Vivado 下 LED 流水灯实验

黑金动力社区 2019-11-12

1 实验简介

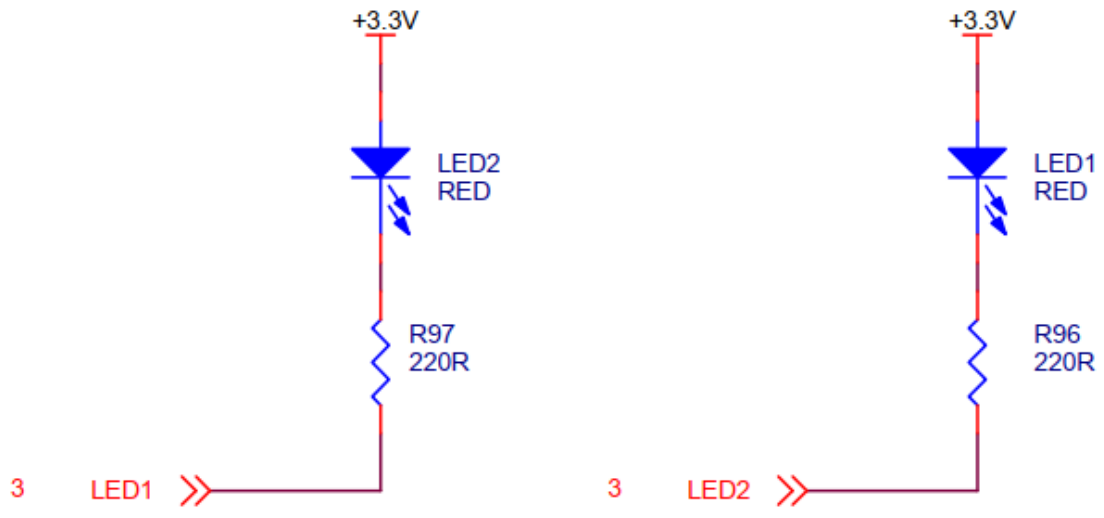
通过 LED 流水灯实验，介绍使用 vivado 软件开发 FPGA 的基本流程，器件选择、设置、代码编写、编译、分配管脚、下载、程序 FLASH 固化、擦除等；同时也检验板上 LED 灯是否正常。

2 实验环境

- Windows 7 SP1 64 位
- vivado 2017.4
- 黑金 FPGA 开发板（AV7K325 开发板）

3 实验原理

3.1 LED 硬件电路



AV7K325 开发板 LED 部分原理图

版权所有

从上面的 LED 部分原理图可以看出， AV7K325 开发板将 IO 经过一个电阻与 LED 连接，FPGA 的 IO 输出低电平点亮 LED。IO 输出高电平 LED 灯熄灭。

3.2 程序设计

FPGA 的设计中通常使用计数器来计时，对于采用 200Mhz 的系统时钟，一个时钟周期是 5ns，那么表示一秒需要 200000000 个时钟周期，如果一个时钟周期计数器累加一次，那么计数器从 0 到 199999999 正好是 200000000 个周期，就是 1 秒的时钟。

程序中定义了一个 32 位的计数器：

```
//Define the time counter
reg [31:0]      timer;
```

最大可以表示 4294967295，十六进制就是 FFFFFFFF，如果计数器到最大值，可以表示 21.474836475 秒。程序设计中是每隔 0.25 秒 LED 变化一次，一共消耗 1 秒做一个循环。

```
always@(posedge sys_clk or negedge rst_n)
begin
    if (~rst_n)
           timer <= 32'd0;
    else if (timer == 32'd199_999_999)
           timer <= 32'd0;
    else
           timer <= timer + 1'b1;
end
```

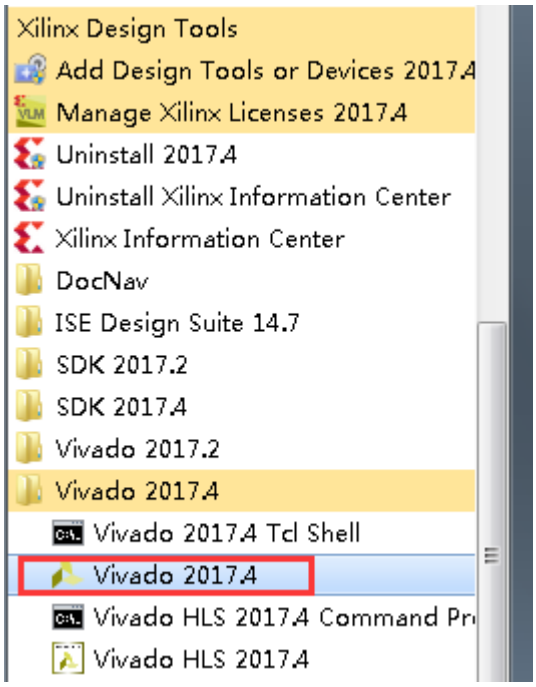
在 0.25 秒、0.5 秒、0.75 秒、1 秒到来的时刻分别改变 LED 的状态，其他时候都保持原来的值不变。

```
// LED control
always@(posedge sys_clk or negedge rst_n)
begin
    if (~rst_n)
           led <= 2'b00;
    else if (timer == 32'd49_999_999)
           led <= 2'b01;
    else if (timer == 32'd99_999_999)
           led <= 2'b10;
    else if (timer == 32'd149_999_999)
           led <= 2'b00;
    else if (timer == 32'd199_999_999)
           led <= 2'b11;
end
```

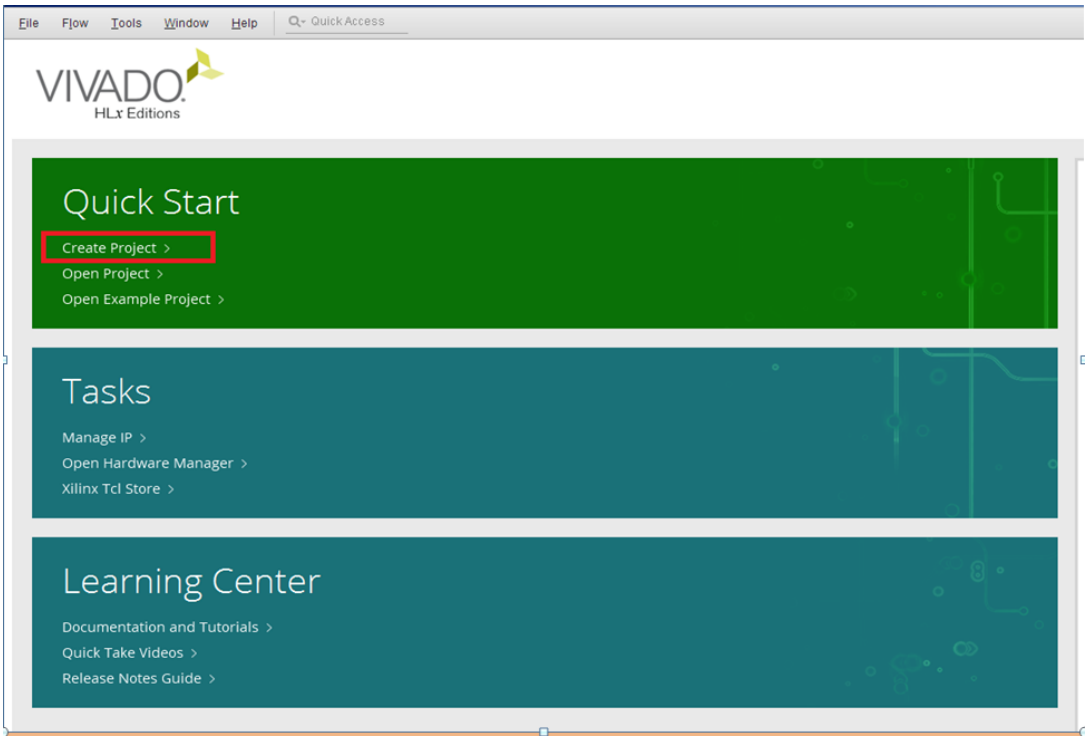
4 Vivado 工程

4.1 创建工程

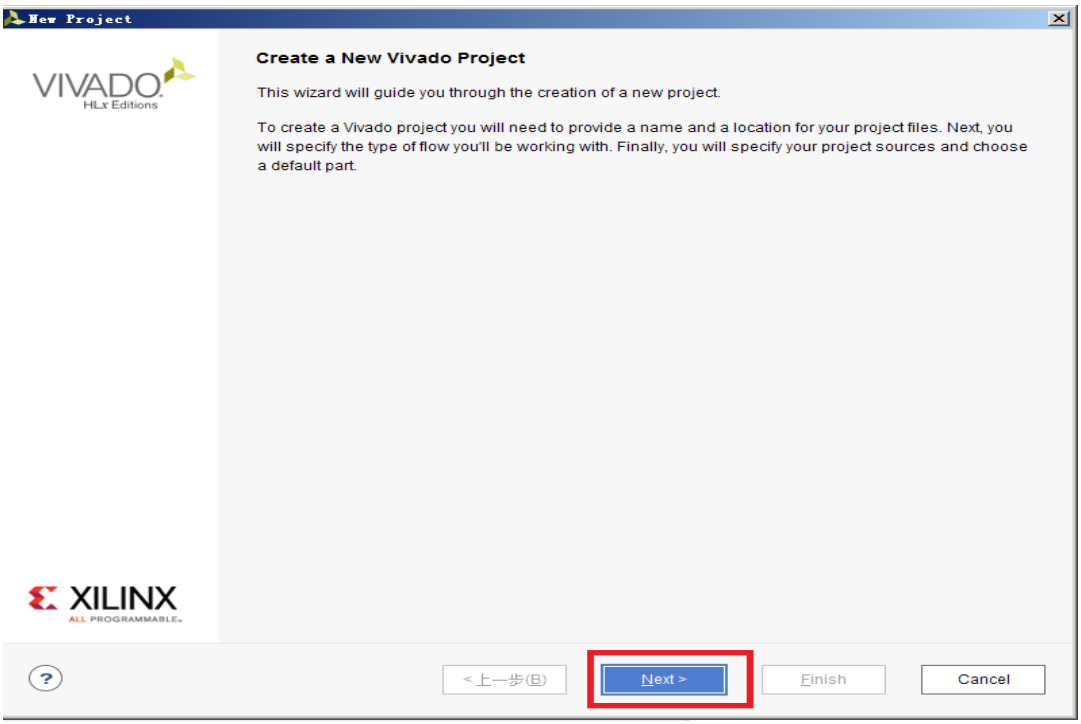
1. 启动 Vivado 2017.4 开发环境(在开始菜单中选择 Xilinx Design Tools->Vivado 2017.4->Vivado 2017.4。或者双击桌面的 Vivado 2017.4 的图标直接打开软件。



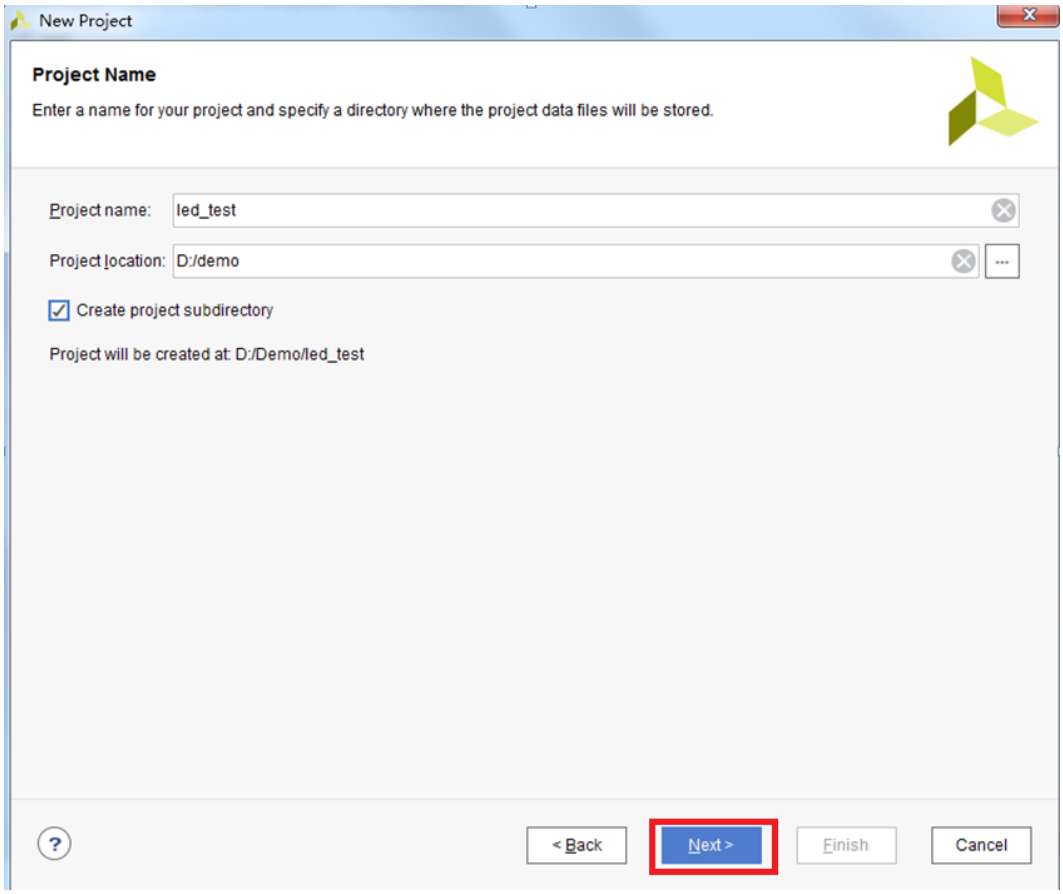
2. 在 Vivado 2017.4 开发环境里双击 Create Project，如下图：



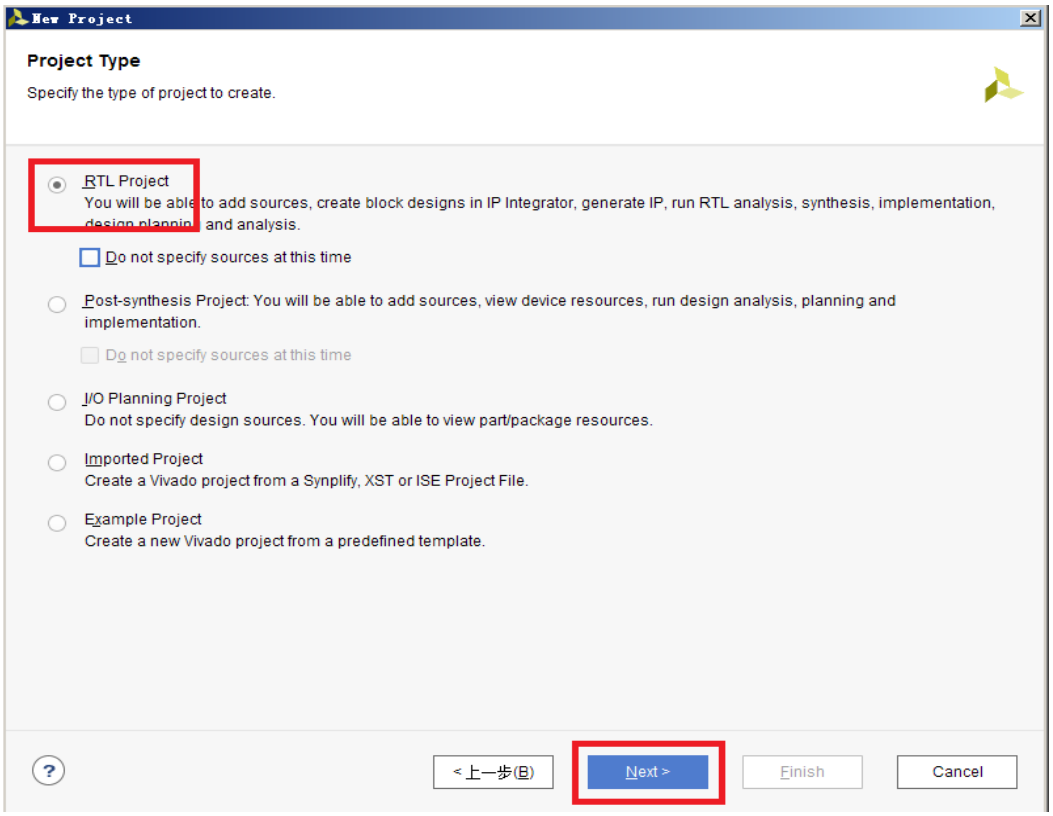
3. 弹出一个 Vivado 的工程向导，点击 Next 按钮。



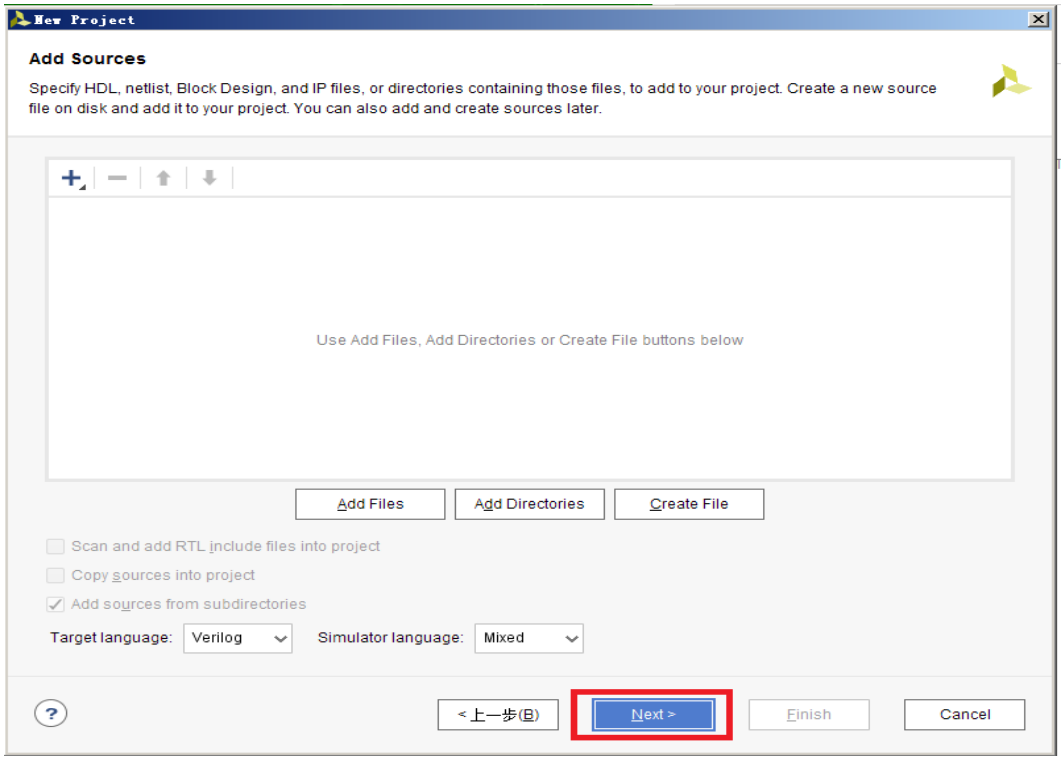
4. 在弹出的对话框中输入工程名和工程存放的目录，这里取一个 led_test 的工程名，点击 Next。



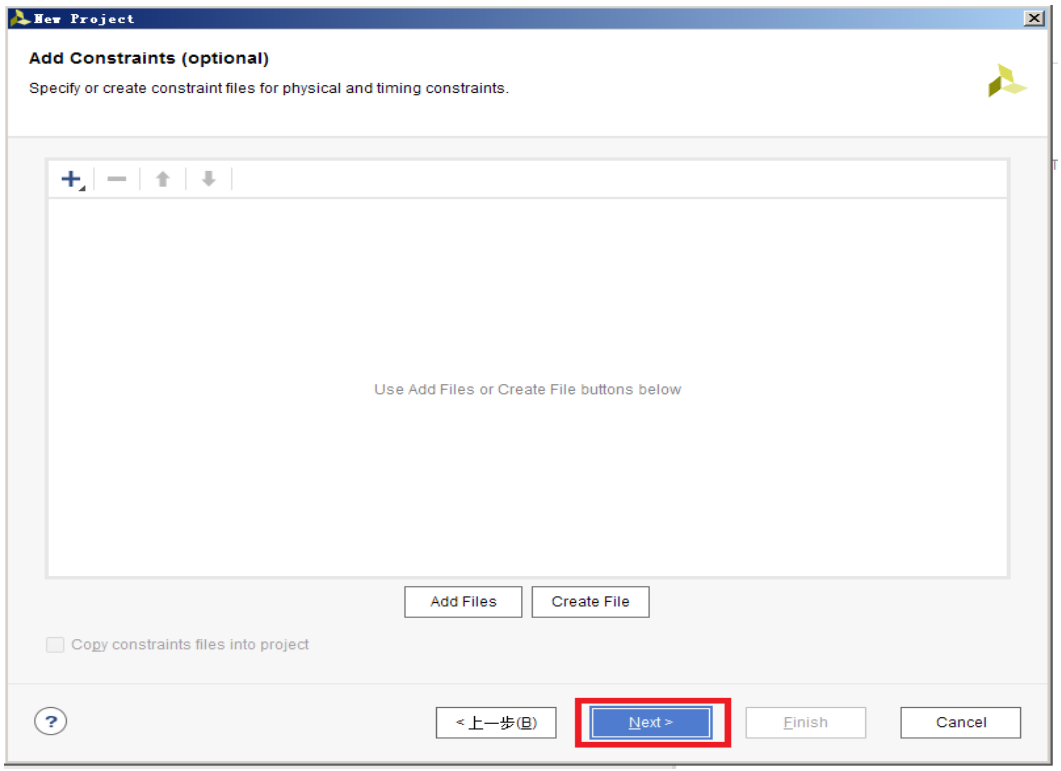
5. 在下面的对话框中默认选择 RTL Project, 因为我们这里使用 verilog 行为描述语言来编程。下面的 Do not specify source at this time 的勾也可以打上。如果不打上，下一步会进入添加 source file 界面，



