有很多的外设和它互相连接一起使用。因此，在 FPGA 开发工程设计中，需要设计者在开发工具中指定一些对应的 IO 引脚信息，告诉 FPGA 接入时钟，输入信号，输出信号的相关信息。在 Vivado 开发工具中，我们可以通过 I/O Planner 进行 IO 引脚的约束（引脚绑定），通过 Timing Constraints Manager 进行设计的时序约束进而达到指定期望设计性能。

➤ 在 SYNTHESIS 的选项卡下点击 Open Synthesized Design , 如下图





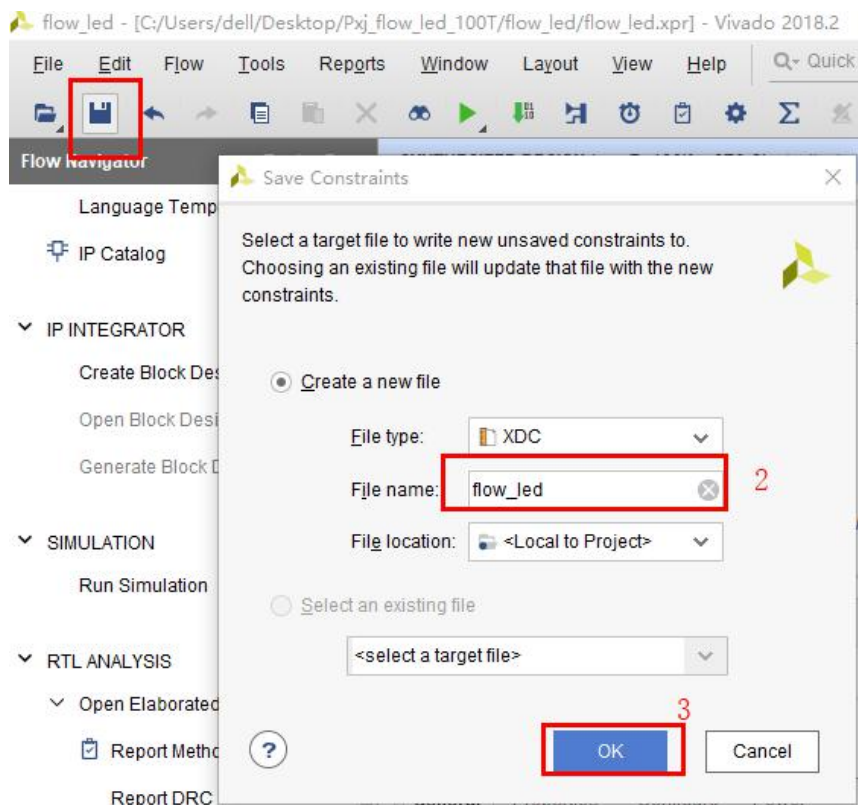
- 在右上角的框中点击倒三角,选择 I/O Planning。



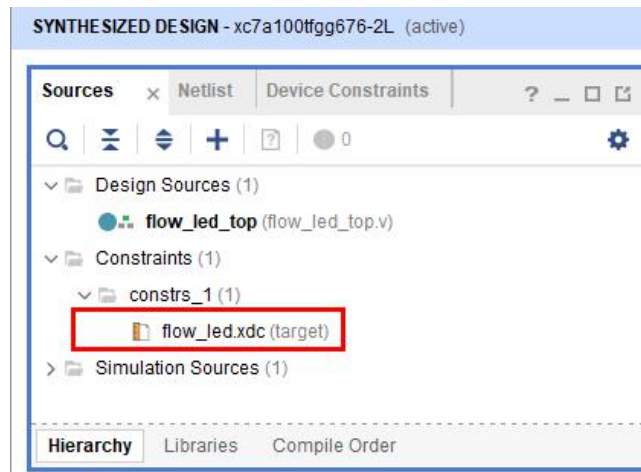
- 在下面的图示 I/O ports 填写红框中的相关信息。对应的端口绑定对应的引脚（引脚信息从对应开发板的原理图中找到），电平标准。

Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std
All ports (10)						
LED (8)						
LED[7]	OUT		N23	<input checked="" type="checkbox"/>	14	LVC MOS33*
LED[6]	OUT		N24	<input checked="" type="checkbox"/>	14	LVC MOS33*
LED[5]	OUT		P19	<input checked="" type="checkbox"/>	14	LVC MOS33*
LED[4]	OUT		N19	<input checked="" type="checkbox"/>	14	LVC MOS33*
LED[3]	OUT		N16	<input checked="" type="checkbox"/>	14	LVC MOS33*
LED[2]	OUT		P16	<input checked="" type="checkbox"/>	14	LVC MOS33*
LED[1]	OUT		M19	<input checked="" type="checkbox"/>	14	LVC MOS33*
LED[0]	OUT		N17	<input checked="" type="checkbox"/>	14	LVC MOS33*
Scalar ports (2)						
IN_CLK_50M	IN		U22	<input checked="" type="checkbox"/>	13	LVC MOS33*
KEY1	IN		M4	<input checked="" type="checkbox"/>	34	LVC MOS33*

- 填写完成之后，在下图数字 1 的位置点击保存引脚绑定的约束信息。数字 2 的地方填写约束信息的名字。数字 3 的地方点 OK。



- 此时在 Source 标签下的 Constainsts 下能找到一个刚才添加引脚信息后 Vivado 软件生成的文件名字为 flow\_led.xdc 的约束的信息。双击打开该文件。



- 下面框中内容为具体的 `flow_led.xdc` 的内容。此处不做讲解。如果大家以后对该内容语法比较熟悉的话，可以直接编辑生成代码生成该文件以脚本的方式添加约束信息。