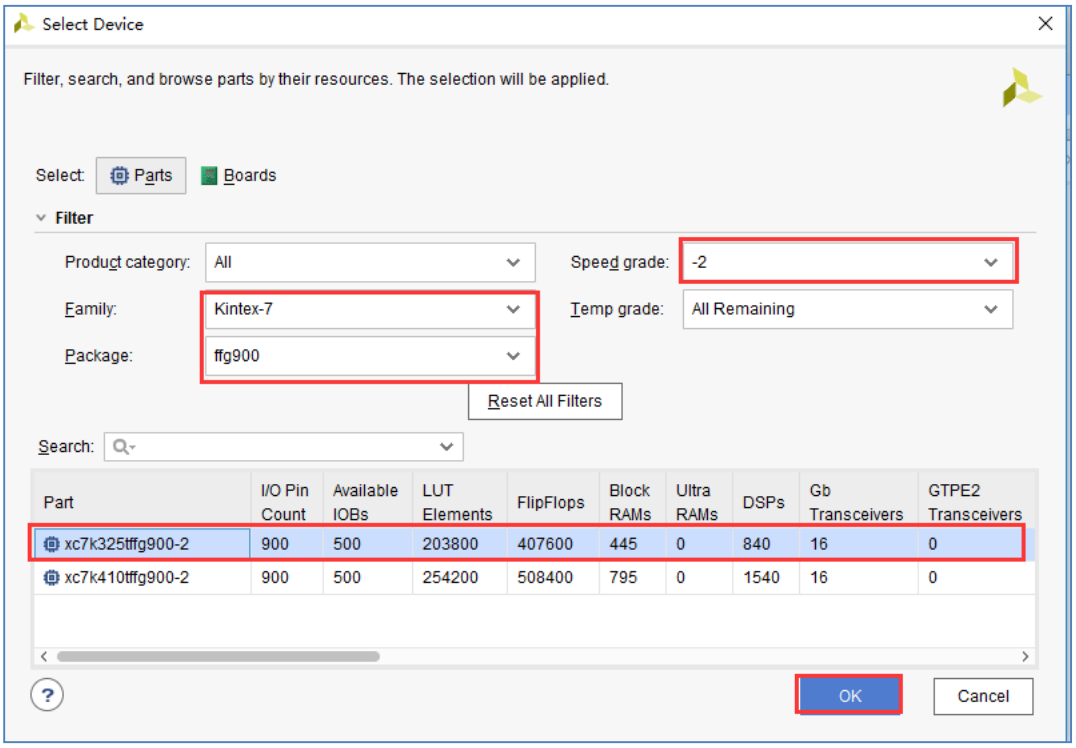
6. 进入添加 source file 界面，这里先不添加任何设计文件。点击 Next



7. 提示是否添加已有的约束文件，这里约束文件我们也没有设计好，也不添加。



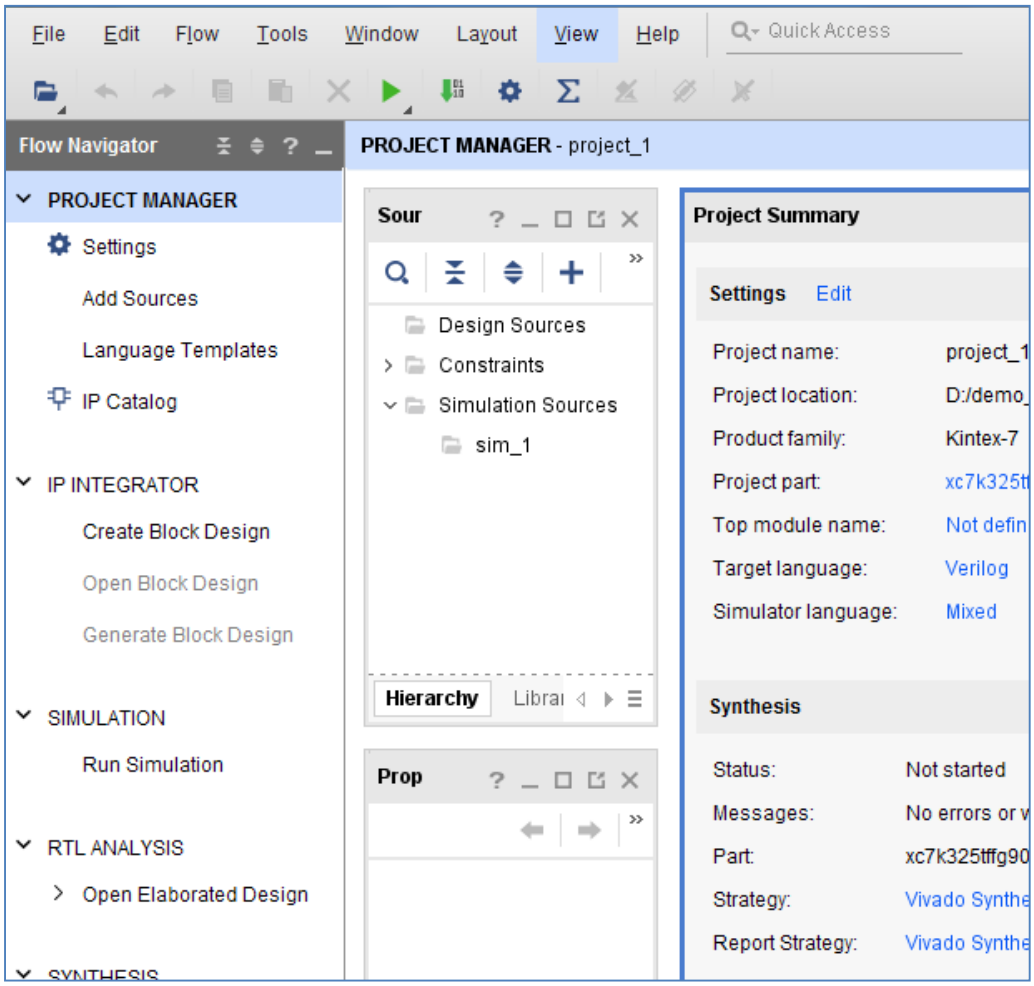
8. 在接下来的对话框选择所用的 FPGA 器件，以及进行一些配置。FPGA 芯片型号一定要跟开发板上的型号一致，AV7K325 开发板首先在 Family 栏里选择 Kintex-7, Speed grade 栏选择-2, 在 Package 栏选择 ffg900, 然后在下面的列表中选择 xc7k325tffg900-2；单击 NEXT 进入下一界面：



AV7K325 开发板 FPGA

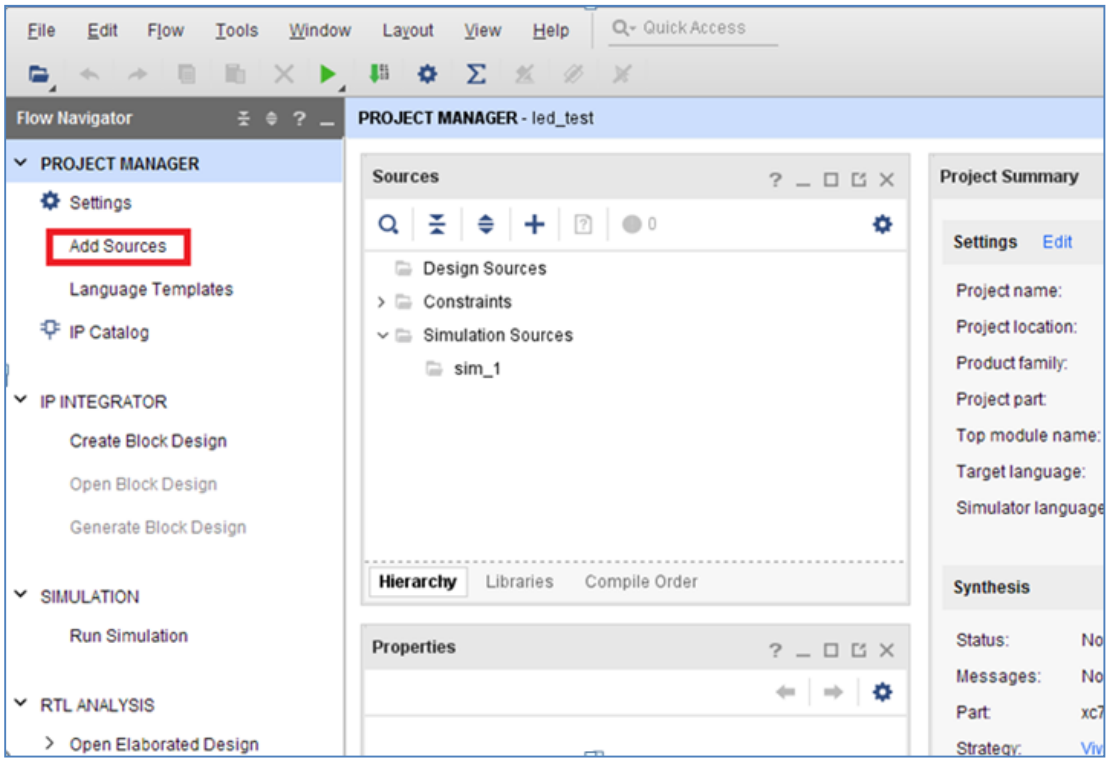
9. 再次确认一下板子型号有没有选对, 没有问题再点击“Finish”完成工程创建。

10. 工程创建后如下图所示:

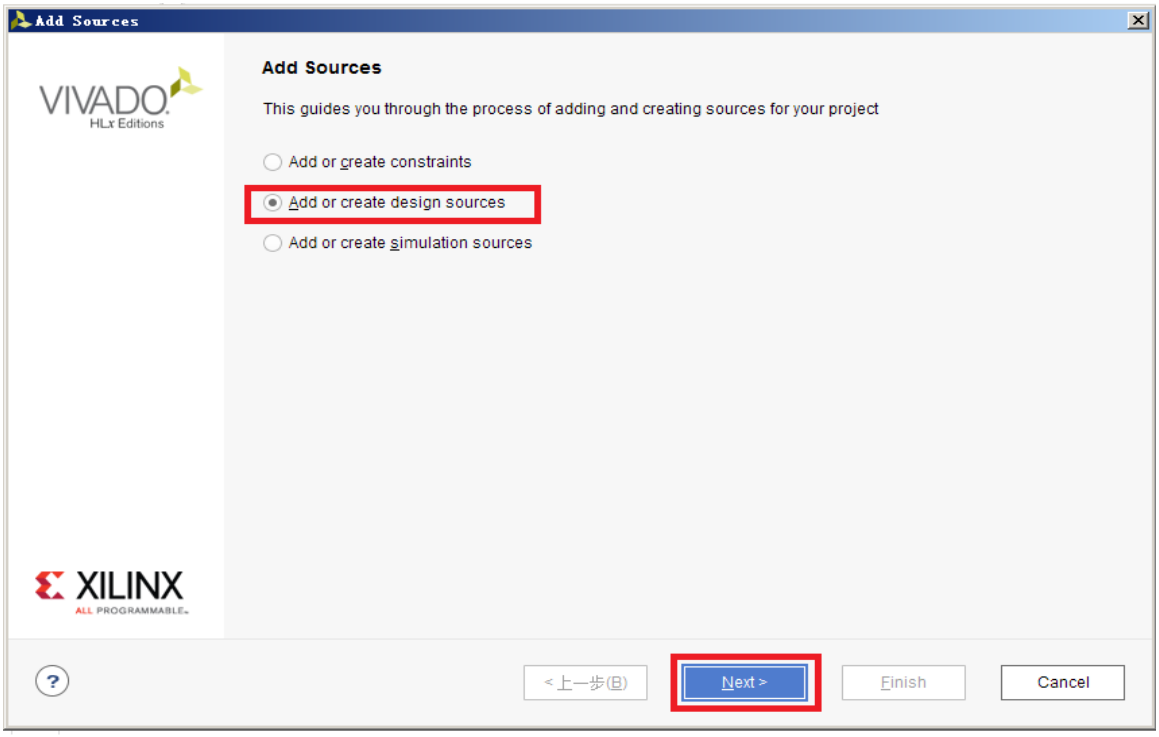


4.2 编写流水灯的 verilog 代码

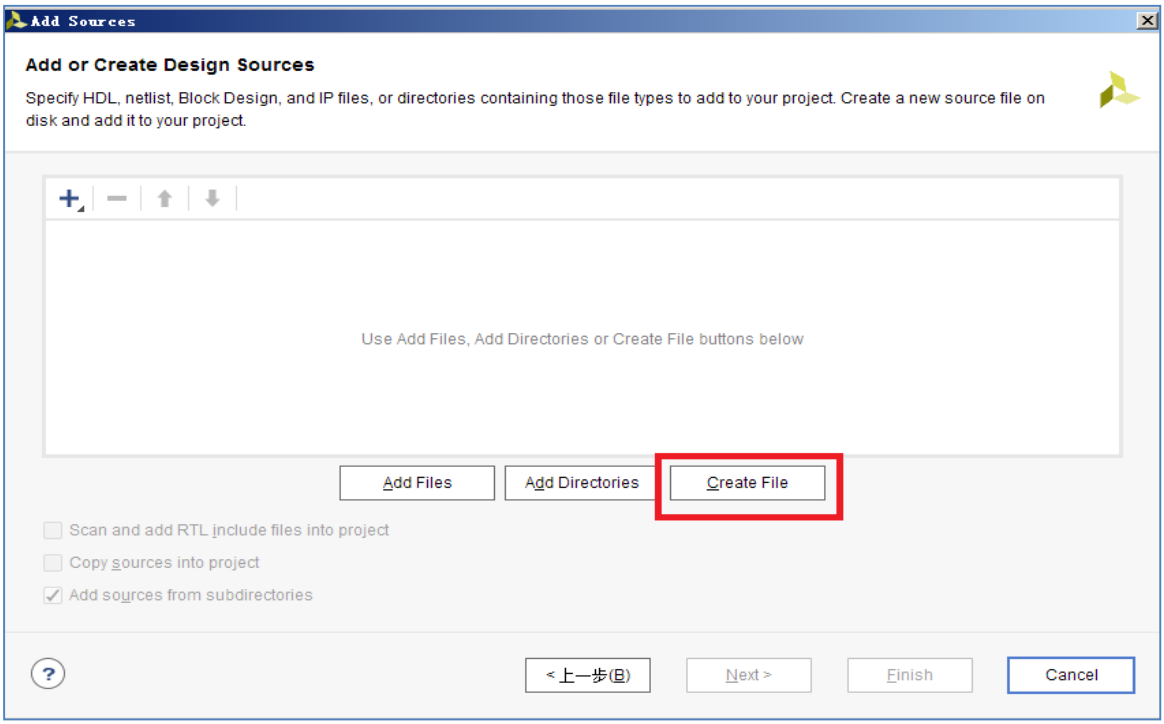
1. 点击 Project Manager 下的 Add Sources 图标（或者使用快捷键 Alt+A）。



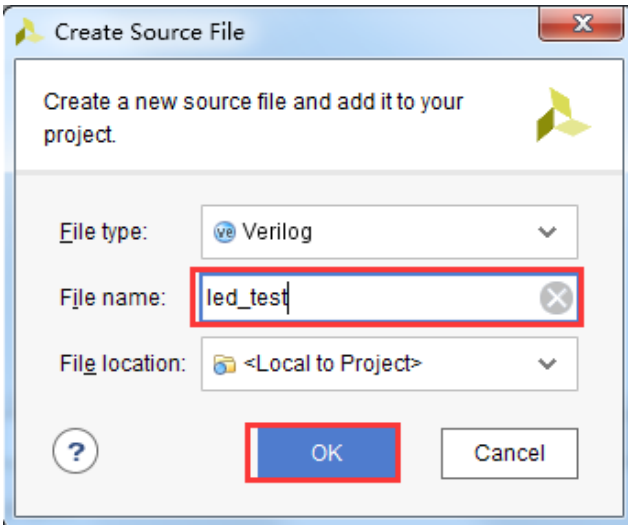
2. 选择 Add or create design sources 选项，点击 Next。



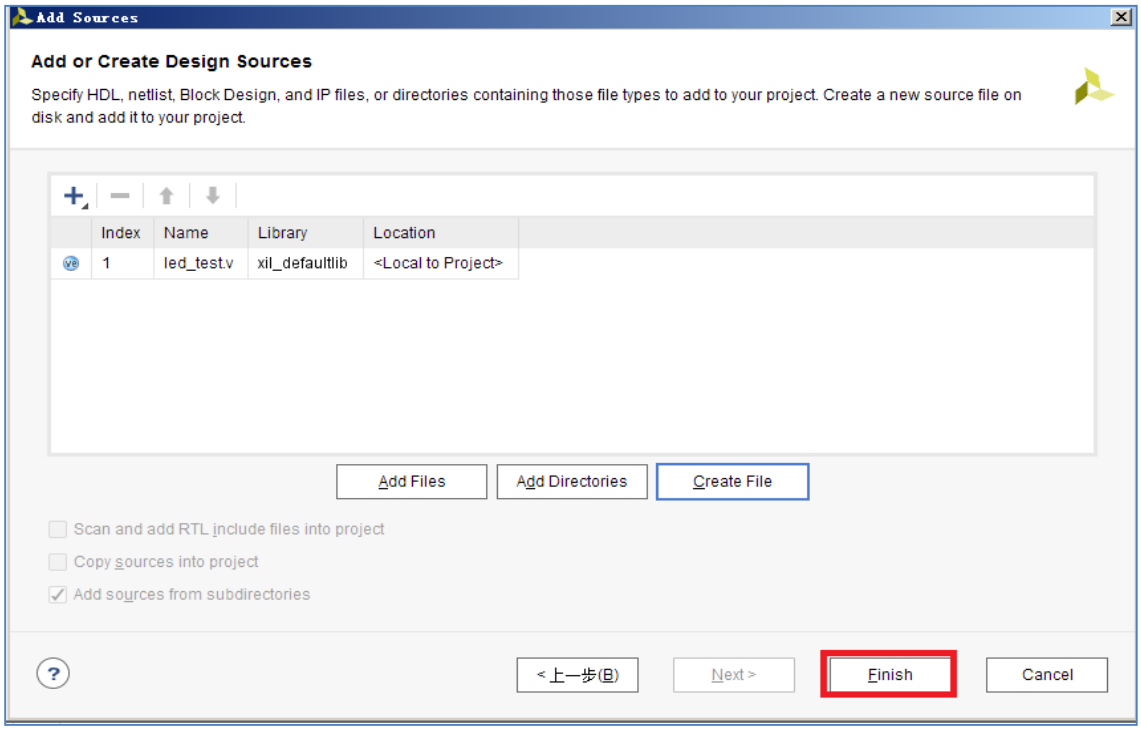
3. 点击 Add Files 可以一个个添加源文件，点击 AddDirectories 可以按目录添加源文件。因为现在我们还没有设计程序，这里要点击 Create File 按钮。