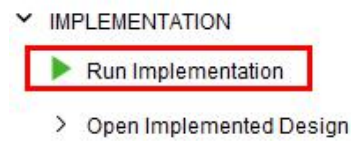
```
set_property IOSTANDARD LVCMOS33 [get_ports {LED[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports IN_CLK_50M]
set_property IOSTANDARD LVCMOS33 [get_ports PB]
set_property PACKAGE_PIN U22 [get_ports IN_CLK_50M]
set_property PACKAGE_PIN M4 [get_ports PB]
set_property PACKAGE_PIN N23 [get_ports {LED[7]}]
set_property PACKAGE_PIN N24 [get_ports {LED[6]}]
set_property PACKAGE_PIN P19 [get_ports {LED[5]}]
set_property PACKAGE_PIN N19 [get_ports {LED[4]}]
set_property PACKAGE_PIN N16 [get_ports {LED[3]}]
set_property PACKAGE_PIN P16 [get_ports {LED[2]}]
set_property PACKAGE_PIN M19 [get_ports {LED[1]}]
set_property PACKAGE_PIN N17 [get_ports {LED[0]}]
```

## ● 设计实现

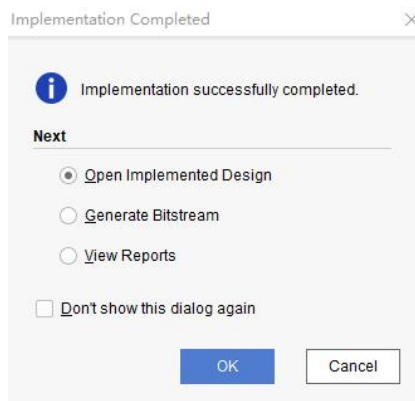
Vivado 下的 FPGA 设计实现是指由 FPGA 实现工具将 FPGA 综合后的电路网表针对某个具体指定的器件以及相关物理与性能约束进行优化、布局、布线并生成最终可以下载到 FPGA 芯片配置文件的过程。

具体步骤如下：

- 在左侧的 PROJECT MANAGER 的标签下找到 IMPLEMENTATION 选项卡，点击 Run Implementation 。



- 出现下图点击 OK，实现过程结束。

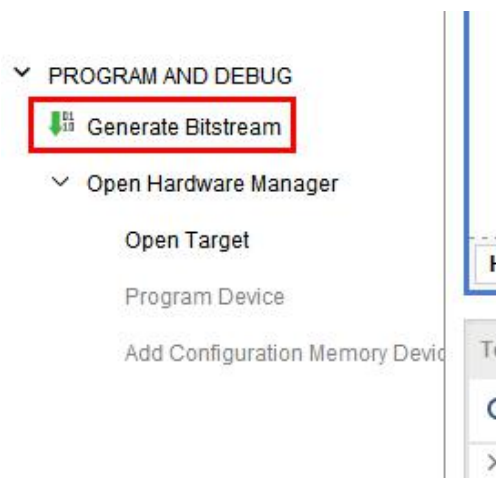


- 下载 FPGA 文件 Bit 的生成。

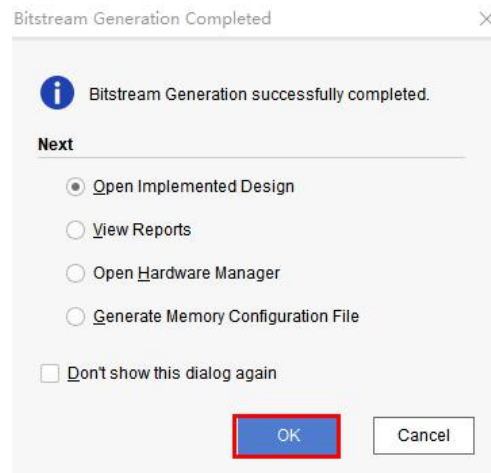
Bit 文件的生成是 FPGA 工程设计的最后一步了。把最后生成的 Bit 文件下载到 FPGA 开发板上，FPGA 开发板就可以正常工作了。