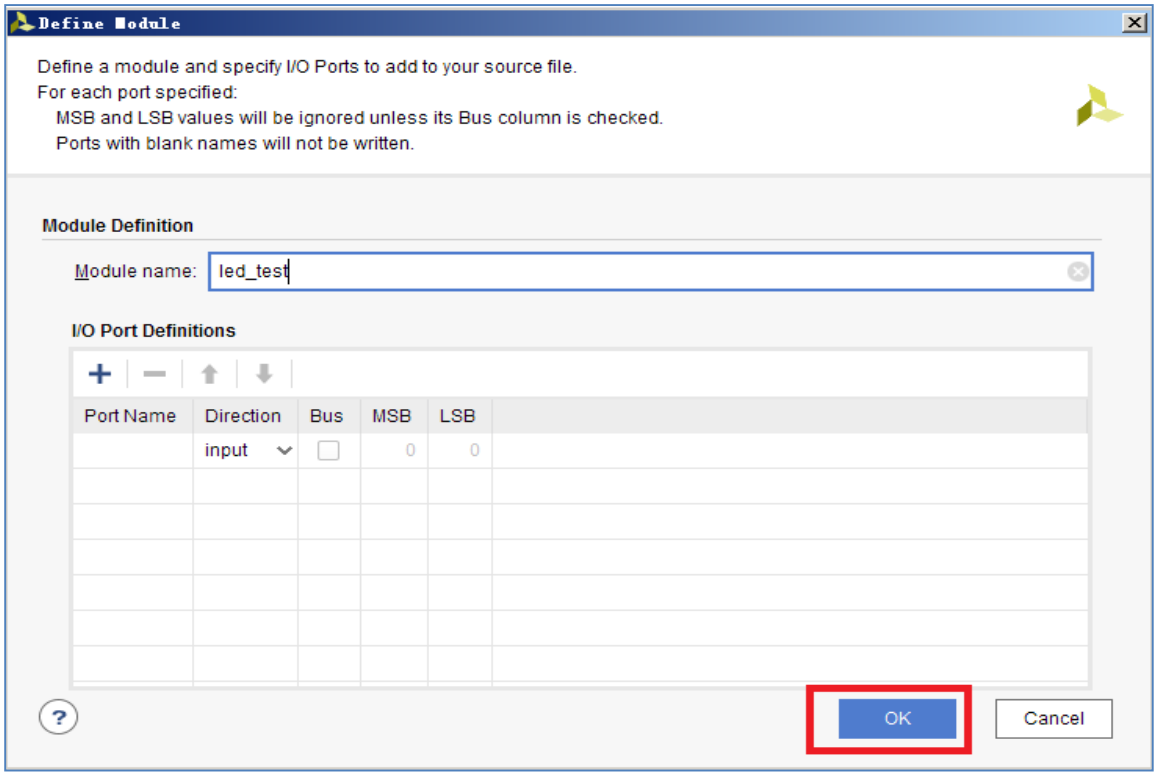
在弹出的对话框里选择 File type 是 verilog, File name 为 led_test, 点击 OK 按钮。

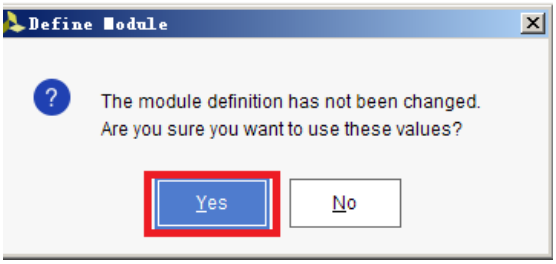


4.点击“Finish”完成。

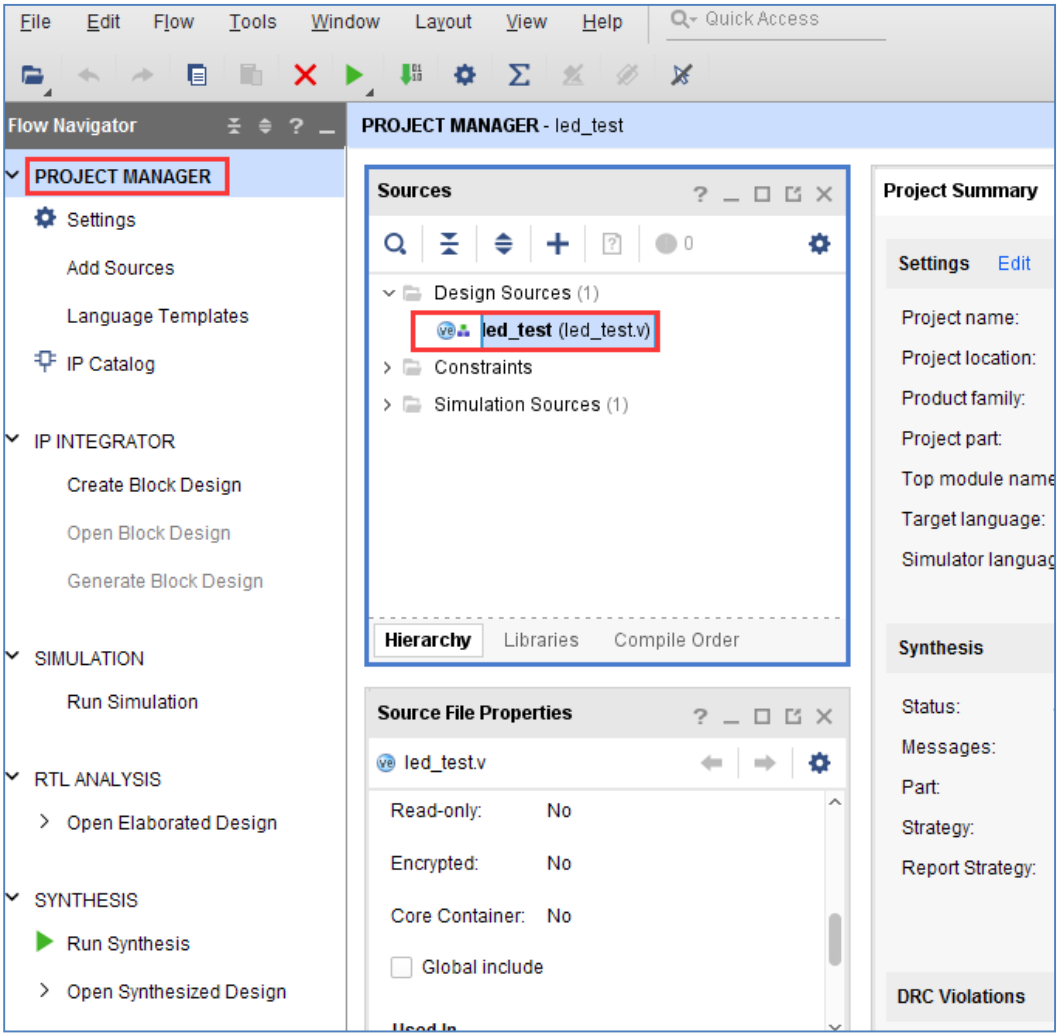


向导会提示您定义 I/O 的端口，这里我们可以不定义，后面自己在程序中编写就可以,单击 OK 完成。





这时在 Project Manager 界面下的 Design Sources 里已经有了一个 led_test.v 文件, 并且自动成为项目的顶层（Top）模块了。



5. 接下去我们来编写 led_test.v 的程序,这里我们定义了一个 32 位的寄存器 timer, 用于循环计数 0~199_999_999(1 秒钟), 当计数到 49_999_999(0.25 秒）的时候，点亮第一个 LED 灯；当计数到 99_999_999(0.5 秒）的时候，点亮第二个 LED 灯；当计数到 149_999_999(0.75 秒）的时候，点亮第三个 LED 灯；当计数到 199_999_999(1 秒）的时候，点亮第四个 LED 灯，计数器再重新计数。具体的操作直接看代码吧。

```
//////////////////////////////////////////////////////////////////////////////////
// Module name: led_test.v
//////////////////////////////////////////////////////////////////////////////////
`timescale 1ns / 1ps

module led_test
(
      input                   sys_clk_p,       // Differentia system clock
200Mhz input on board
      input                   sys_clk_n,
      input                   rst_n,           // reset ,low active
      output reg [1:0]          led,             // LED,use for control the LED
signal on board
      output                  fan_pwm          //fan control
 );
//define the time counter
reg [31:0]   timer;
assign fan_pwm = 1'b0;
//////////////////////////////////////////////////////////////////////////////////
//Differentia system clock to single end clock
//////////////////////////////////////////////////////////////////////////////////
wire        sys_clk;

// IBUFDS: Differential Input Buffer
//         Kintex-7
// Xilinx HDL Language Template, version 2017.4

IBUFDS #(
      .DIFF_TERM("TRUE"),        // Differential Termination
      .IBUF_LOW_PWR("TRUE"),     // Low power="TRUE", Highest
performance="FALSE"
      .IOSTANDARD("DEFAULT")     // Specify the input I/O standard
) u_ibuf_sys_clk (
      .O(sys_clk),  // Buffer output
      .I(sys_clk_p),  // Diff_p buffer input (connect directly to top-level
port)
      .IB(sys_clk_n) // Diff_n buffer input (connect directly to top-level
port)
);
//////////////////////////////////////////////////////////////////////////////////
// cycle counter:from 0 to 1 sec
//////////////////////////////////////////////////////////////////////////////////
  always @(posedge sys_clk or negedge rst_n)
    begin
      if (~rst_n)
          timer <= 32'd0;                       // when the reset signal
valid,time counter clearing
      else if (timer == 32'd199_999_999)     //1 seconds count(200M-
1=199999999)
          timer <= 32'd0;                         //count done,clearing the
time counter
      else
                timer <= timer + 1'b1;             //timer counter = timer
counter + 1
    end

//////////////////////////////////////////////////////////////////////////////////
// LED control
//////////////////////////////////////////////////////////////////////////////////
```

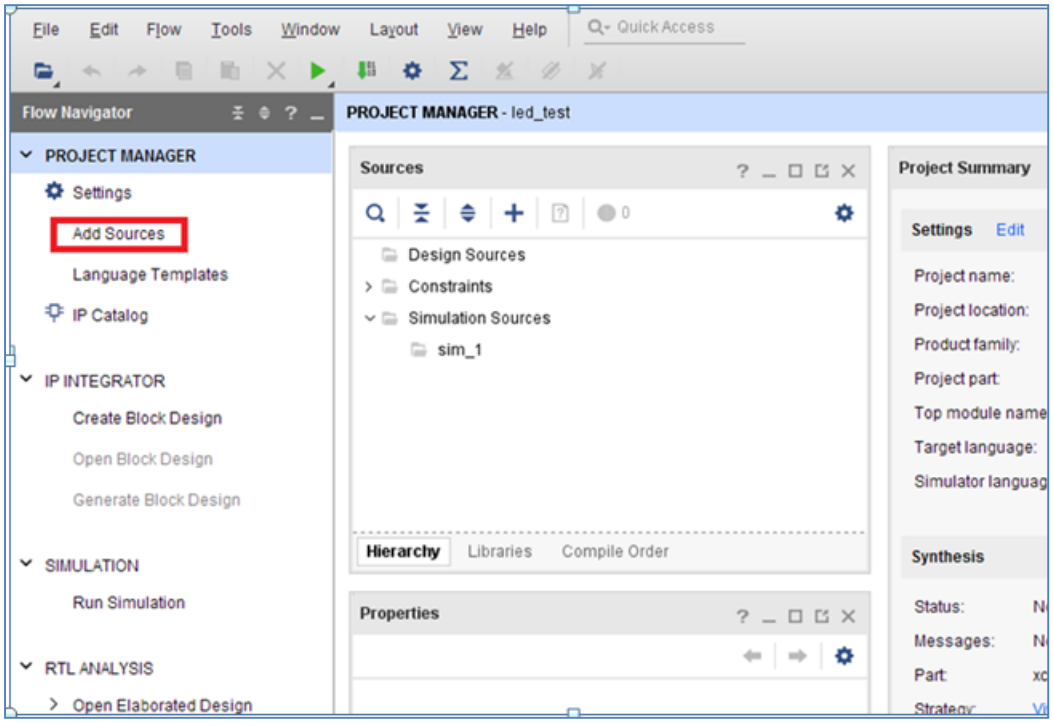
```
  always @(posedge sys_clk or negedge rst_n)
    begin
      if (~rst_n)
          led <= 2'b00;                    //when the reset signal active
      else if (timer == 32'd49_999_999)     //time counter count to 0.25
sec,LED1 lighten
          led <= 2'b01;
      else if (timer == 32'd99_999_999)     //time counter count to 0.5
sec,LED2 lighten
          led <= 2'b10;
      else if (timer == 32'd149_999_999)    //time counter count to 0.75 sec,
          led <= 2'b00;
      else if (timer == 32'd199_999_999)    //time counter count to 1
sec,LED1,LED2 lighten
          led <= 2'b11;
    end

endmodule
```

6. 编写好代码后保存,点击菜单 File -Save All Files。

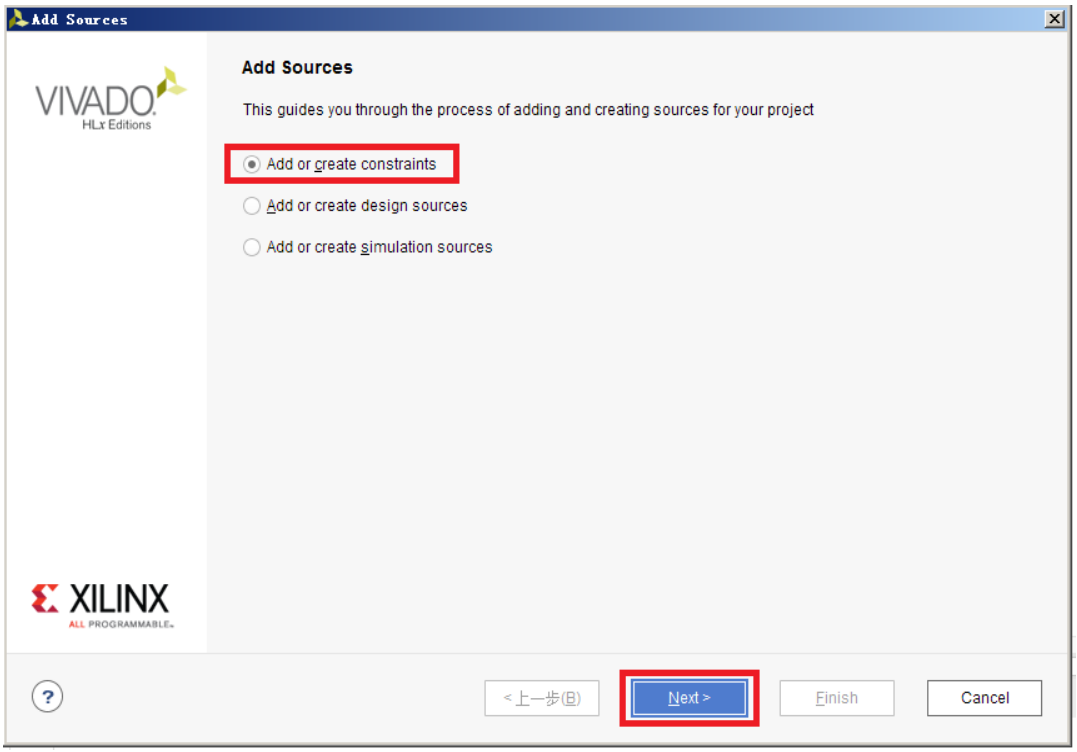
4.3 添加 XDC 管脚约束文件

和 ISE 软件不同， Vivado 使用的约束文件格式为 xdc 文件。xdc 文件里主要是完成管脚的约束,时钟的约束, 以及组的约束。这里我们需要对 led_test.v 程序中的输入输出端口分配到 FPGA 的真实管脚上, 这需要准备一个 FPGA 的引脚绑定文件.xdc 并添加到工程中。

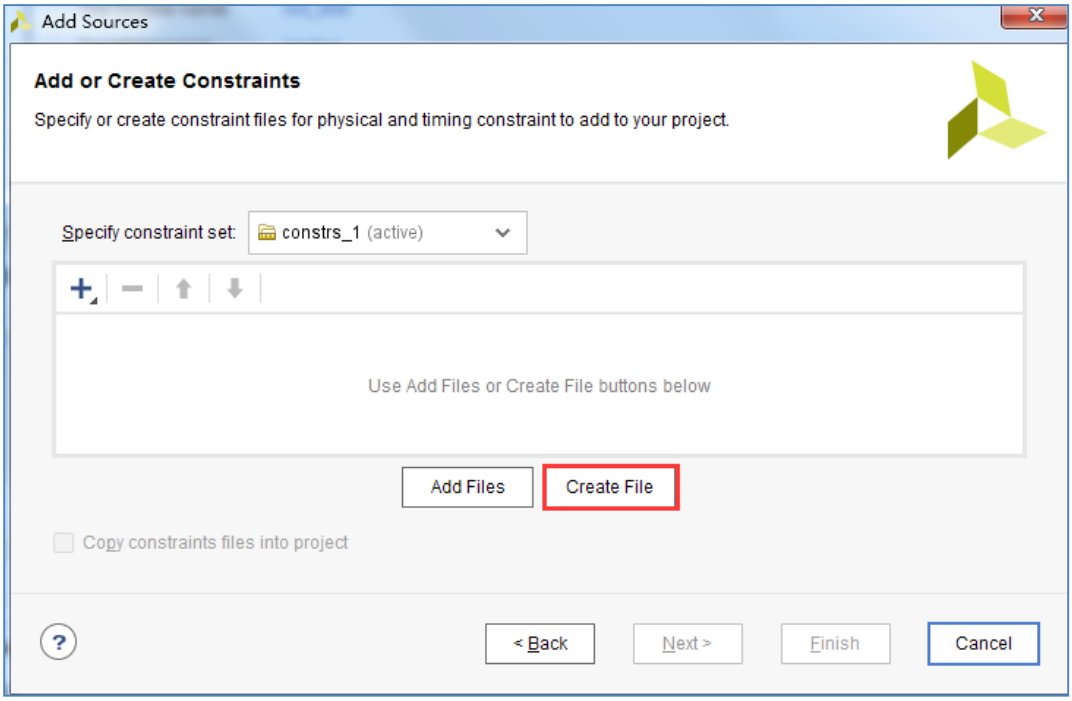
1. 点击 Project Manager 下的 Add Sources 图标。



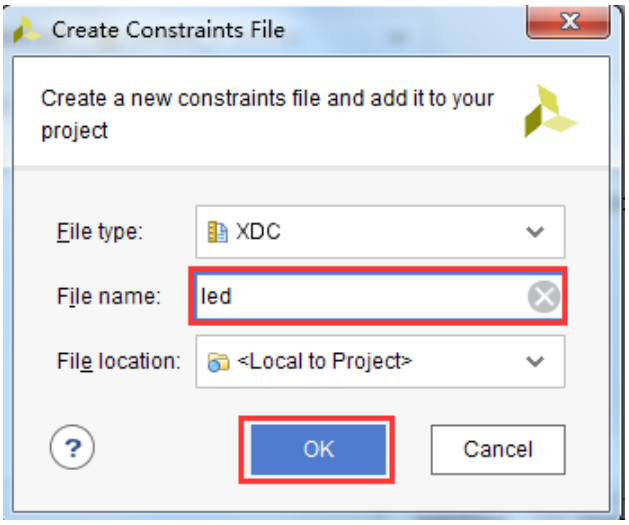
2. 选择 Add or create constraints 选项，点击 Next。