- Bit 文件的生成

在左侧的 PROGRAM AND DEBUG 选项卡展开，点击 Generate Bitstream 选项卡开始生成 Bit 文件。

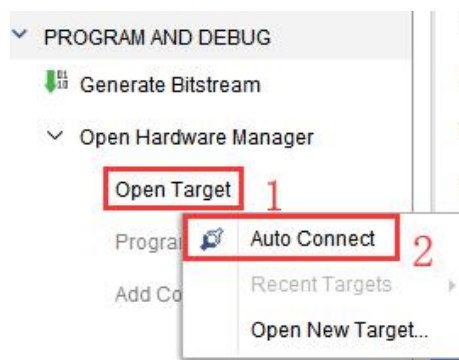


过一小会时间，会出现如下界面如下图所示：Bitstream Generation Completed 对话框，表明 bit 文件生成。

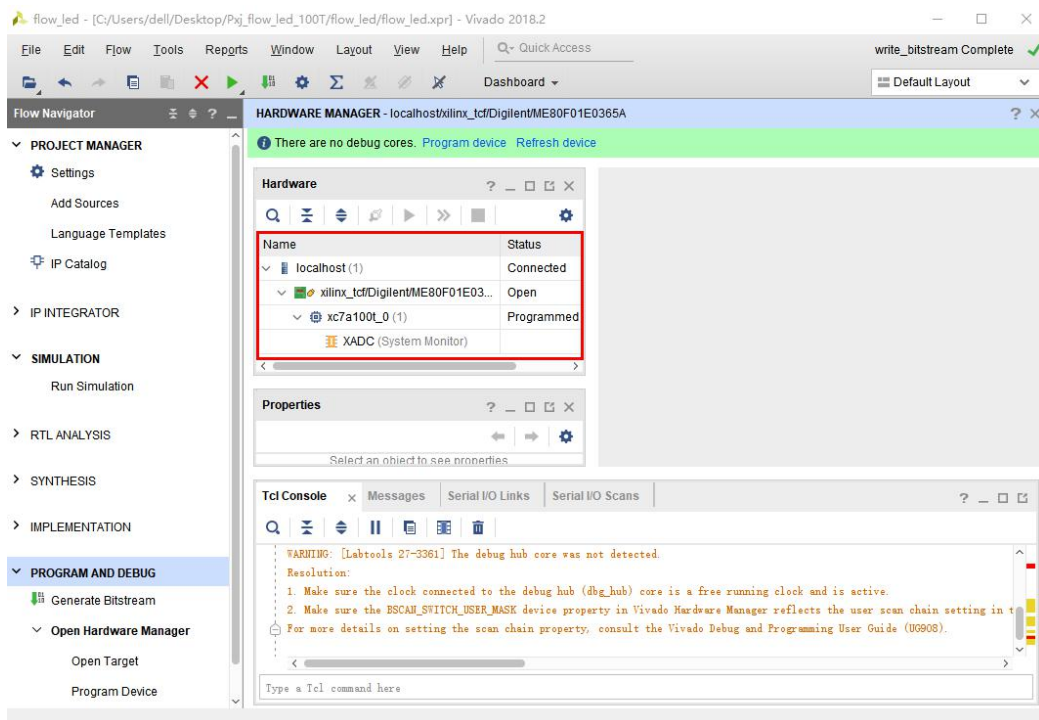


### ➤ Bit 文件的下载

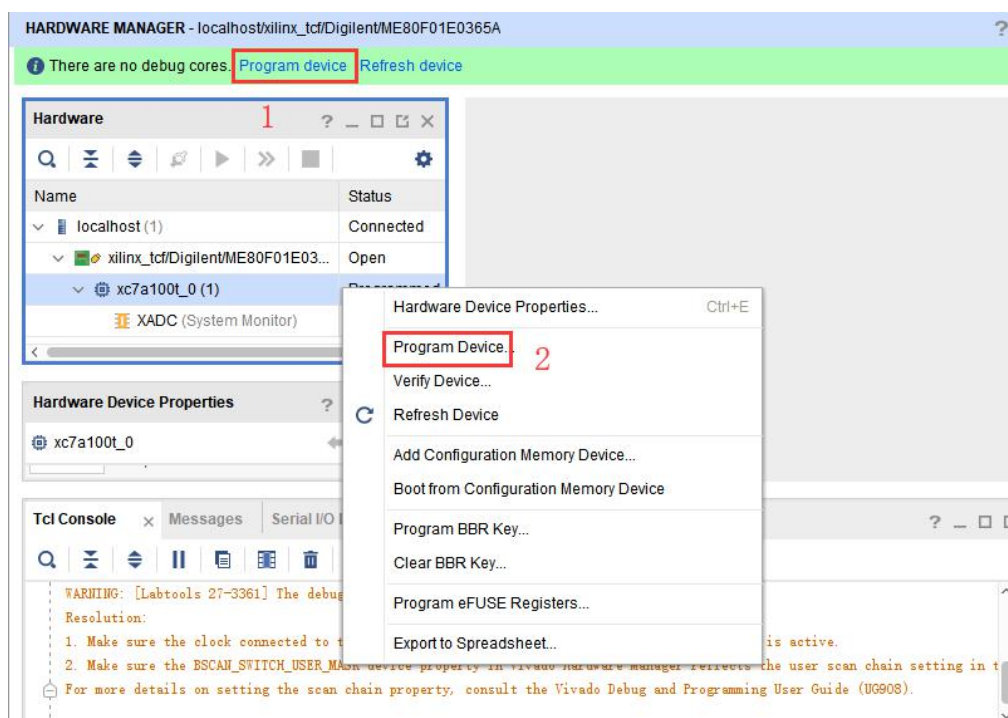
Bit 文件生成之后，首先连接好开发板。在左侧的 PROGRAM AND DEBUG 选项卡展开，点击 Open Target 选项，再点击 Auto Connect。