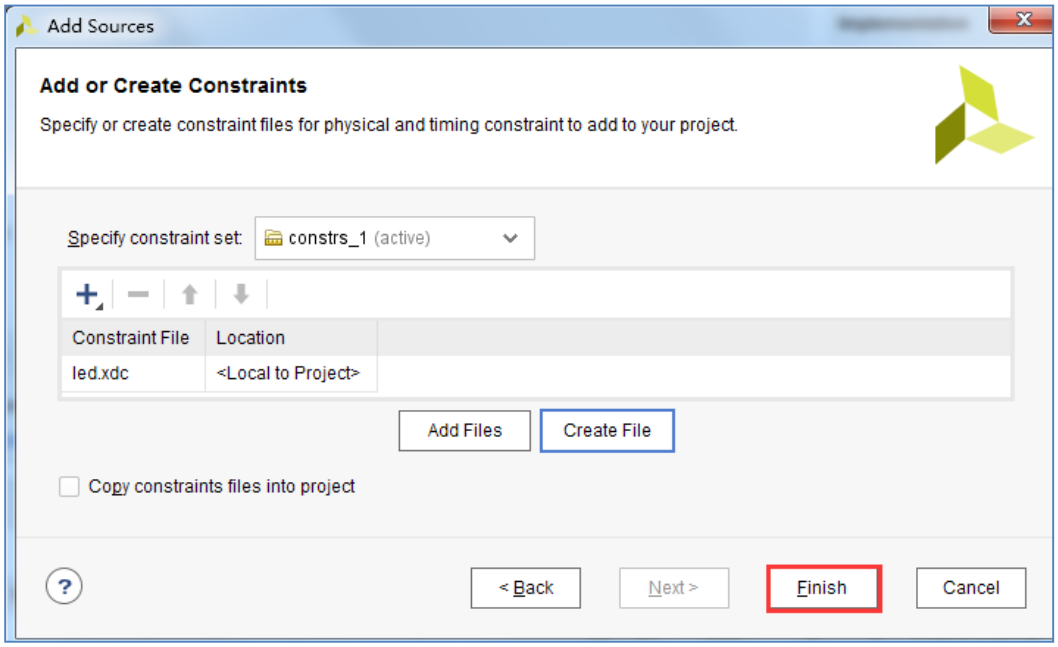
3. 点击 Create File 按钮。



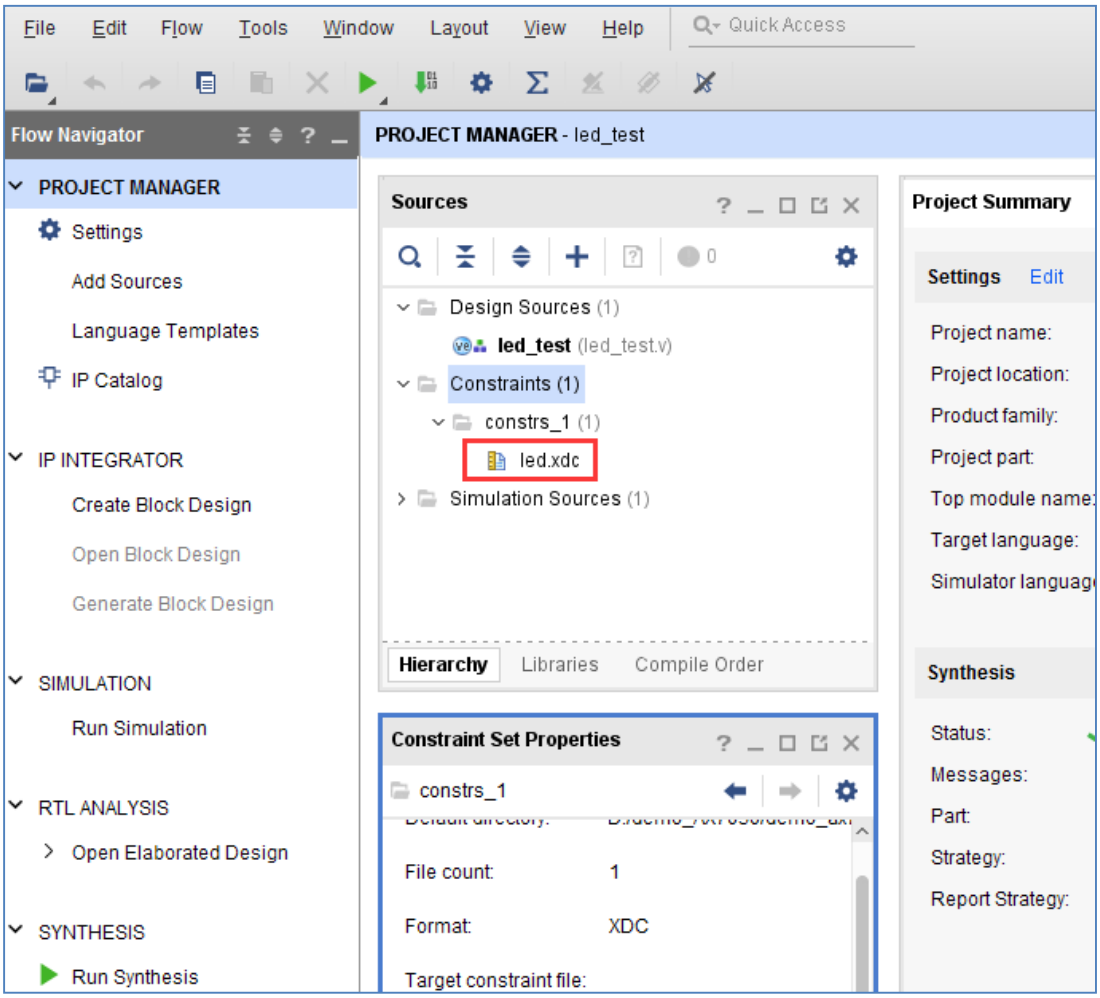
在弹出的对话框里选择 File type 是 XDC, File name 为 led, 点击 OK 按钮。



4.点击“Finish”完成。



这时在 Project Manager 界面下的 Constraints 目录的 constrs_1 目录下已经有了一个 led.xdc 文件。



5. 双击打开这个 led.xdc 文件，在这个文件里添加以下的引脚定义。

```
############### clock define###################

create_clock -period 5.000 [get_ports sys_clk_p]

set_property PACKAGE_PIN AE10 [get_ports sys_clk_p]

set_property IOSTANDARD DIFF_SSTL15 [get_ports sys_clk_p]

############### key define###################

set_property PACKAGE_PIN AD24 [get_ports rst_n]

set_property IOSTANDARD LVCMOS33 [get_ports rst_n]

###################fan define###################

set_property IOSTANDARD LVCMOS33 [get_ports fan_pwm]

set_property PACKAGE_PIN AB30 [get_ports fan_pwm]

###############LED define###################
```

```
set_property PACKAGE_PIN Y28 [get_ports {led[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]

set_property PACKAGE_PIN AA28 [get_ports {led[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]

##############SPI Configurate Setting##################
set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 50 [current_design]
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.UNUSEDPIN Pullup [current_design]
set_property CFGBVS VCCO [current_design]
set_property CONFIG_VOLTAGE 3.3 [current_design]
```

XDC 文件中最后是配置 CFGBVS 管脚的电压和配置电路的电压，因为在开发板上 CFGBVS 管脚是上拉到 3.3V 的，也就是 BANK0 的 VCCIO。另外配置电路的电压是 3.3V。所以这里分别配置成 VCCIO 和 3.3V。

下面来介绍一下最基本的 XDC 编写的语法，普通 IO 口只需约束引脚号和电压，管脚约束如下：

set_property PACKAGE_PIN "引脚编号" [get_ports “端口名称”]

电平信号的约束如下：

set_property IOSTANDARD "电压" [get_ports “端口名称”]

这里需要注意文字的大小写，端口名称是数组的话用{ }刮起来，端口名称必须和源代码中的名字一致，且端口名字不能和关键字一样。

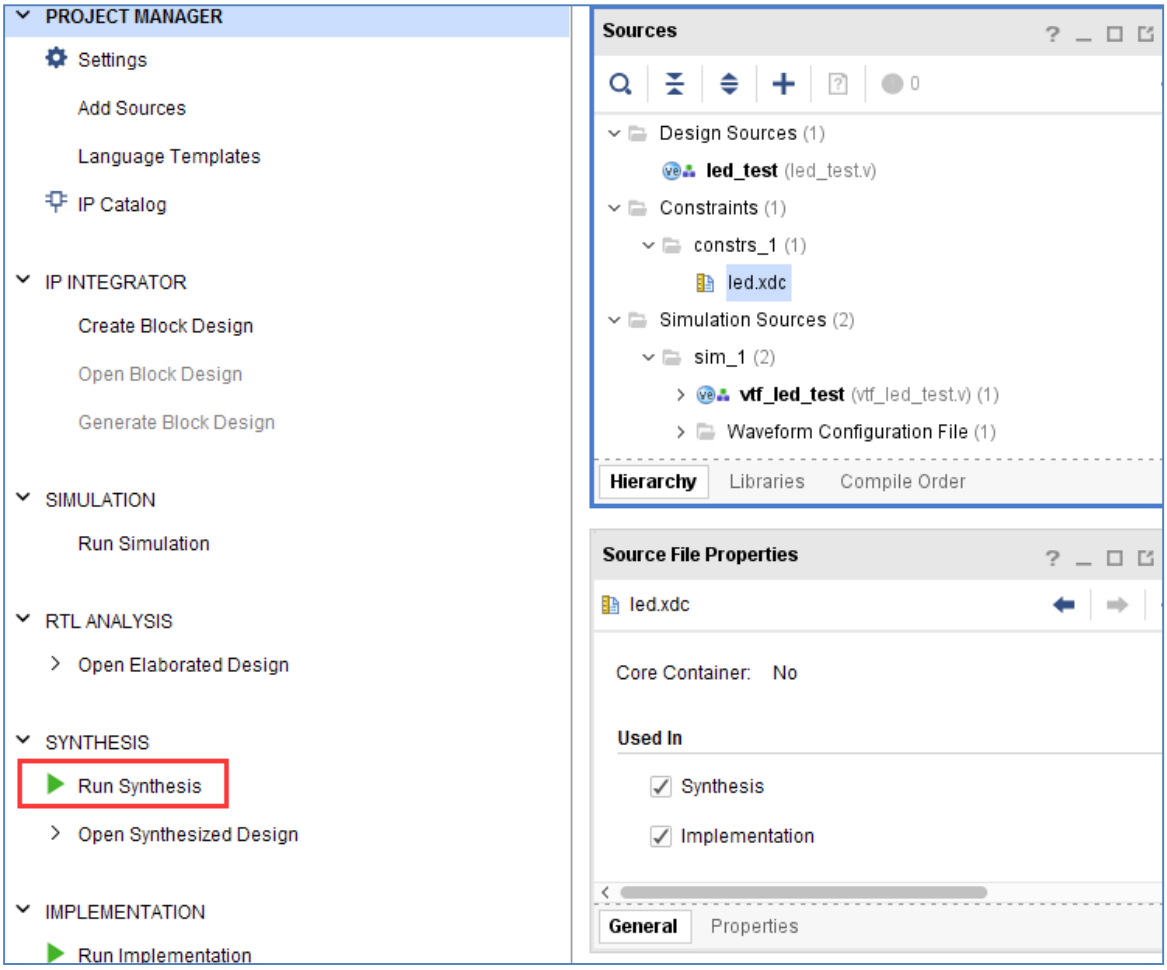
这里时钟输入为差分时钟，接入到 1.5V 的 BANK 中时钟差分输入管脚上，电平标准需要设置成 DIFF_SSTL15。时钟端口还可以定义时钟周期约束，比如我们在 XDC 里面定义了输入的差分时钟的时钟周期为 5ns。 时钟周期的约束方法如下：

create_clock -period "周期" [get_ports “端口名称”]

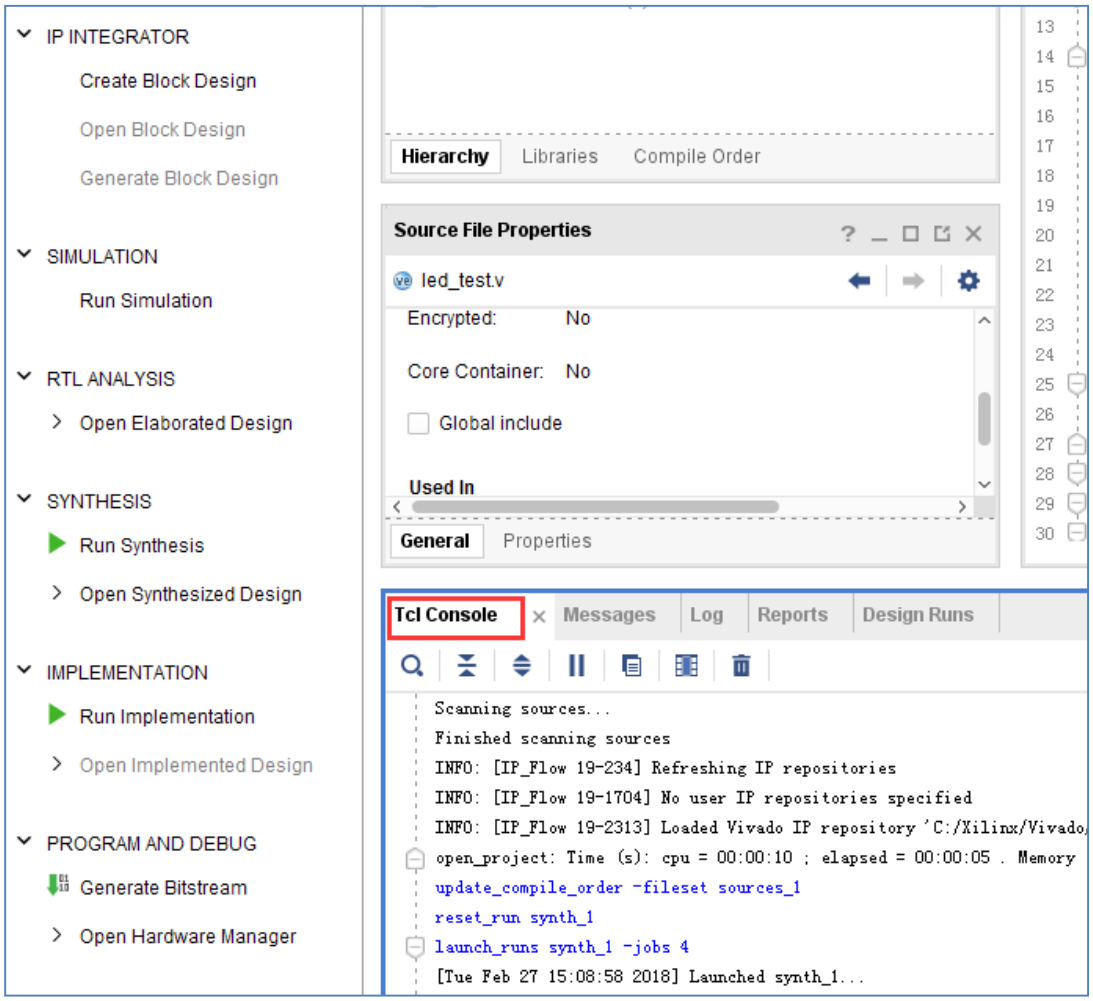
完成后选择菜单 File->Save all files 保存所有文件。

4.4 编译

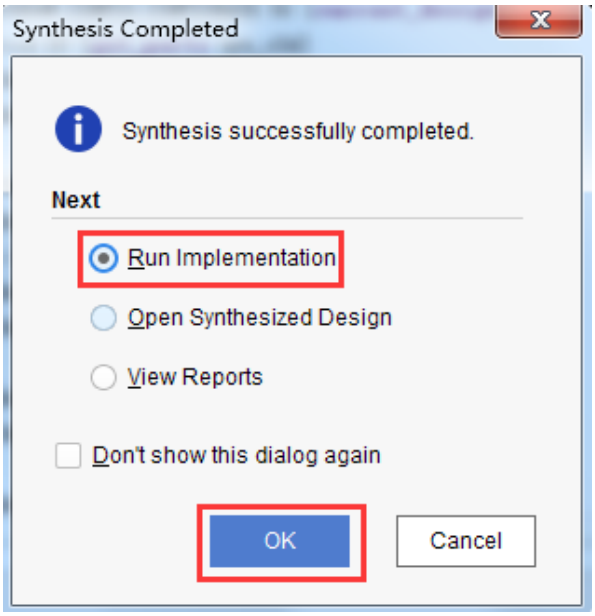
点击 Run Synthesis，即可开始综合并生成网表文件：



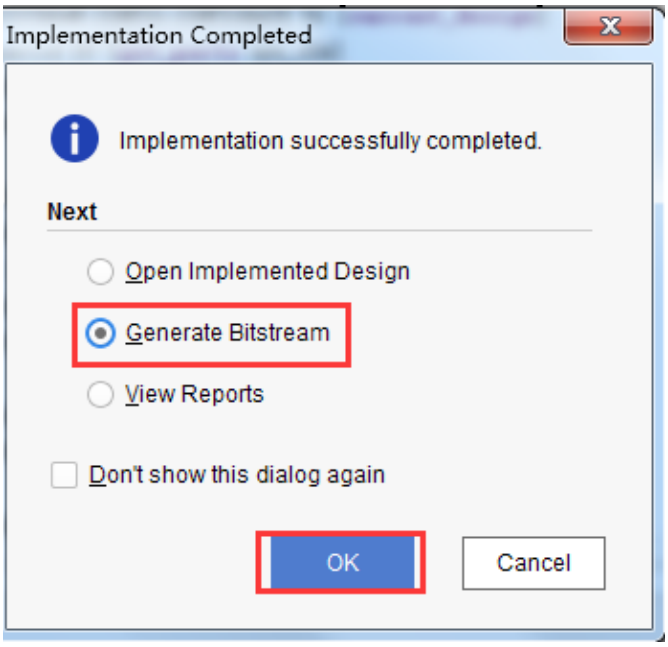
在 Tcl Console 窗口或者 Messages 窗口可以看到一些状态信息。



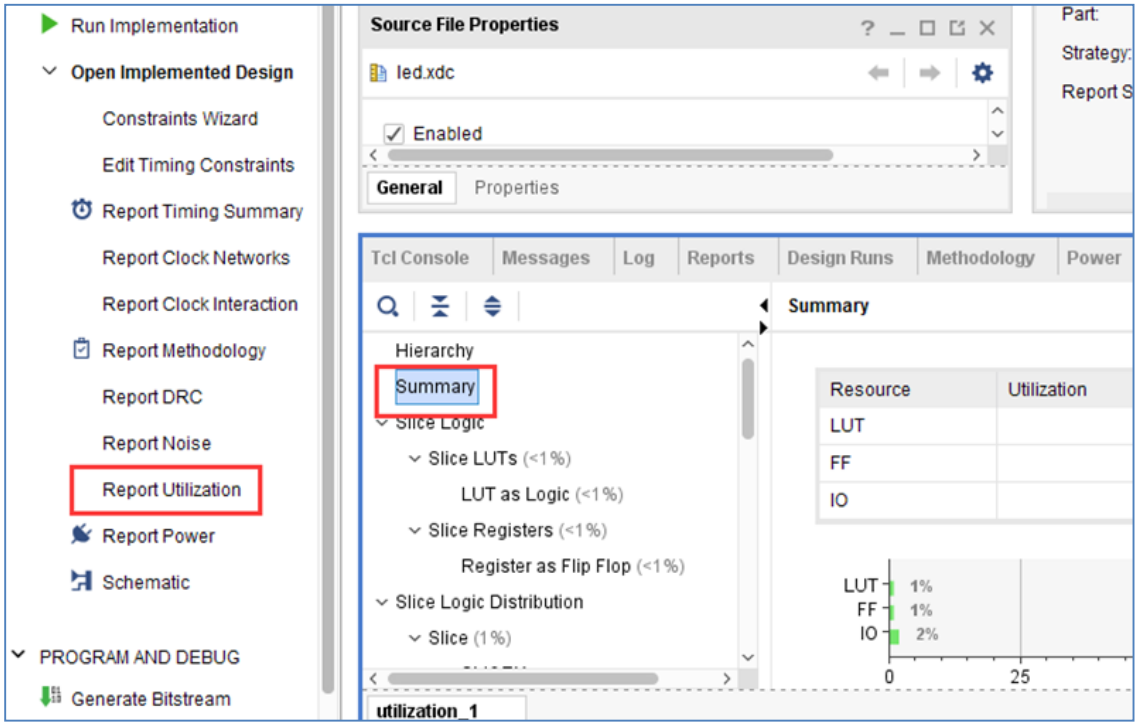
综合完成后，会弹出这样一个提示小窗口。可以点这里的 Run Implementation 来开始布局布线：



布线完成后会弹出这样一个提示小窗口。可以点这里的 Generate Bitstream 即可生成 bit 文件。



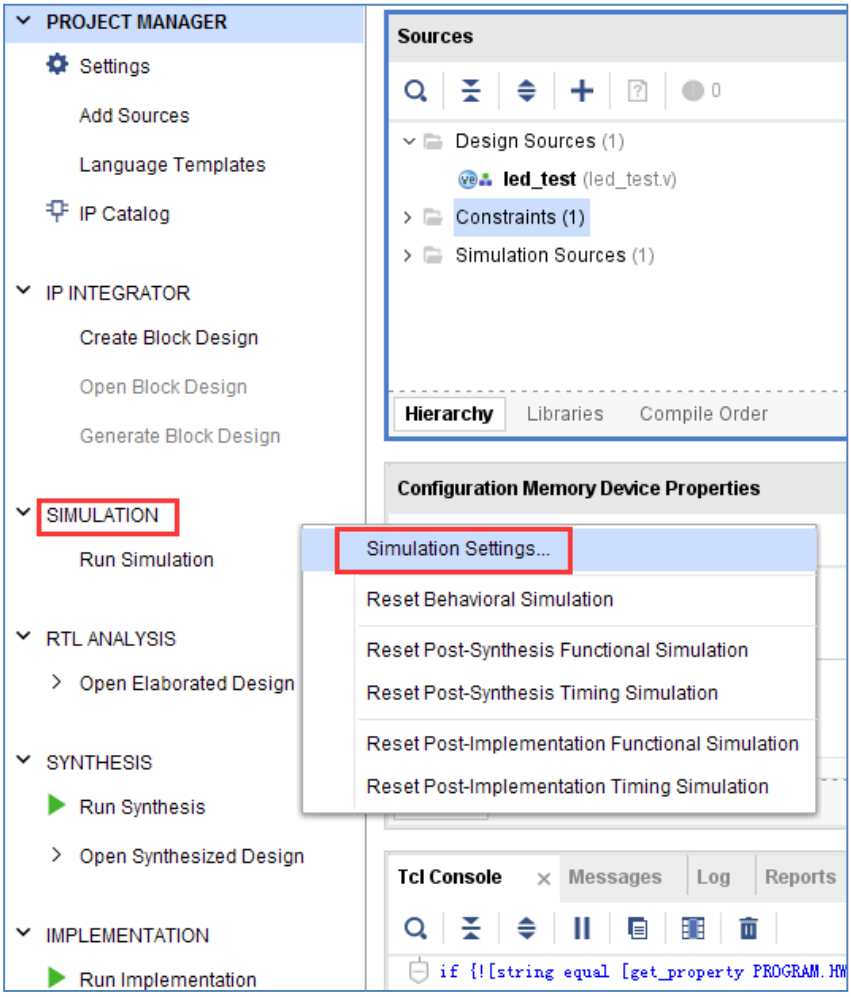
Bit 文件生成完成后，我们可以打开 Project Summary 页面的 Table 来查看板子上实际资源的使用情况,因为我们这里的 led_test 程序比较简单，只用到了四个资源: LUT(查找表），FF(Flip Flop 寄存器）， IO（管脚）和 BUFG(时钟 Buffer)。