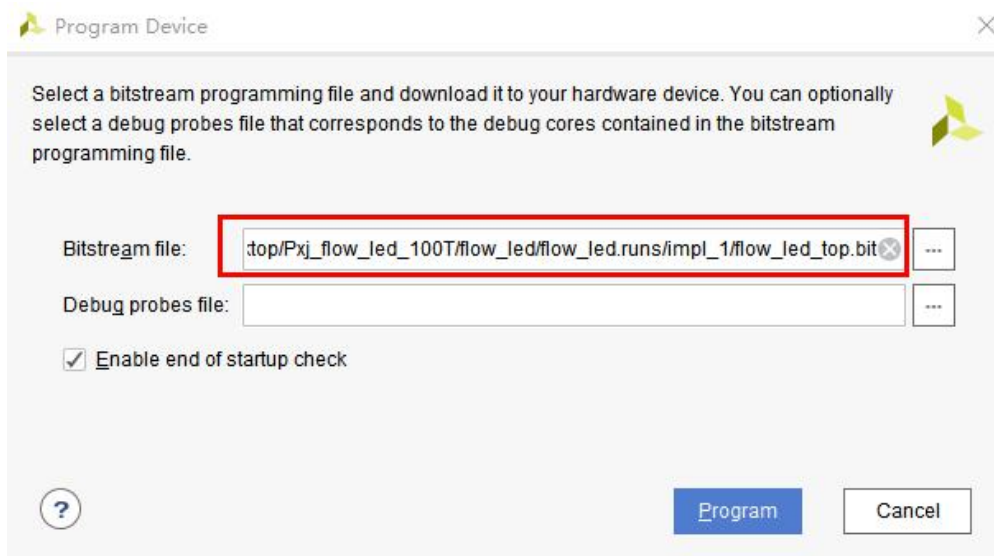
弹出如下界面，如下图红框中部分，即表示开发板与电脑已经正确连接。



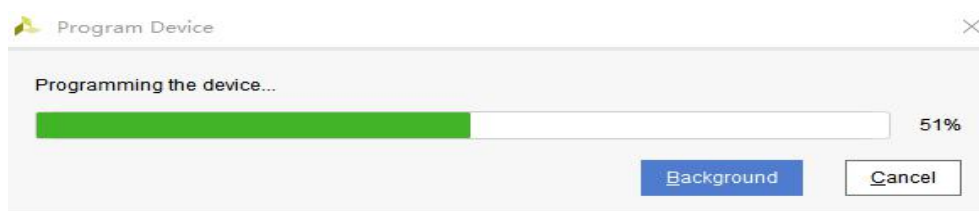
点击下图红框 1 或者选中目标板 xc7a100t, 如途中红框 2, 右键点击 Program Device。



弹出如下图, 红框中即为我们要往开发板里下载的 bit 文件, 如果该文件不是我们要下载的文件, 那么我们可以选择需要下载的 bit 文件添加。确定文件之后吗, 点击 Program 选项。



弹出如下进度条，绿条进度条走完之后即下载成功。

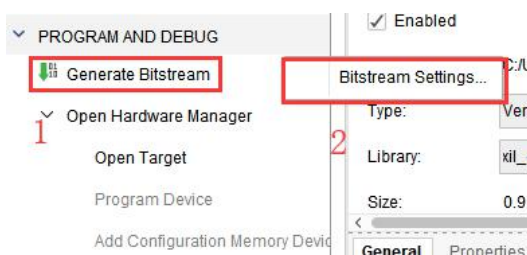


下载完成之后，观察开发板上的 LED 灯闪烁效果，验证流水灯 FPGA 工程是否满足我们的设计目标。如果满足，则设计完成，此次 vivado 基本设计流程结束。

- bin 文件生成与 flash 烧写

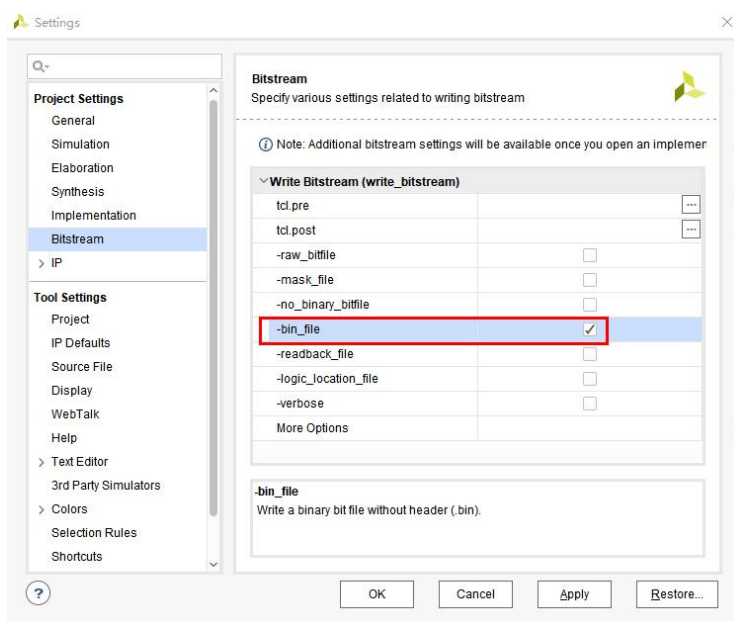
上一步生成的 bit 文件是下载到 FPGA 芯片里运行的，如果开发板断电重新上电后，则刚才下载的 bit 文件已经丢失。要想在开发板上电的时候，刚才的 FPGA 工程也随之正确运行，则需要把生成的电路结构存储到掉电不丢失的存储期间中去，供 FPGA 上电后读取。ICF\_PXJ100T 开发板上搭载的存储器件为一个 128Mb 的 Flash 芯片。把 FPGA 工程实现生成的 bin 文件下载到板载的 Flash 芯片里，即可满足上电启动运行的要求。

➤ 如下图， 在左侧的 PROGRAM AND DEBUG 选项卡展开，右键点击红框 1 的 Generate Bitstream，弹出红框 2 Bitstream Settings，点击 Bitstream Settings。

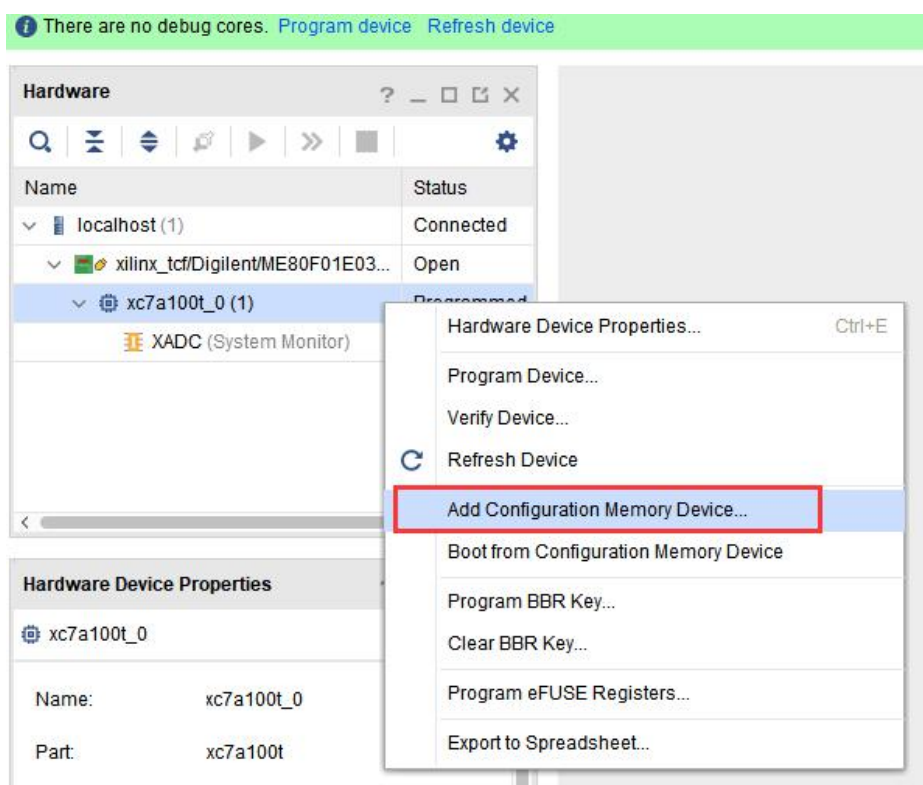




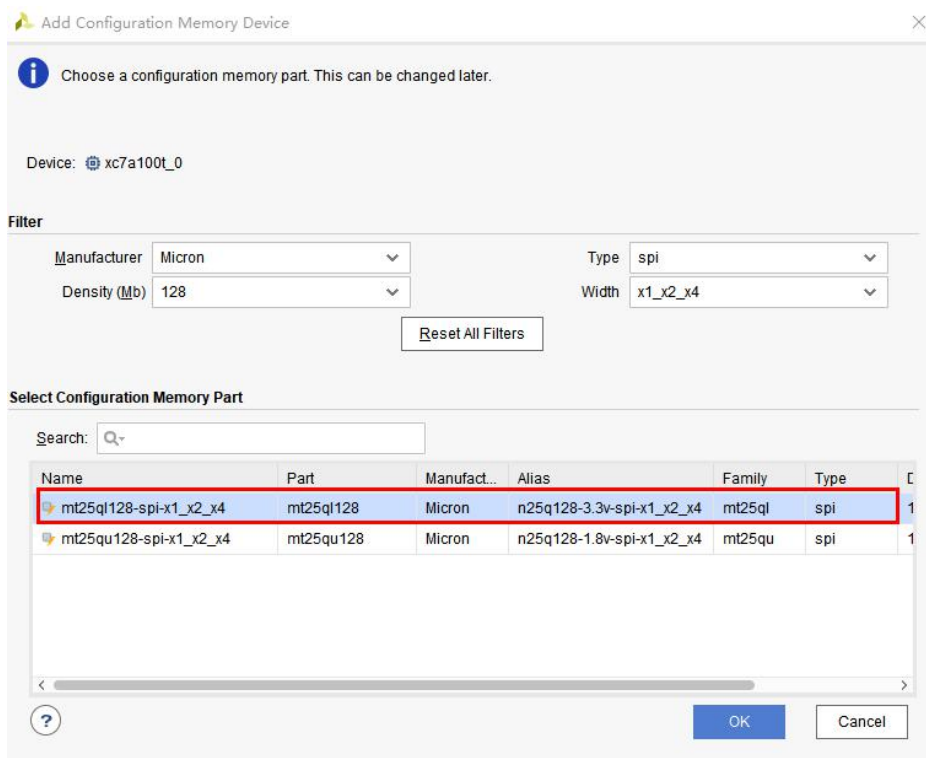
弹出如下界面，勾选红框中的选项，先点击 Apply,再点击 OK。



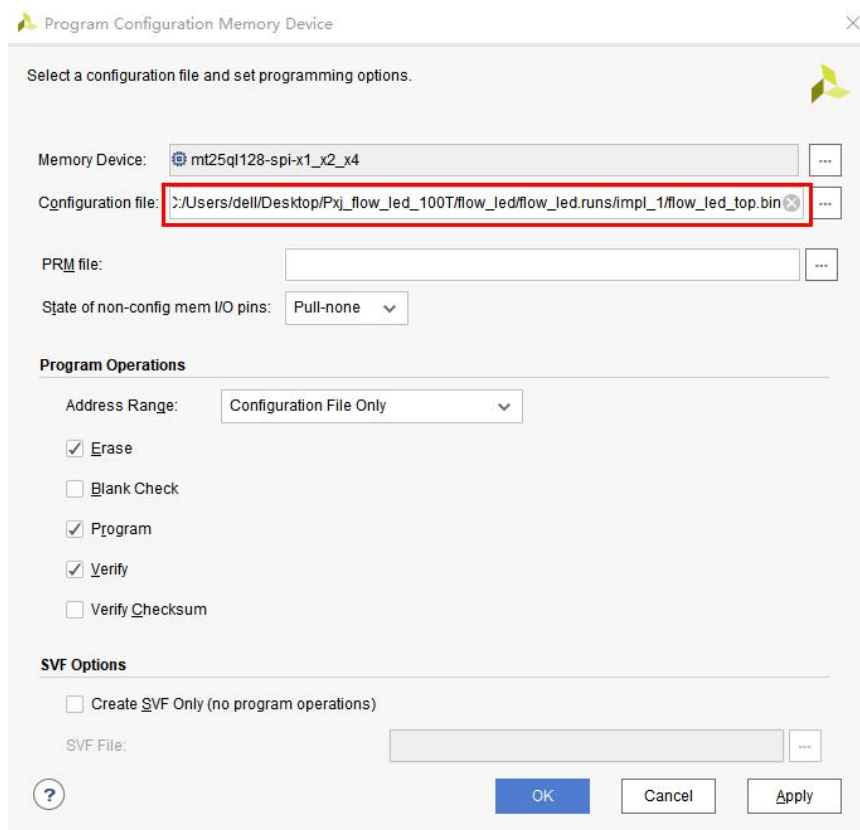
重复前边生成 bit 文件的流程，到电脑与开发板正常连接的步骤。选中目标器件，右键选择 Add Configuration Memory Device。