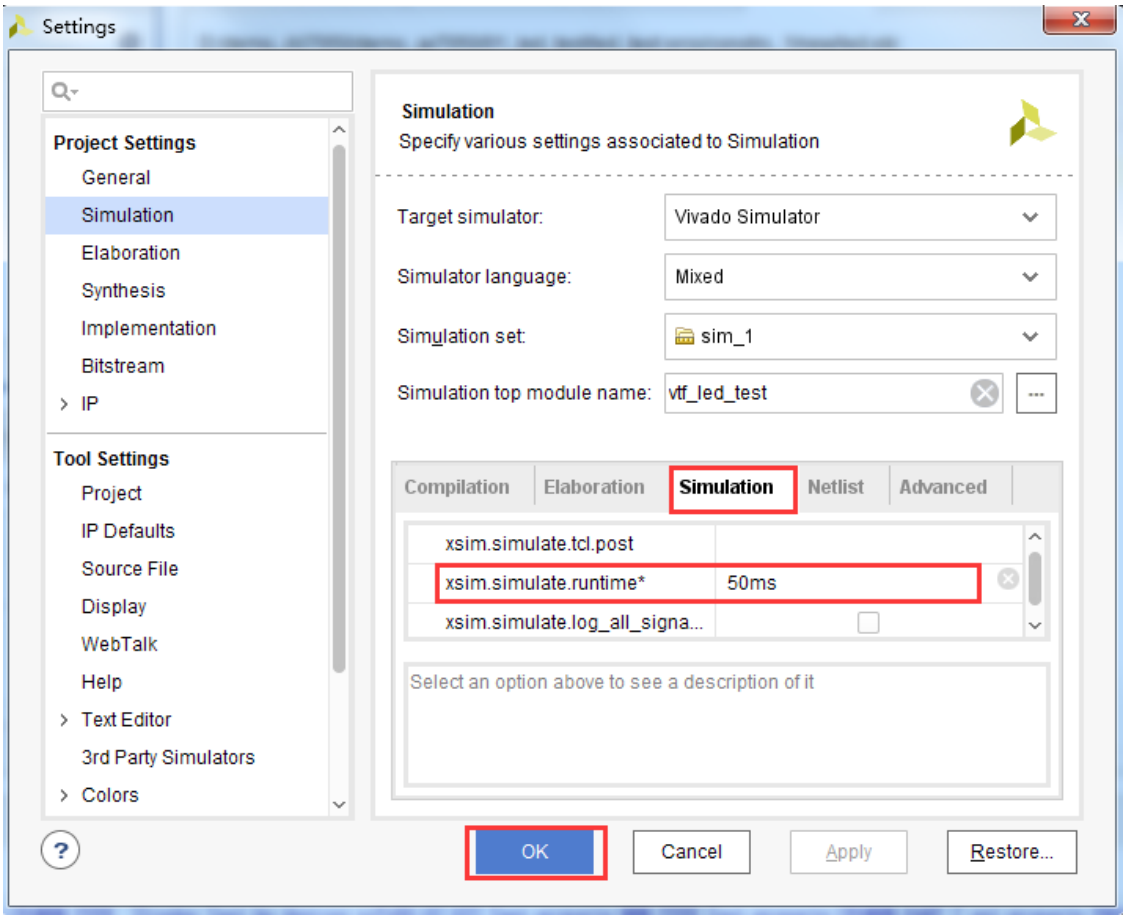
4.5 Vivado 仿真验证

接下来我们不妨小试牛刀，让仿真工具 Vivado 来输出波形验证流水灯程序设计结果和我们的预想是否一致。具体步骤如下：

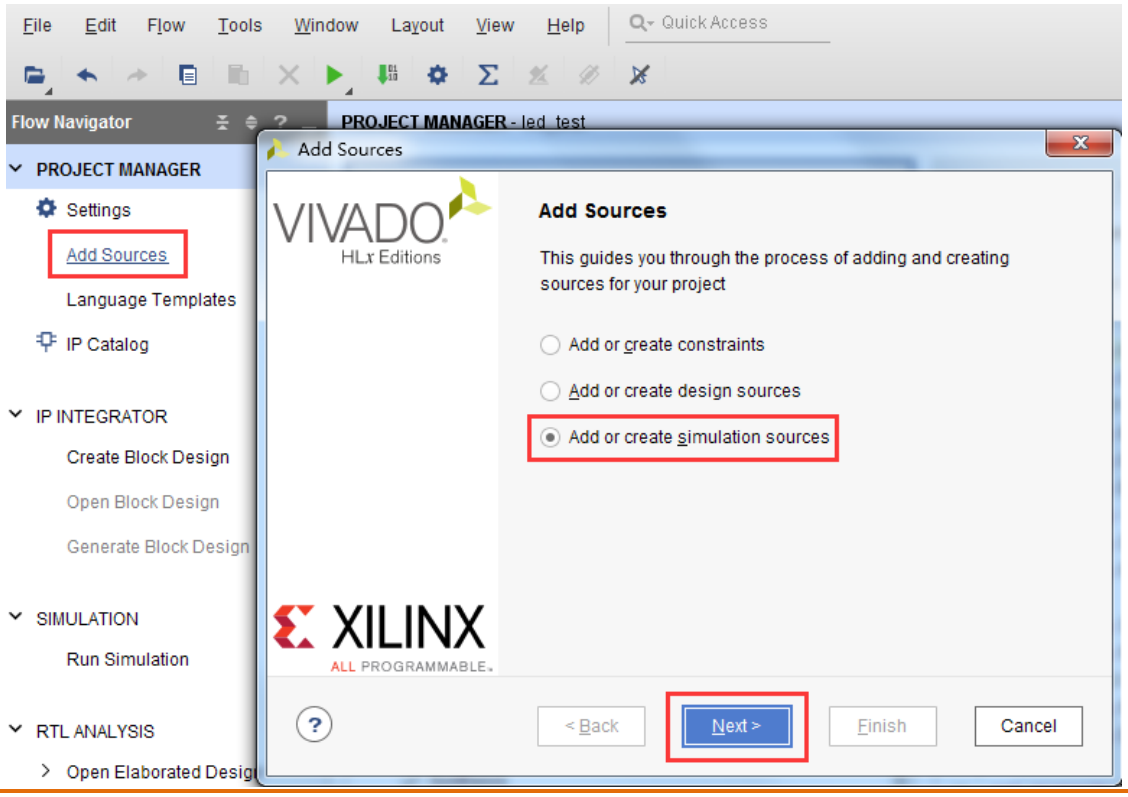
1. 设置 Vivado 的仿真配置，右击 SIMULATION 中 Simulation Settings。



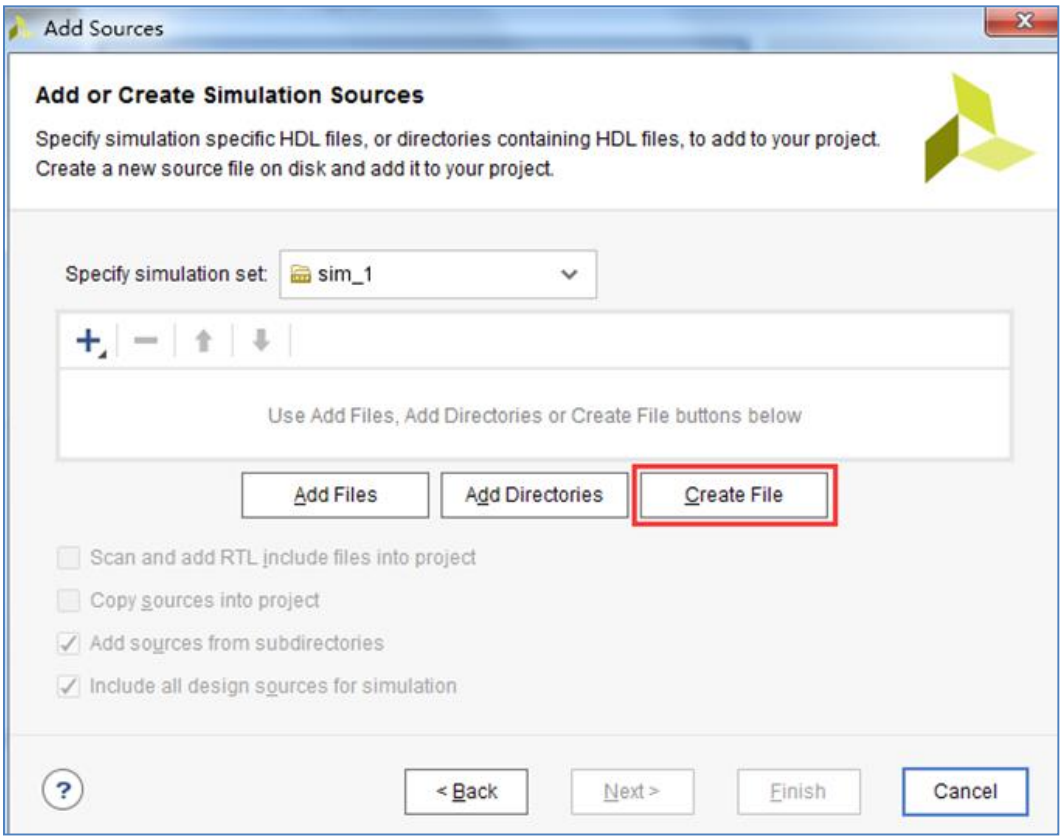
2. 在 Simulation Settings 窗口中进行如下图来配置，这里设置成 50ms（根据需要自行设定）,其它按默认设置，单击 OK 完成。



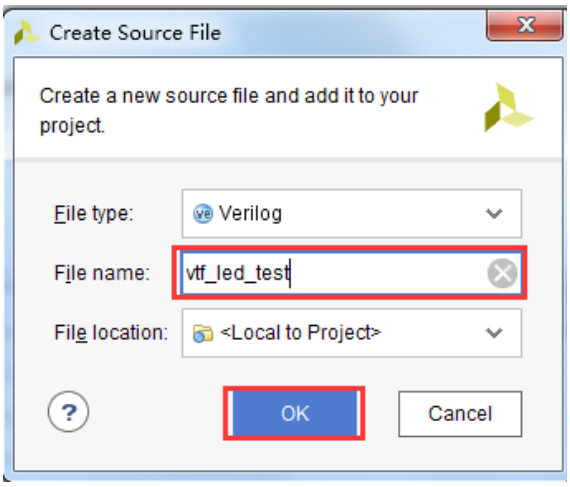
3. 添加激励测试文件，点击 Project Manager 下的 Add Sources 图标,按下图设置后单击 Next。



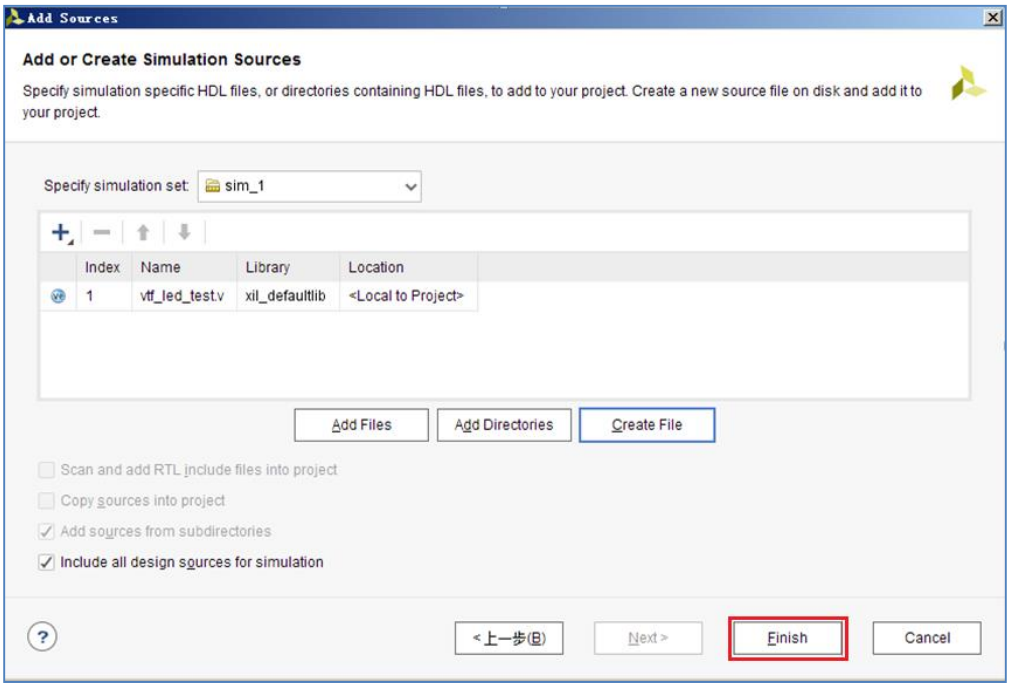
4. 点击 Create File 生成仿真激励文件。



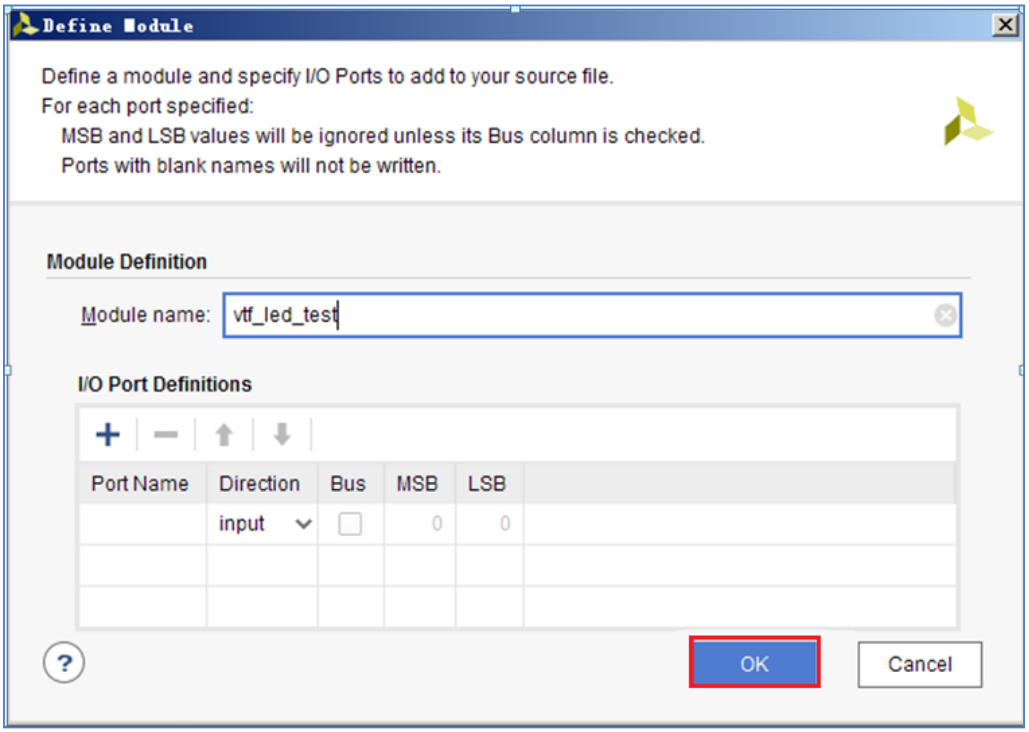
在弹出的对话框中输入激励文件的名字，这里我们输入名为 vtf_led_test。

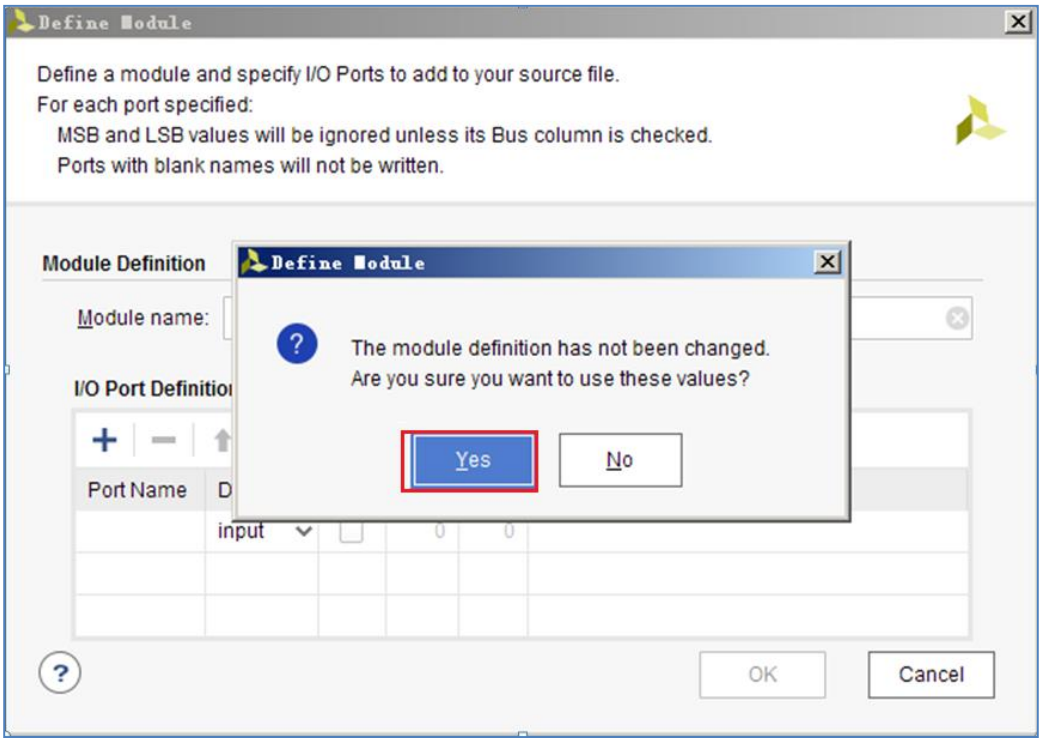


6. 点击 Finish 按钮返回。

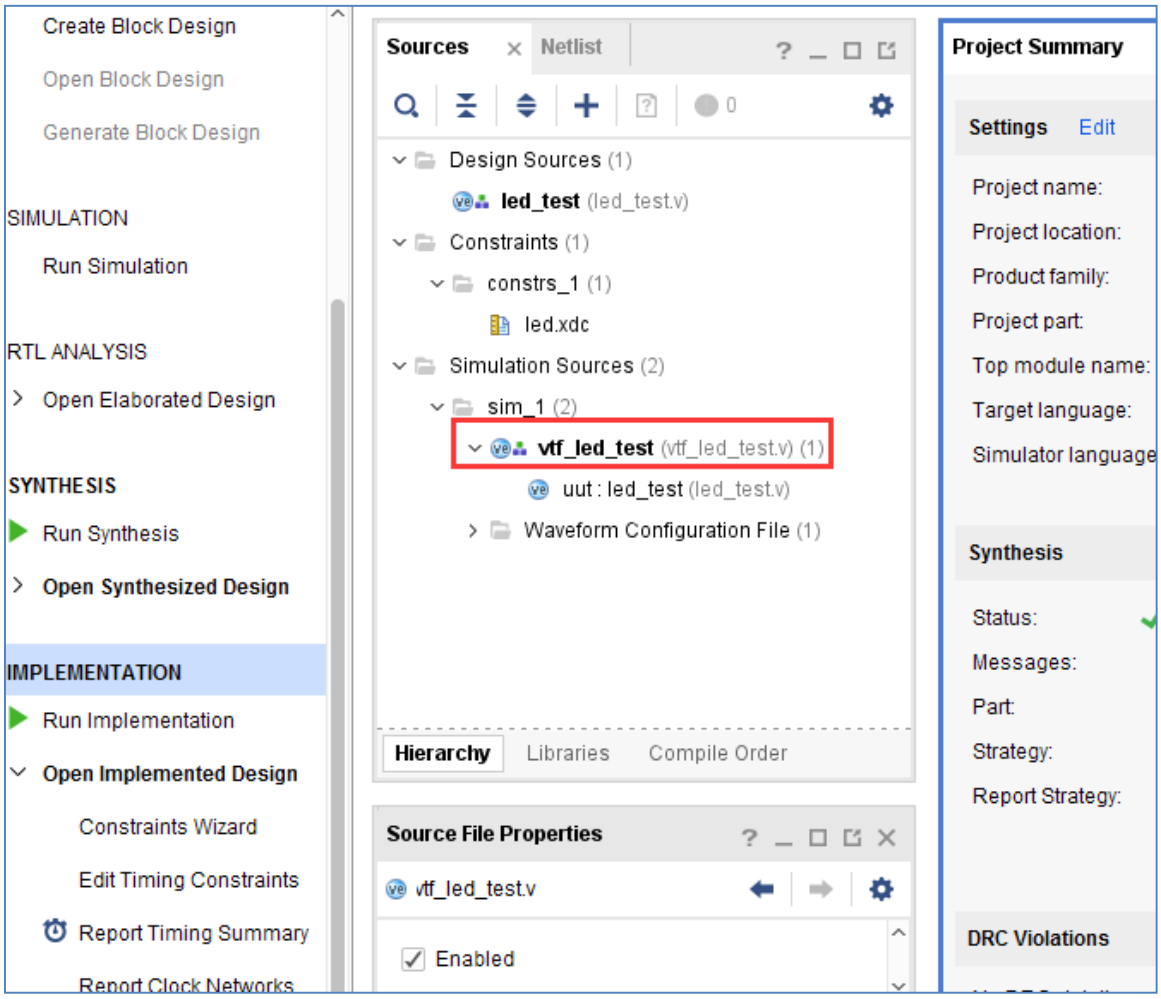


这里我们先不添加 IO Ports，点击 OK。





在 Simulation Sources 目录下多了一个刚才添加的 vtf_led_test 文件。双击打开这个文件，可以看到里面只有 module 名的定义，其它都没有。