在弹出的界面按照下图填写相关信息，选则红框中板载对应的 flash 芯片，点击 OK。



在弹出的下图红框中选中工程生成的 bin 文件，点击 Apply,再点击 OK。

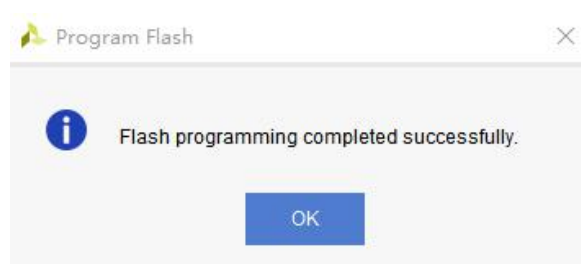
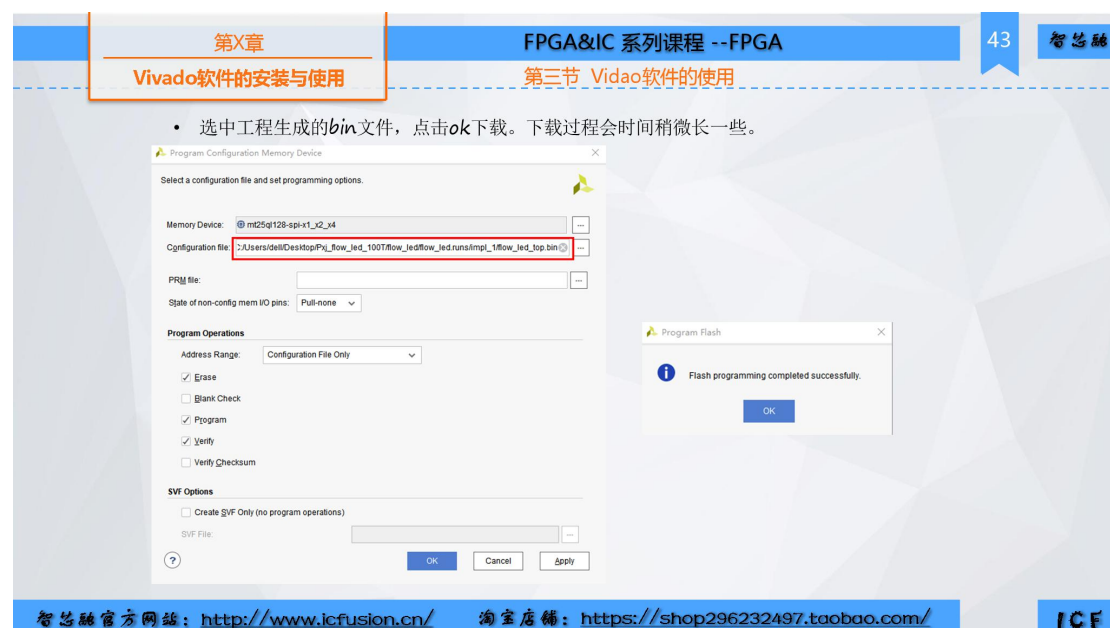


弹出如下的进度条。下载 bin 文件到 flash 里边，要比下载 bit 文件到 FPGA

慢很多。



出现如下界面表明烧写 flash 成功。此时，按下开发板上的 RESET 按键或者开发板断电之后再上电，过大约 15s 之后，程序就会运行起来。



## ● 程序下板调试

有些时候对于一些复杂，不方便仿真的 FPGA 工程，我们可以直接下载程序到 FPGA 开发板进行信号调试，把需要观察的信号的数据信息从 FPGA 芯片里读出来，查看信号的数据流走向是否符合我们的设计目标。

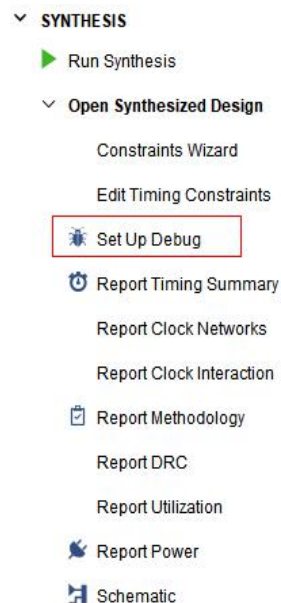
Vivado 软件进行信号调试的方法如下（此处只介绍一种常用的）：

- 把需要观察的信号进行标记：比如我们前边的一个流水灯的实验工程，需要观察 cnt 和 LED 这两个信号。我们可以使用如下语法(**\***

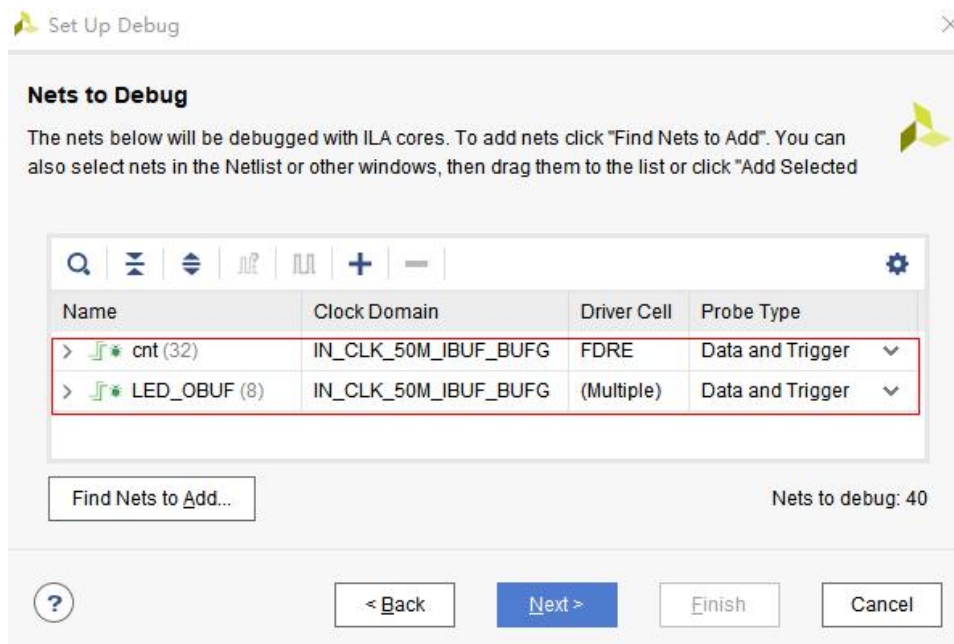
**MARK\_DEBUG="true" \*)**把这两个信号标注，告诉 Vivado 软件，我们在后边的调试阶段，需要对这两个信号进行观察。

```
module flow_led_top
    #(parameter end_cnt = 10)  // for timeing simlation
    // #(parameter end_cnt = 500000000)
    (
        input  wire      IN_CLK_50M    ,
        input  wire      PB             ,
        (* MARK_DEBUG="true" *) output reg [7:0] LED = 8'hff
    );
    // cnt_reg  define
    (* MARK_DEBUG="true" *) reg [31:0] cnt = 0 ;
    reg sys_rst_n = 0 ;
```

➤ 标记完成之后，进行综合。综合完成之后，如下图红框，选择 Set up Debug。

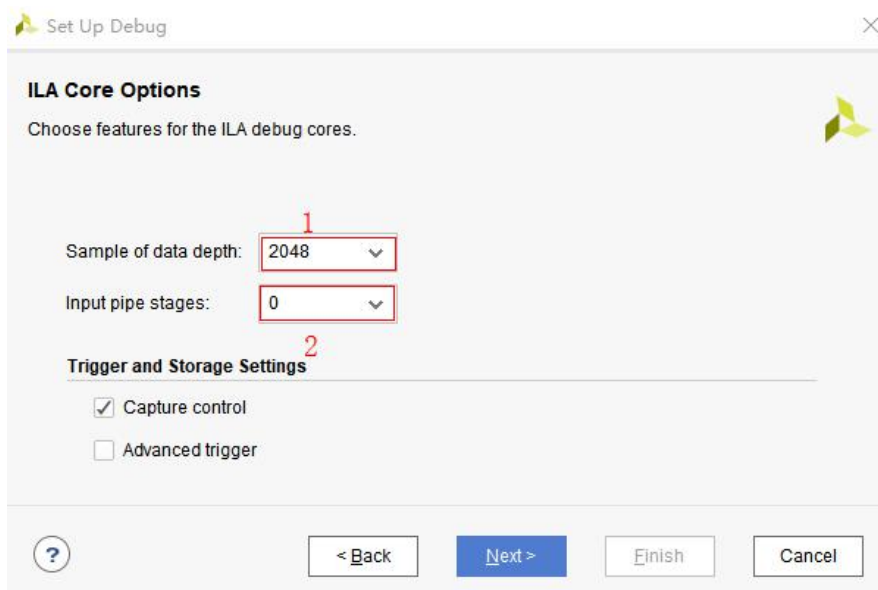


➤ 一直点击 Next,直到出现如下界面：红框中就是我们要观察的信号。选择正确的观察时钟域，点击 Next。



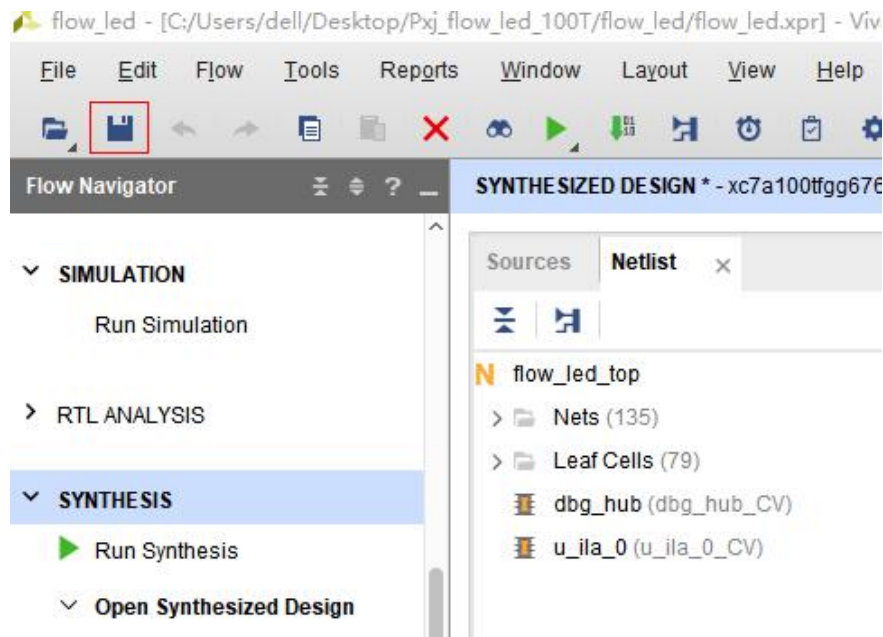
- 如下图，在红框 1 中设置观察数据的深度。当然，深度越大，观察的数据越多，当然需要的逻辑资源也就越多，在实现布线阶段 Vivado 软件所需时间也就越长。由于 FPGA 芯片的逻辑资源是有限的，另外调试信号所占用的逻辑资源越多，对 FPGA 工程的布线实现是不利的，所以调试阶段的深度是一个综合考量的结果，此处我们设置深度 2048。