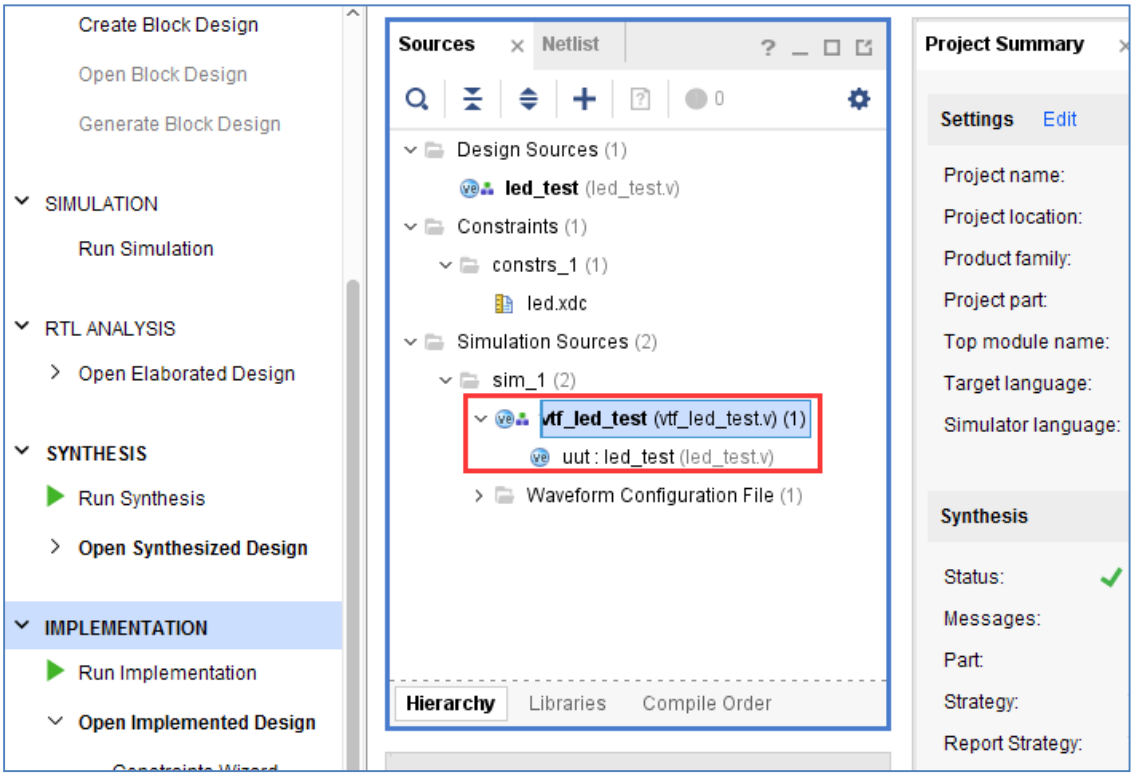


7. 接下去我们需要编写这个 vtf_led_test.v 文件的内容。首先定义输入和输出信号，然后需要实例化 led_test 模块，让 led_test 程序作为本测试程序的一部分。再添加复位和时钟的激励。完成后的 vtf_led_test.v 文件如下：

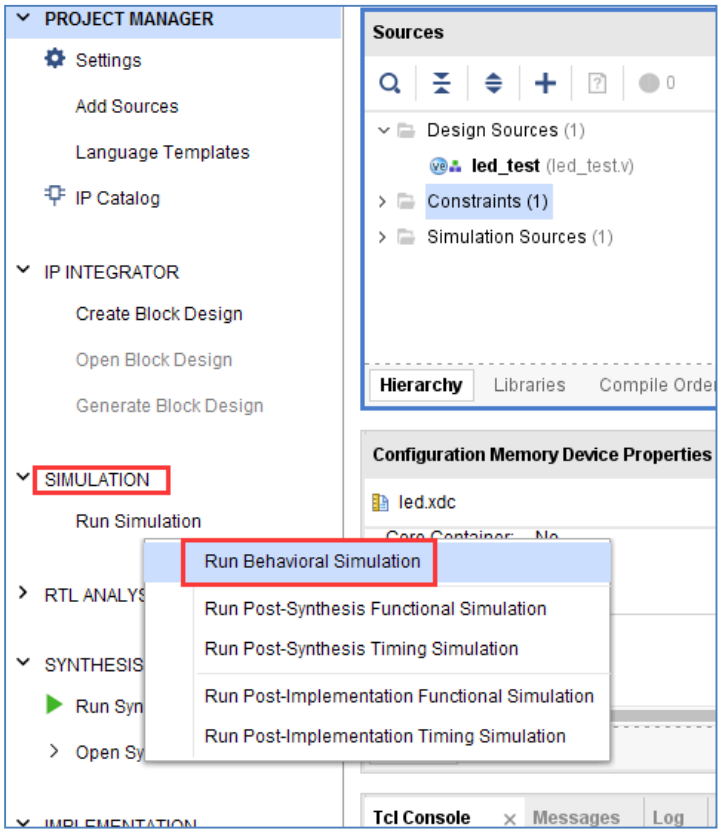
```
`timescale 100ps / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////
// Module Name: vtf_led_test
//////////////////////////////////////////////////////////////////////////////////
//////

module vtf_led_test;
	// Inputs
	reg sys_clk_p;
	wire sys_clk_n;
	reg rst_n;

	// Outputs
	wire [1:0] led;

	// Instantiate the Unit Under Test (UUT)
	led_test uut (
		.sys_clk_p(sys_clk_p),
```

```
            .sys_clk_n(sys_clk_n),
            .rst_n(rst_n),
            .led(led)
      );

      initial begin
            // Initialize Inputs
            sys_clk_p = 0;
            rst_n = 0;

            // Wait 100 ns for global reset to finish
            #1000;
          rst_n = 1;
            // Add stimulus here
            #20000;
      //   $stop;
       end

    always #25 sys_clk_p = ~ sys_clk_p;    //5ns一个周期，产生200MHz时钟源
    assign sys_clk_n=~sys_clk_p;

endmodule
```

8. 编写好后保存，vtf_led_test.v 自动成了这个仿真 Hierarchy 的顶层了，它下面是设计文件 led_test.v。

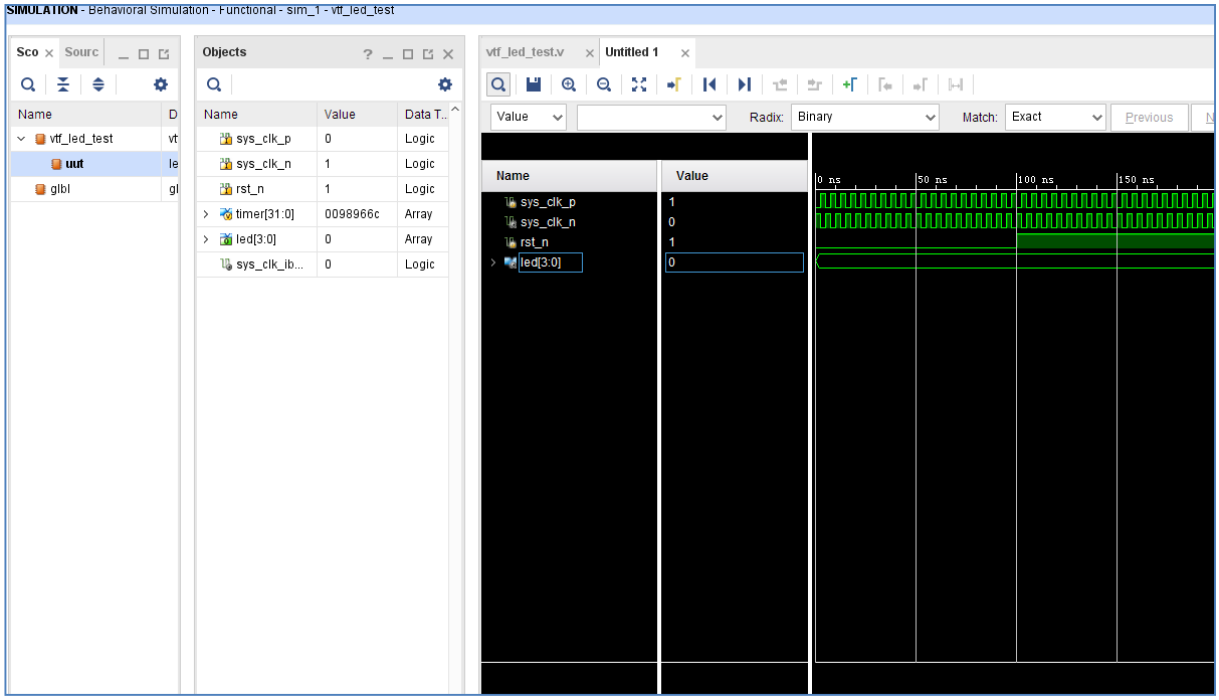


9. 点击 Run Simulation 按钮，再选择 Run Behavioral Simulation。这里我们做一下行为级的仿真就可以了。



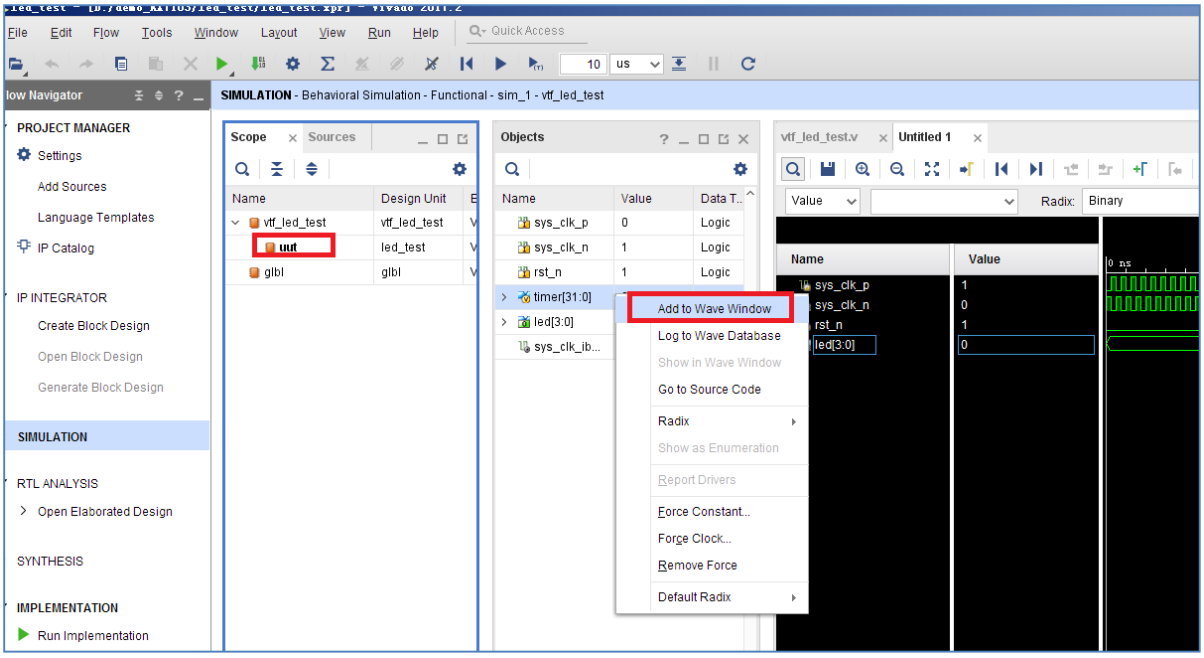
如果没有错误，Vivado 中的仿真软件开始工作了。

10. 在弹出仿真界面后如下图，界面是仿真软件自动运行到仿真设置的 50ms 的波形。

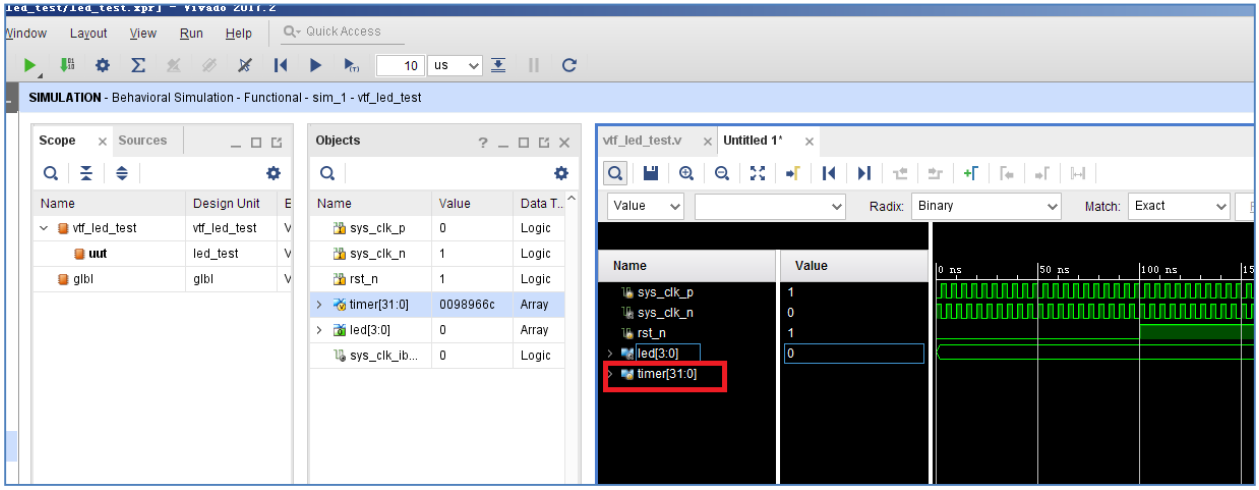


由于 LED 在程序中设计的状态变化时间长，而仿真又比较耗时，在这里观测 timer[31:0]计数器变

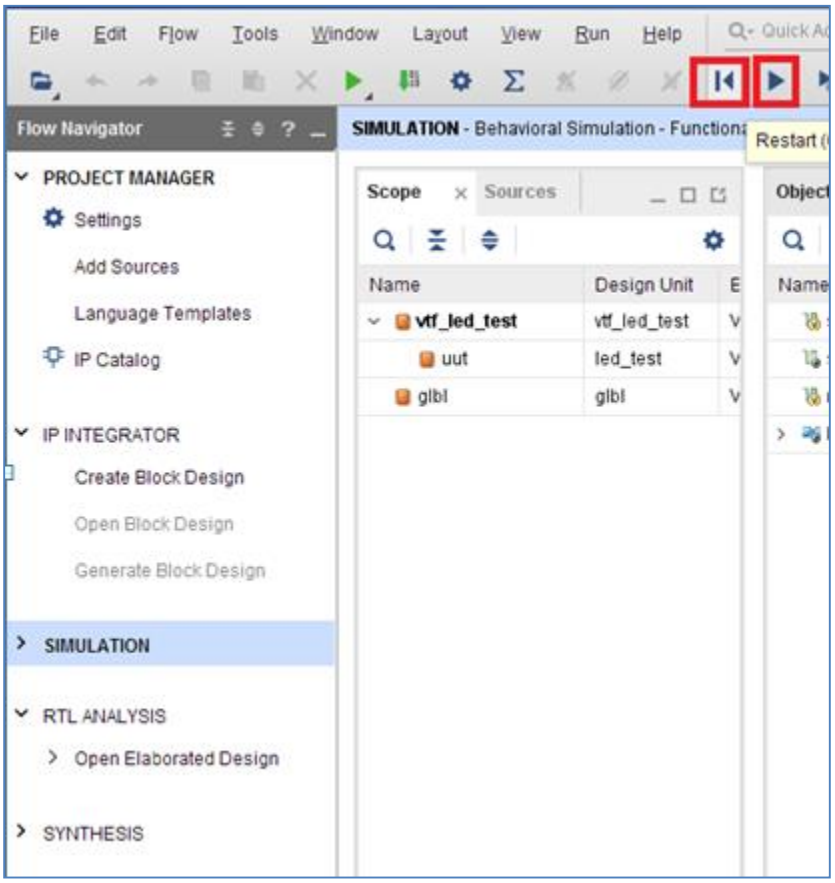
化。把它放到 Wave 中观察(点击 Scope 界面下的 uut， 再右键选择 Objects 界面下的 sys_clk,和 timer， 在弹出的下拉菜单里选择 Add Wave Window)。

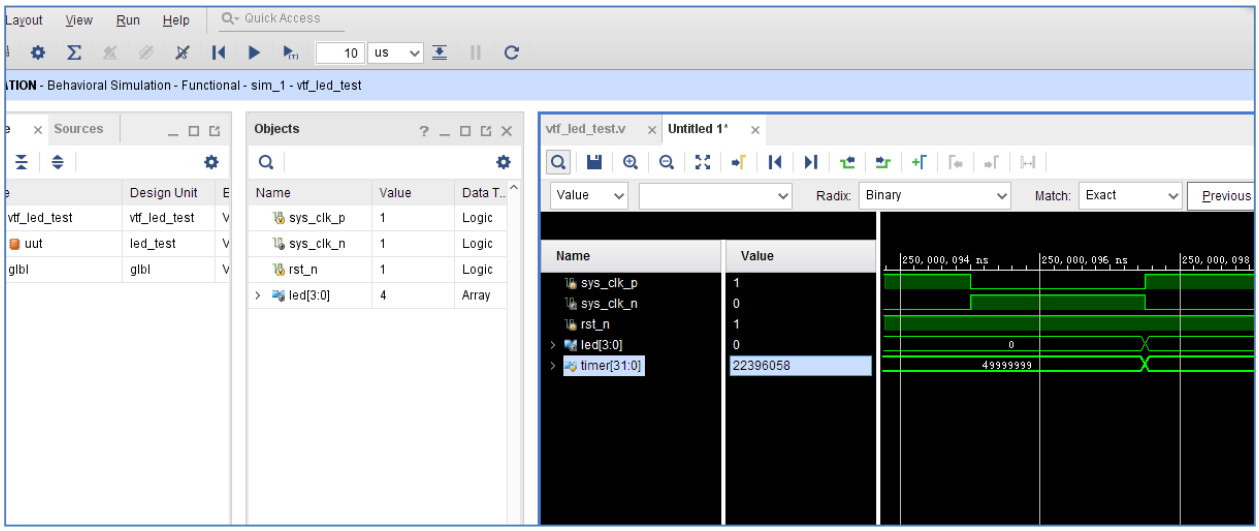


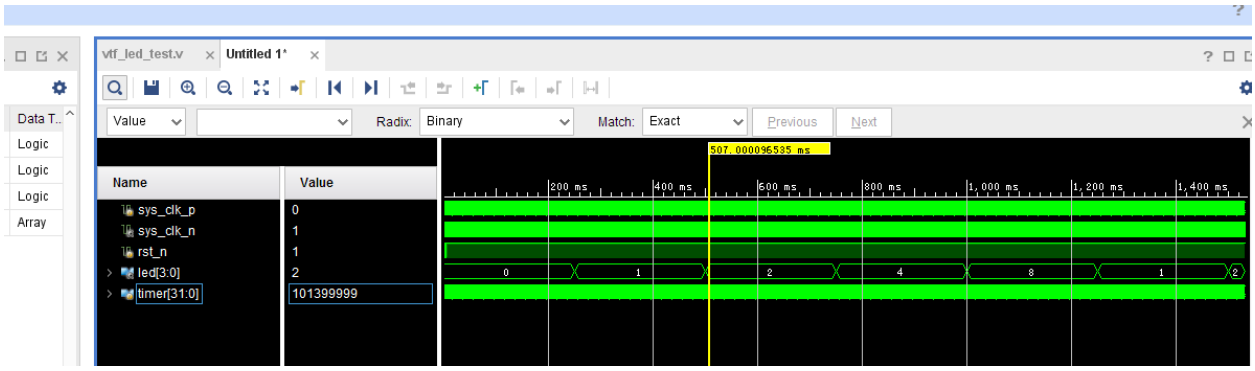
添加后 timer 显示在 Wave 的波形界面上，如下图所示。



11. 点击 Restart 按钮复位一下，再点击 Run All 按钮。（需要耐心！！！），可以看到仿真波形与设计相符。


File Edit Flow Tools Window Layout View Run Help
Flow Navigator
PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
RTL ANALYSIS
Open Elaborated Design
SYNTHESIS
SIMULATION - Behavioral Simulation - Function
Restart
Scope
Sources
Name
Design Unit
vtf_led_test
uut
led_test
glbl


Layout View Run Help
Quick Access
TION - Behavioral Simulation - Functional - sim_1 - vtf_led_test
Sources
Objects
Name
Value
Data T..
sys_clk_p
sys_clk_n
rst_n
led[3:0]
Logic
Array
vtf_led_test.v
Untitled 1*
Radix: Binary
Match: Exact
Previous
timer[31:0]
22396058
49999999

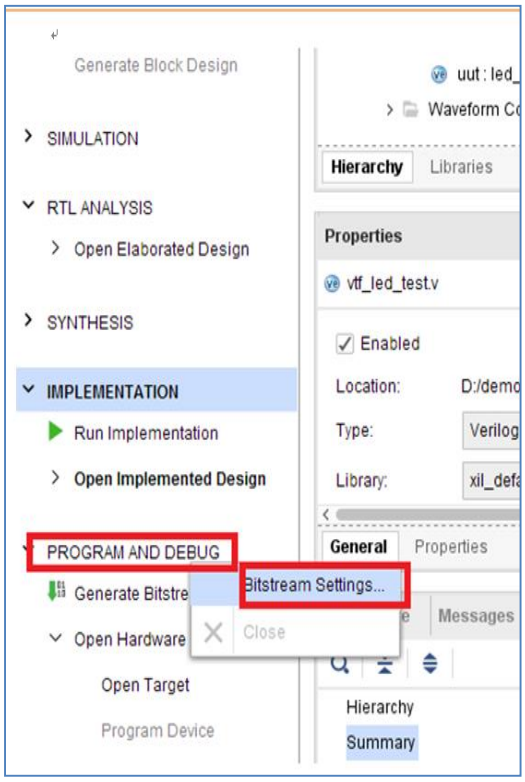


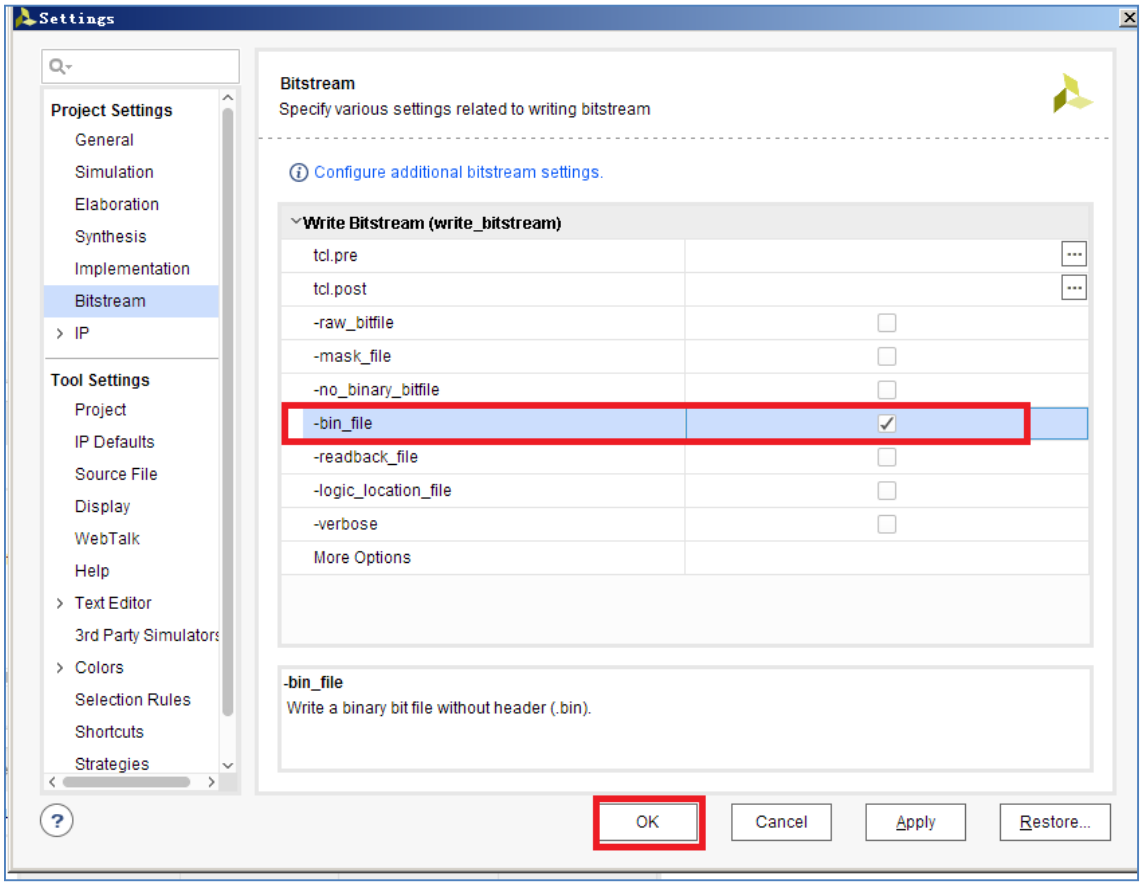
我们可以看到 led 的信号变化。

4.6 下载和调试

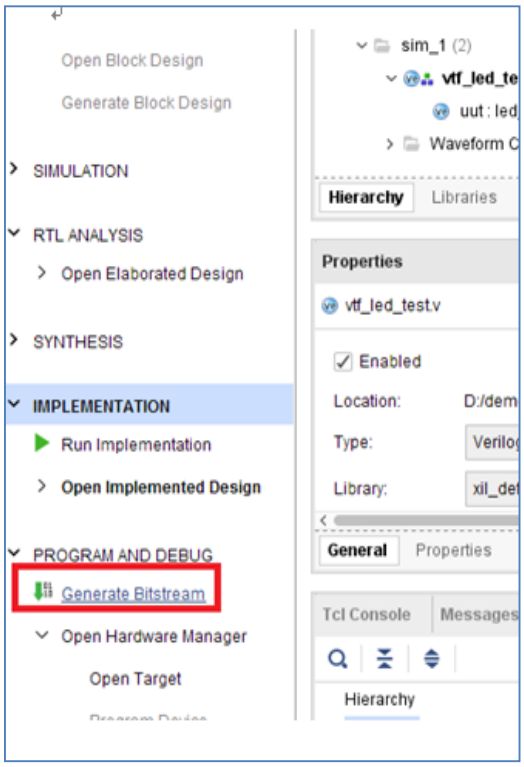
经过前面的编译和仿真，我们可以把 bit 文件下载到 FPGA 芯片中，看一下 LED 实际运行的效果。下载和调试之前先连接硬件，把 JTAG 下载器和开发板连接如下图，然后开发板上电。

1. 下载之前还需进行设置：右击 PROGRAM AND DEBUG 按下图进行设置。

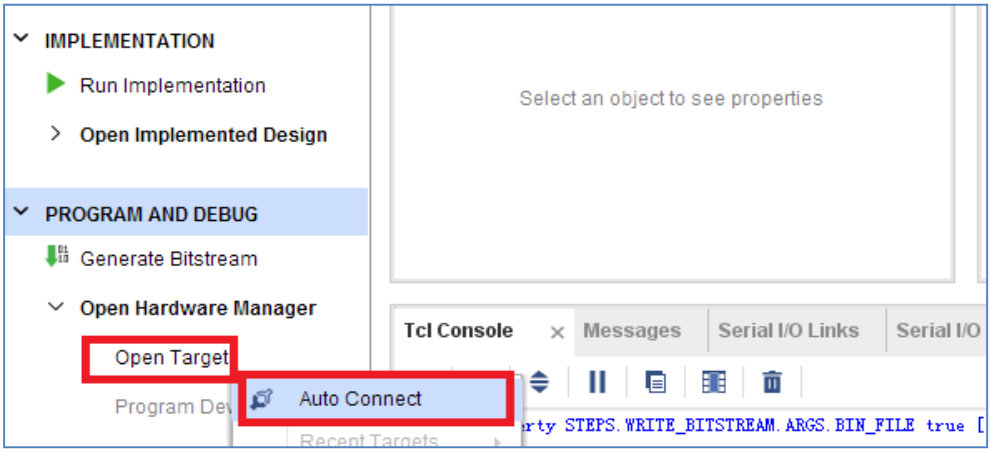

Generate Block Design
SIMULATION
RTL ANALYSIS
Open Elaborated Design
SYNTHESIS
IMPLEMENTATION
Run Implementation
Open Implemented Design
PROGRAM AND DEBUG
Generate Bitstre
Open Hardware
Open Target
Program Device
Bitstream Settings...
Close
uut : led_
Waveform Co
Hierarchy
Libraries
Properties
vtf_led_test.v
Enabled
Location:
Type:
Library:
General
Properties
Messages
Hierarchy
Summary

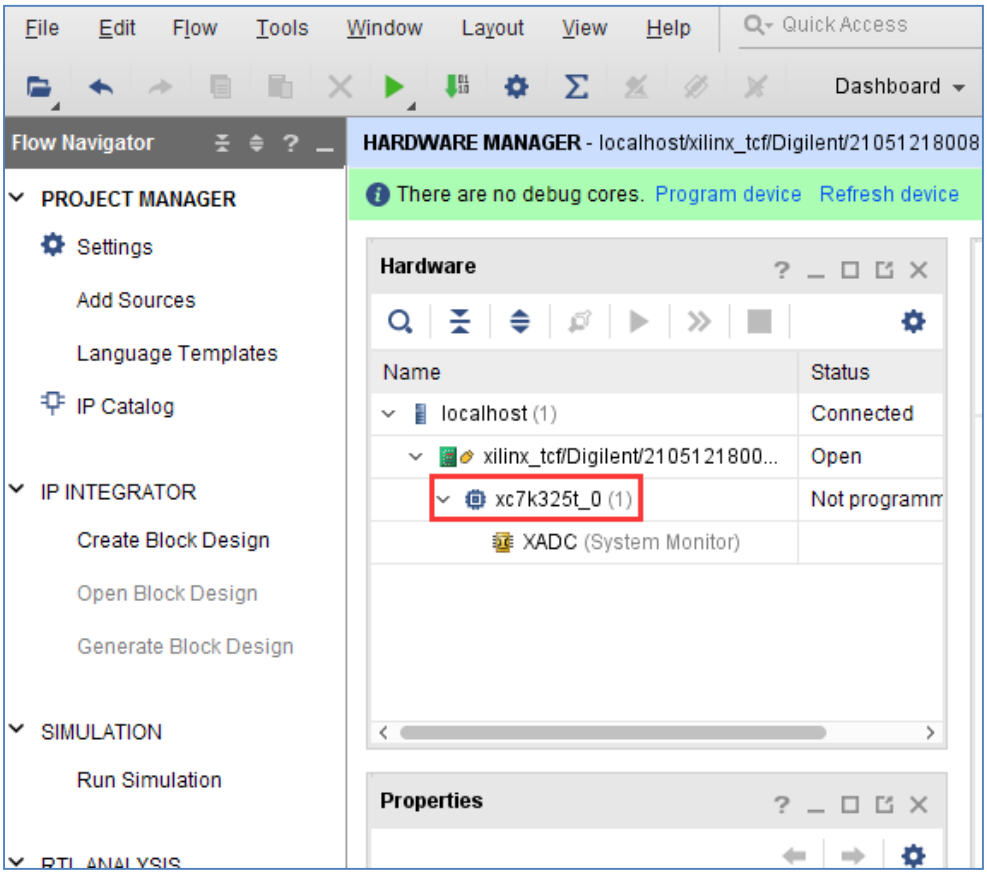

Settings
Project Settings
General
Simulation
Elaboration
Synthesis
Implementation
Bitstream
IP
Tool Settings
Project
IP Defaults
Source File
Display
WebTalk
Help
Text Editor
3rd Party Simulators
Colors
Selection Rules
Shortcuts
Strategies
Bitstream
Specify various settings related to writing bitstream
Configure additional bitstream settings.
Write Bitstream (write_bitstream)
tcl.pre
tcl.post
-raw_bitfile
-mask_file
-no_binary_bitfile
-bin_file
-readback_file
-logic_location_file
-verbose
More Options
-bin_file
Write a binary bit file without header (.bin).
OK
Cancel
Apply
Restore...

设置完成后单击 Generate Bitstream 产生中 bit 和 bin 文件。

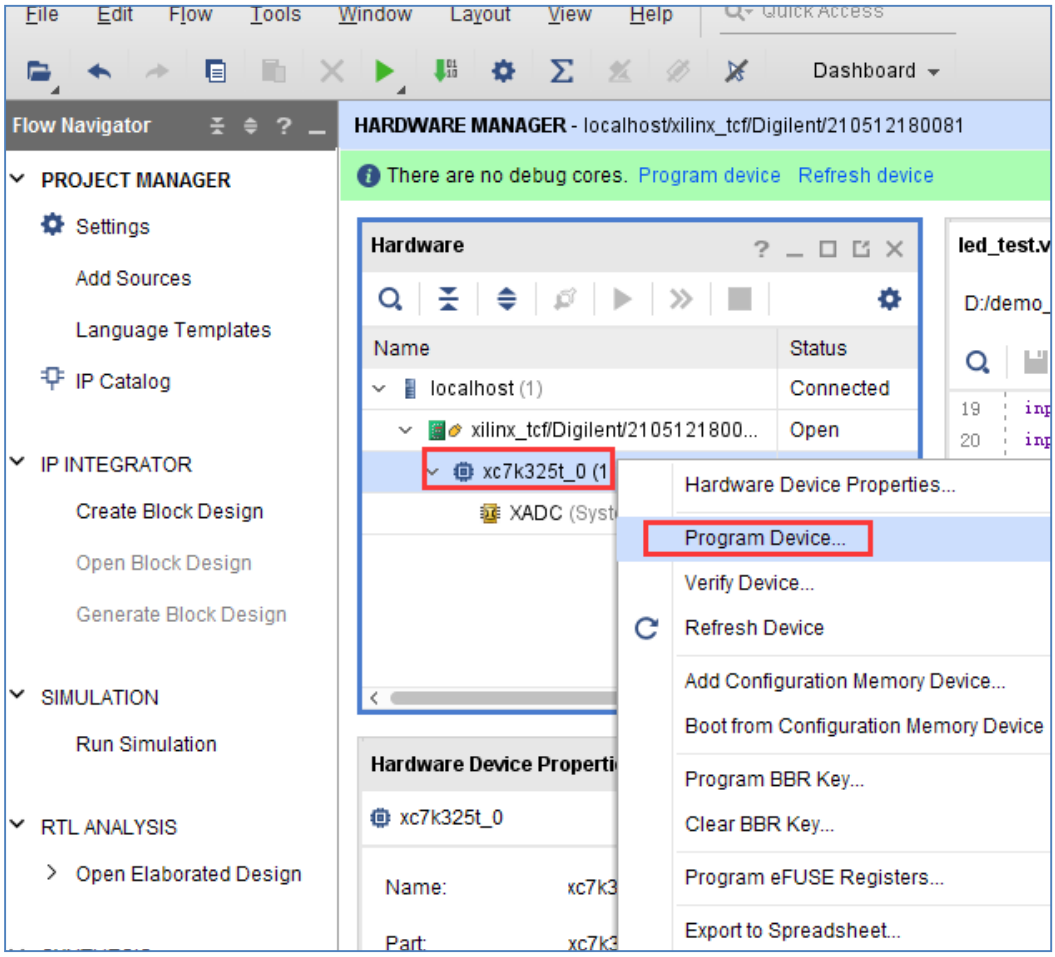


2. bit 文件生成后，点击 Open target 按钮->Auto Connect，在 hardware 界面下会显示 xc7k325t_0 的图标，说明 JTAG 连接已经建立。

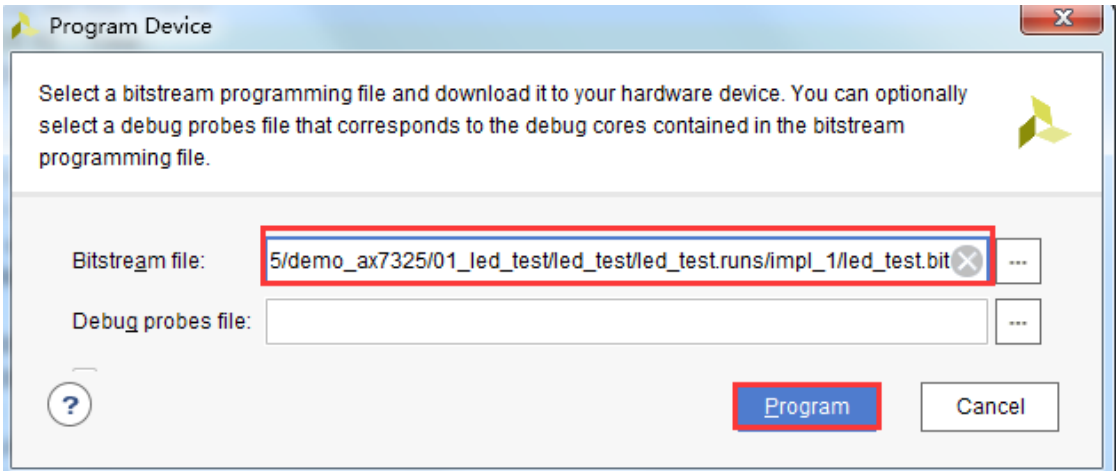




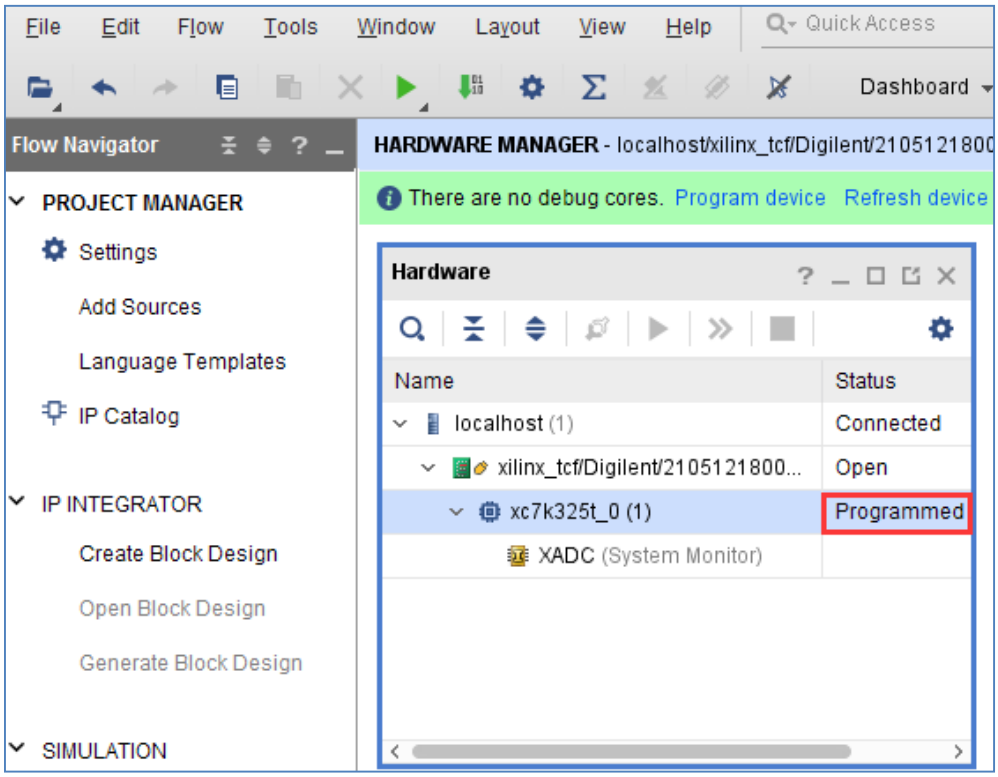
右键选择 xc7k325t_0，在弹出的选项里选择 Program Device 项。



在弹出的 Program Device 对话框中，选择 led_test 项目生成的 bit 文件，点击 Program 按钮烧写 FPGA。



烧写完成后的状态会变成 Programmed, 这时我们可以看到开发板上的四个 LED 灯已经在做流水灯动作了。



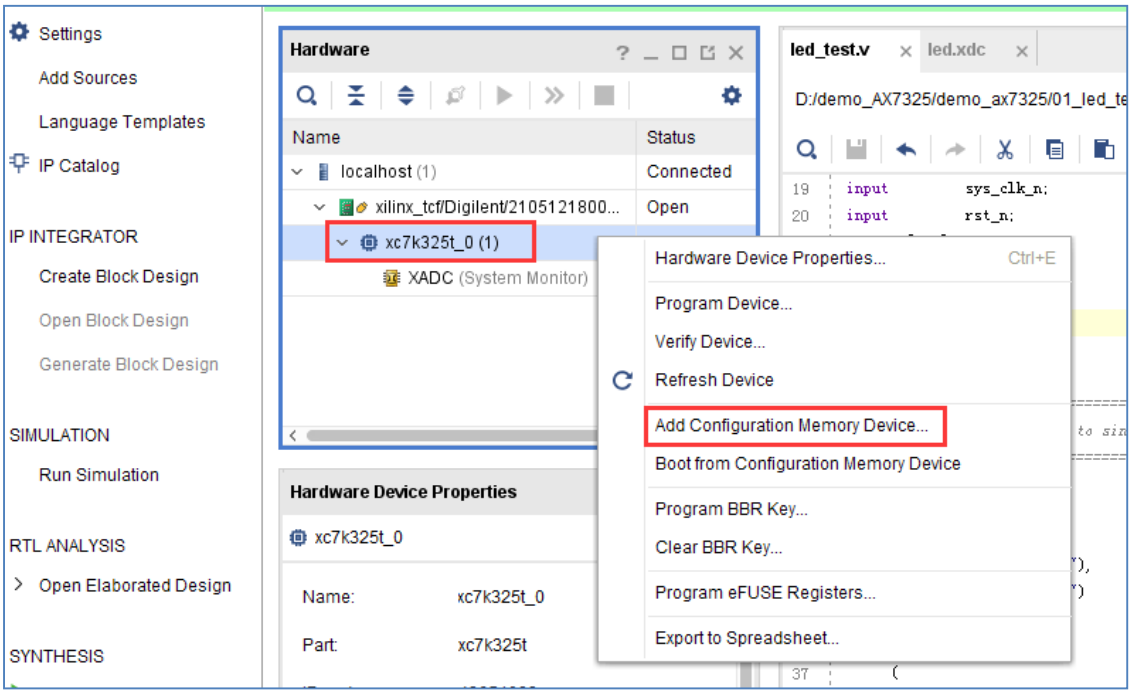
你也可以试着别的花样来点亮 LED，比如，让灯跑得更快一些，或几个灯同时亮同时灭等等，就看你的想象力了，通过自己写程序更能有成就感，而且还能把书本的知识用到实际中，何乐而不为呢！是吧?