红框 2 中的设置数值，表示我们在调试观察信号的数值时，需要增加几个流水节拍，此处不需要，设置为 0。设置好之后，点击 Next，再点击 Finish。



- 设置完 Debug 信号之后，点击下图中红框位置，保存 Debug 设置。





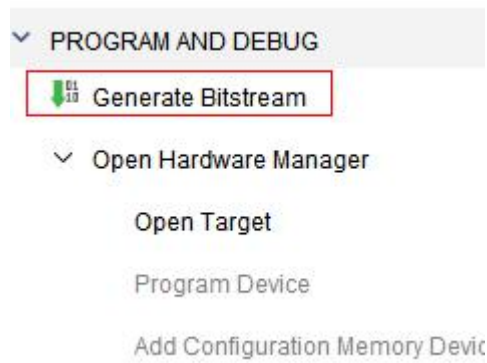
- 在设置保存完 Debug 信号之后，Vivado 软件会在约束文件 .xdc 文件中写入对应的约束信息（下面红框中信息）。

```

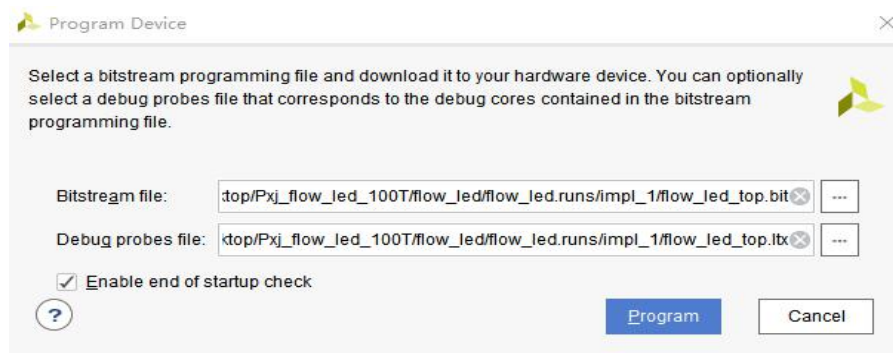
15 set_property PACKAGE_PIN P19 [get_ports {LED[5]}]
16 set_property PACKAGE_PIN N19 [get_ports {LED[4]}]
17 set_property PACKAGE_PIN N16 [get_ports {LED[3]}]
18 set_property PACKAGE_PIN P16 [get_ports {LED[2]}]
19 set_property PACKAGE_PIN M19 [get_ports {LED[1]}]
20 set_property PACKAGE_PIN N17 [get_ports {LED[0]}]
21
22
23 create_debug_core u_ila_0 ila
24 set_property ALL_PROBE_SAME_MU true [get_debug_cores u_ila_0]
25 set_property ALL_PROBE_SAME_MU_CNT 2 [get_debug_cores u_ila_0]
26 set_property C_ADV_TRIGGER false [get_debug_cores u_ila_0]
27 set_property C_DATA_DEPTH 2048 [get_debug_cores u_ila_0]
28 set_property C_EN_STRG_QUAL true [get_debug_cores u_ila_0]
29 set_property C_INPUT_PIPE_STAGES 0 [get_debug_cores u_ila_0]
30 set_property C_TRIGIN_EN false [get_debug_cores u_ila_0]
31 set_property C_TRIGOUT_EN false [get_debug_cores u_ila_0]
32 set_property port_width 1 [get_debug_ports u_ila_0/clock]
33 connect_debug_port u_ila_0/clock [get_nets [list IN_CLK_50M_IBUF_BUFG]]
34 set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe0]
35 set_property port_width 8 [get_debug_ports u_ila_0/probe0]
36 connect_debug_port u_ila_0/probe0 [get_nets [list {LED_OBUF[0]} {LED_OBUF[1]} {LED_OBUF[2]} {LED_OBUF[3]} {LED_OBUF[4]} {LED_OBUF[5]}]]
37 create_debug_port u_ila_0 probe
38 set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe1]
39 set_property port_width 32 [get_debug_ports u_ila_0/probe1]
40 connect_debug_port u_ila_0/probe1 [get_nets [list {cnt[0]} {cnt[1]} {cnt[2]} {cnt[3]} {cnt[4]} {cnt[5]} {cnt[6]} {cnt[7]} {cnt[8]} {cnt[9]} {cnt[10]} {cnt[11]} {cnt[12]} {cnt[13]} {cnt[14]} {cnt[15]} {cnt[16]} {cnt[17]} {cnt[18]} {cnt[19]} {cnt[20]} {cnt[21]} {cnt[22]} {cnt[23]} {cnt[24]} {cnt[25]} {cnt[26]} {cnt[27]} {cnt[28]} {cnt[29]} {cnt[30]} {cnt[31]}]]
41 set_property C_CLK_INPUT_FREQ_HZ 300000000 [get_debug_cores dbg_hub]
42 set_property C_ENABLE_CLK_DIVIDER false [get_debug_cores dbg_hub]
43 set_property C_USER_SCAN_CHAIN 1 [get_debug_cores dbg_hub]
44 connect_debug_port dbg_hub/clock [get_nets IN_CLK_50M_IBUF_BUFG]
45

```

- 直接点击生成 Bit 文件标签（Generate Bits）。此时生产 Bit 文件的时间需要时间会比单独生成 Bit 文件要长一些。



- 下载生成的 Bit 文件和 ltx 文件到 FPGA 开发板。



- 调试结果如下图，从图中可以看出信号实际在 FPGA 开发板中的运行状态和我们的设计目标保持一致的。