4.7 FLASH 程序生成与固化

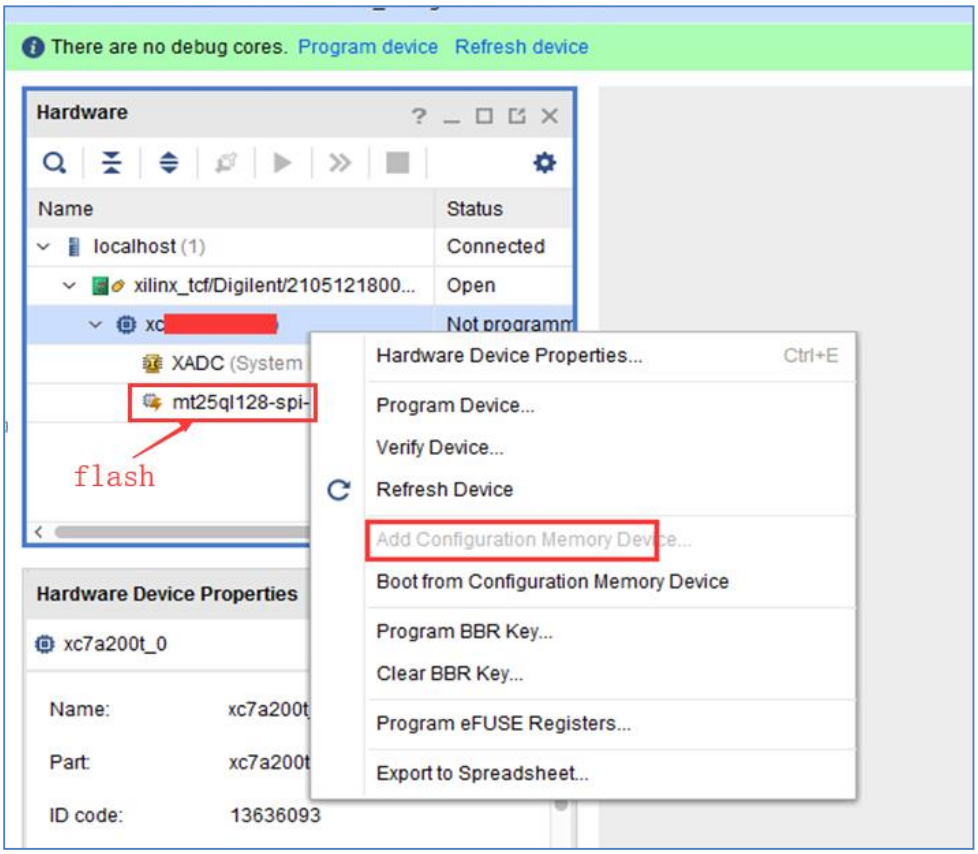
可能已经有朋友发现下载 Bit 文件到 FPGA 后，开发板重新上电后配置程序已经丢失，还需要 JTAG 下载。这岂不麻烦！好吧，这一节我们来介绍如何把配置程序固化到开发板上的 FLASH 中，这样不用担心掉电后程序丢失了。

在我们的开发板上有一个 8Pin 的 128Mbit 的 FLASH, 用于存储配置程序。我们不能直接把 Bit 文件下载到这个 FLASH 中，需要把 Bit 文件转换成 BIN 文件或者 MCS 文件。下面以下载 BIN 文件为例为大家介绍 FLASH 程序的固化。

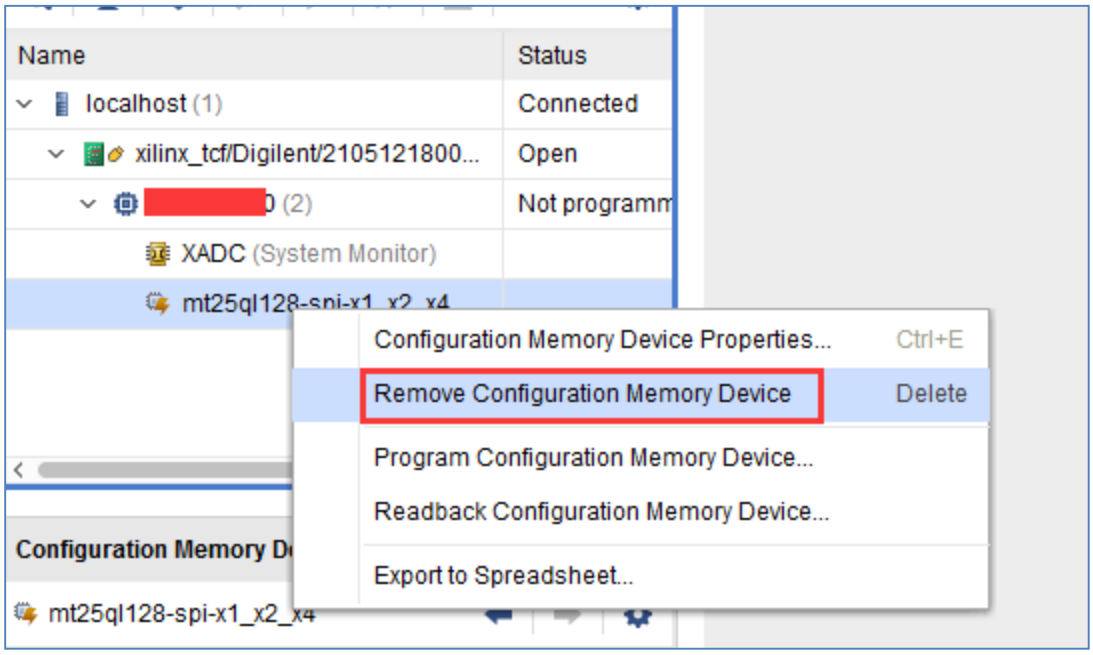
1. 在如下图中右键选择 xc7k325t_0 芯片，在弹出的列表中选择 Add Configruation Memory Device...



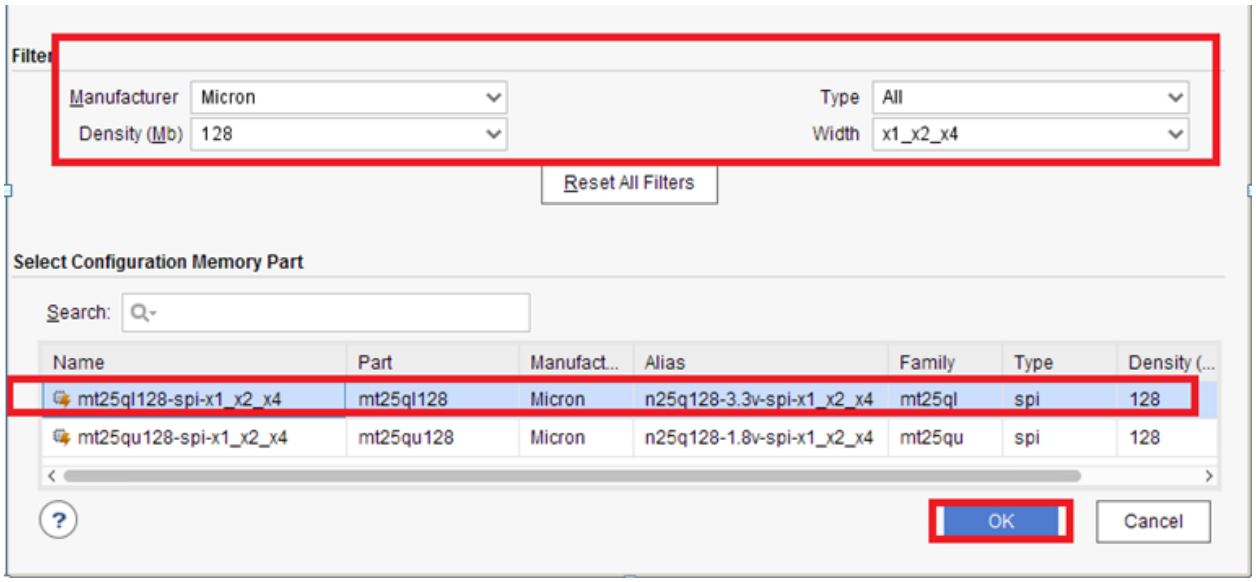
注意：发现此项变为灰色不能选，是因为工程中已经选有 FLASH 配置，不用再添加 flash，如下：



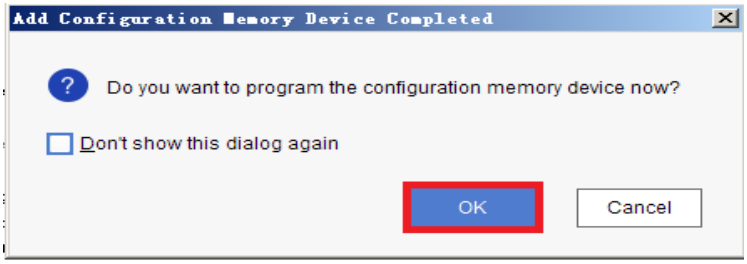
当然自己如果想在已有 flash 的工程中再次添加一下进行实验也有办法，可按如下图移除 flash，然后按上面添加 flash 的步骤进行即可：



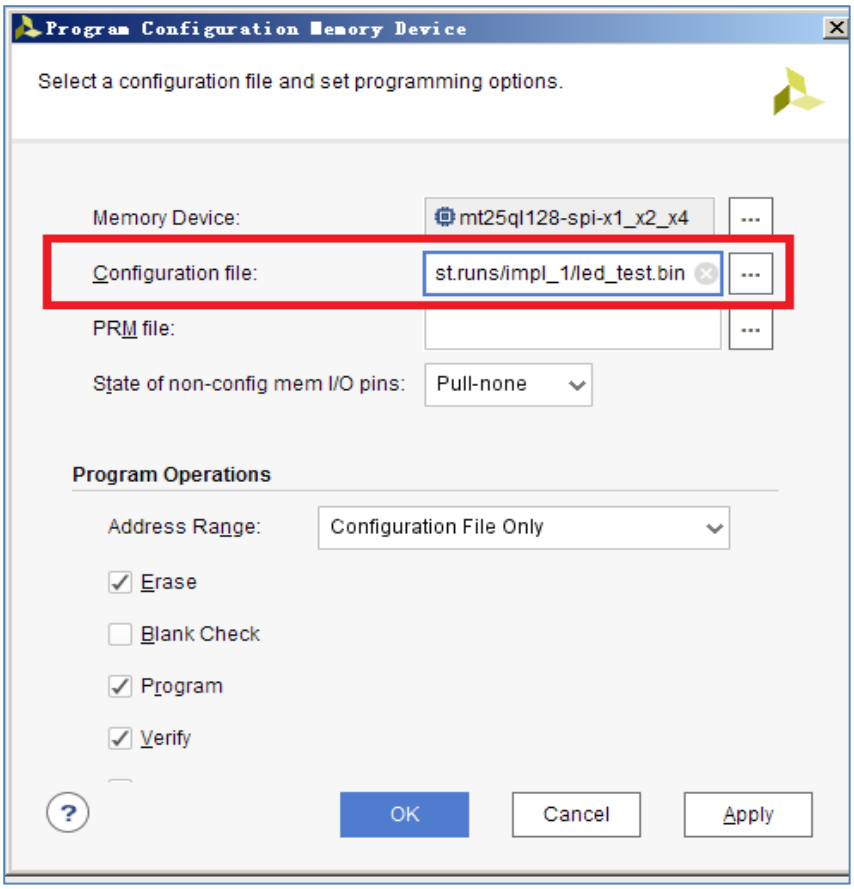
2. 在 Add configruation Memory Device 的配置界面里选择正确的 FLASH 型号，如下图所示：



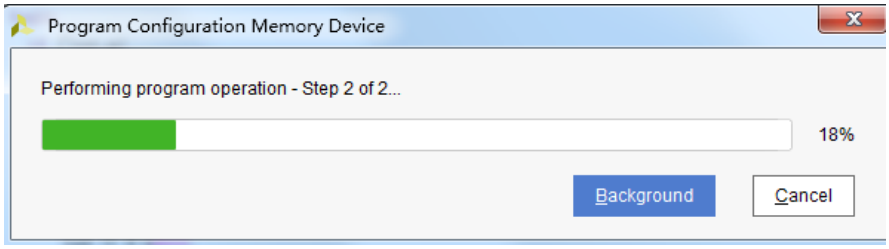
3 . 提示是否对 SPI FLASH 进行编程， 点击 OK。



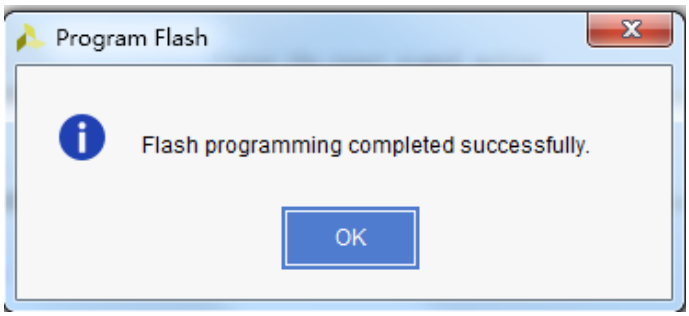
在弹出的 Program Configuration Memory Device 窗口中，Configration file 项选择 Vivado 生成的 led_test.bin 文件（此文件默认在 imp1_1 目录下）。PRM File 项不用选。另外在这个窗口用户还可以配置 I/O 为上拉，下拉或者无上下拉。配置操作选项保留默认就可以。



点击 OK 开始编程 FLASH。



FLASH 编程完毕后，会弹出如下成功的界面。



至此， FLASH 烧写完毕，led_test 程序已经固化到 FLASH 中了。我们来验证一下，关电后拔掉下载器重新启动开发板，等待一会儿你就可以看到开发板上的 LED 灯已经在做跑马运动了。

可能您也发现了，关电后重新上电需要等好一会儿，开发板上的 LED 灯才会开始启动跑马动作。这对有些上电马上就要工作的项目肯定是不满足了，那有没有办法解决的呢！当然有的，我们可以提高 SPI FLASH 的读写时钟，方法很简单，我们只要在 xdc 文件里再加入以下几条语句：

```
#############SPI Configurate Setting##################

set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [current_design]

set_property CONFIG_MODE SPIx4 [current_design]

set_property BITSTREAM.CONFIG.CONFIGRATE 50 [current_design]
```

修改 xdc 文件后需要重新编译，再重新生成 bit 和 bin 文件，然后按前面的方法再烧写一遍 SPI FLASH 芯片哈。完成后开发板上电，这下是不是一上电，LED 灯就开始做运动了吧?

这里为止，我们的第一个项目就圆满完成了，相信您也掌握了 Vivado 的 FPGA 开发的整个流程，再也不是那个 FPGA 的门外汉了吧！ 师傅领进门，修行还需要靠本身！vivado 软件的一些技巧的使用和掌握就需要靠大家在长期实践和探索中慢慢熟悉了。

5 附录

led_test.v(verilog 代码)

```
//////////////////////////////////////////////////////////////////////////////////
// Module name: led_test.v
//////////////////////////////////////////////////////////////////////////////////
`timescale 1ns / 1ps

module led_test
(
        input               sys_clk_p,      // Differentia system clock 200Mhz input on board
        input               sys_clk_n,
        input               rst_n,          // reset ,low active
        output reg [1:0]        led,            // LED,use for control the LED signal on board
        output              fan_pwm         //fan control
 );
//define the time counter
reg [31:0]   timer;
```

```
assign fan_pwm = 1'b0;
//=============================================================================
//Differentia system clock to single end clock
//=============================================================================
wire        sys_clk;

// IBUFDS: Differential Input Buffer
//         Kintex-7
// Xilinx HDL Language Template, version 2017.4

IBUFDS #(
        .DIFF_TERM("TRUE"),       // Differential Termination
        .IBUF_LOW_PWR("TRUE"),     // Low power="TRUE", Highest performance="FALSE"
        .IOSTANDARD("DEFAULT")     // Specify the input I/O standard
) u_ibuf_sys_clk (
        .O(sys_clk),  // Buffer output
        .I(sys_clk_p),  // Diff_p buffer input (connect directly to top-level port)
        .IB(sys_clk_n) // Diff_n buffer input (connect directly to top-level port)
);
//=============================================================================
// cycle counter:from 0 to 1 sec
//=============================================================================
  always @(posedge sys_clk or negedge rst_n)
    begin
      if (~rst_n)
          timer <= 32'd0;                     // when the reset signal valid,time counter
clearing
      else if (timer == 32'd199_999_999)    //1 seconds count(200M-1=199999999)
          timer <= 32'd0;                       //count done,clearing the time counter
      else
                  timer <= timer + 1'b1;            //timer counter = timer counter + 1
    end

//=============================================================================
// LED control
//=============================================================================
  always @(posedge sys_clk or negedge rst_n)
    begin
      if (~rst_n)
          led <= 2'b00;                  //when the reset signal active
      else if (timer == 32'd49_999_999)    //time counter count to 0.25 sec,LED1 lighten
          led <= 2'b01;
      else if (timer == 32'd99_999_999)    //time counter count to 0.5 sec,LED2 lighten
          led <= 2'b10;
      else if (timer == 32'd149_999_999)   //time counter count to 0.75 sec,
          led <= 2'b00;
      else if (timer == 32'd199_999_999)   //time counter count to 1 sec,LED1,LED2 lighten
          led <= 2'b11;
    end

endmodule
```

注意：在定义寄存器时，如果寄存器在 always 块里使用必须定义为 reg 类型，如果仅是用于连线或是直接赋值需定义为 wire 类型，输入信号的类型不能定义为 reg 型，不管是 reg 类型信号还是 wire 类型的信号，定义的寄存器宽度必须满足使用时的需要，但必须稍大于或等于需要使用的位宽。若定义寄存器位宽远远大于使用需求则会浪费资源，如果定义的位宽小于使用需求，则会造成数据位截断，导致程序错误。还有其他信号的类型及用法请大家参考 Verilog 语法教程。