LED Blinking experiment and simulation in vivado

Technical Email: alinx@aithtech.com Sales Email: rachel.zhou@alinx.com

1 Experiment Introduction

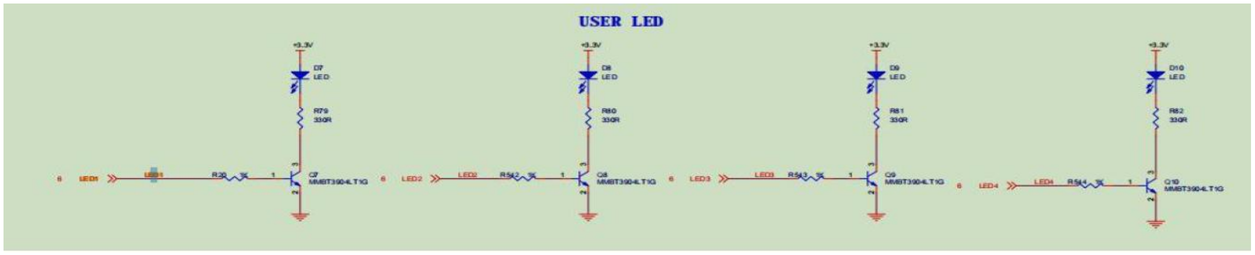
Through the LED flow lamp experiment, the basic process of developing FPGA using vivado software is introduced, device selection, setting, code writing, compiling, distributing pins, downloading, program FLASH curing, erasing, etc.; also checking whether the LED lights on the board are normal.

2 Experiment Environment

- Windows 7 SP1 64 bit
- vivado 2017.4
- ALINX Brand FPGA Development Board (AXKU040 FPGA Development Board)

3 Experiment Principle

3.1 LED Hardware Circuit



LED Hardware Circuit on AXKU040 FPGA Development Board

As can be seen from the LED part schematic above, the AXKU040 FPGA development board connect the IO through a resistor and LED in series. The IO output of the FPGA are high level, that will illuminates the LED. The IO output are low level, then LED is off.

3.2 Programming

In the design of the FPGA, the counter is usually used for timing. For a 200Mhz system clock, one clock cycle is 5ns, then one second needs 200000000 clock cycles. If one clock cycle counter is accumulated once, the counter is from 0 to 199999999 is exactly 200000000 cycles. That is the 1 second clock.

A 32-bit counter is defined in the program::

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

The maximum can be 4294967295, hexadecimal is FFFFFFFF, if the counter reaches the maximum value, it can represent 85.89934592 seconds. In the program design, the LED changes once every 0.25 second, and a total of 1 seconds is used to run a cycle.

```
    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
```

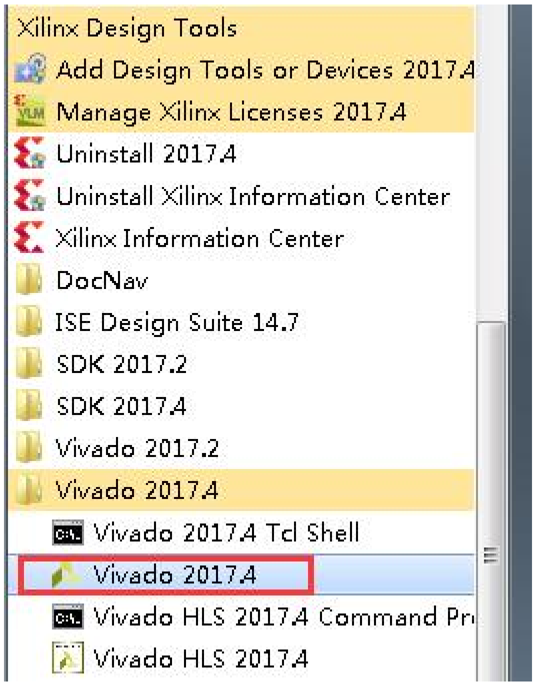
The state of the LED is changed at 0.25 seconds, 0.5 seconds, 0.75 seconds, and 1 second, and the original value remains unchanged at other
times.

```
    // 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'b01;
        else if (timer == 32'd199_999_999)
            led <= 2'b10;
    end
```

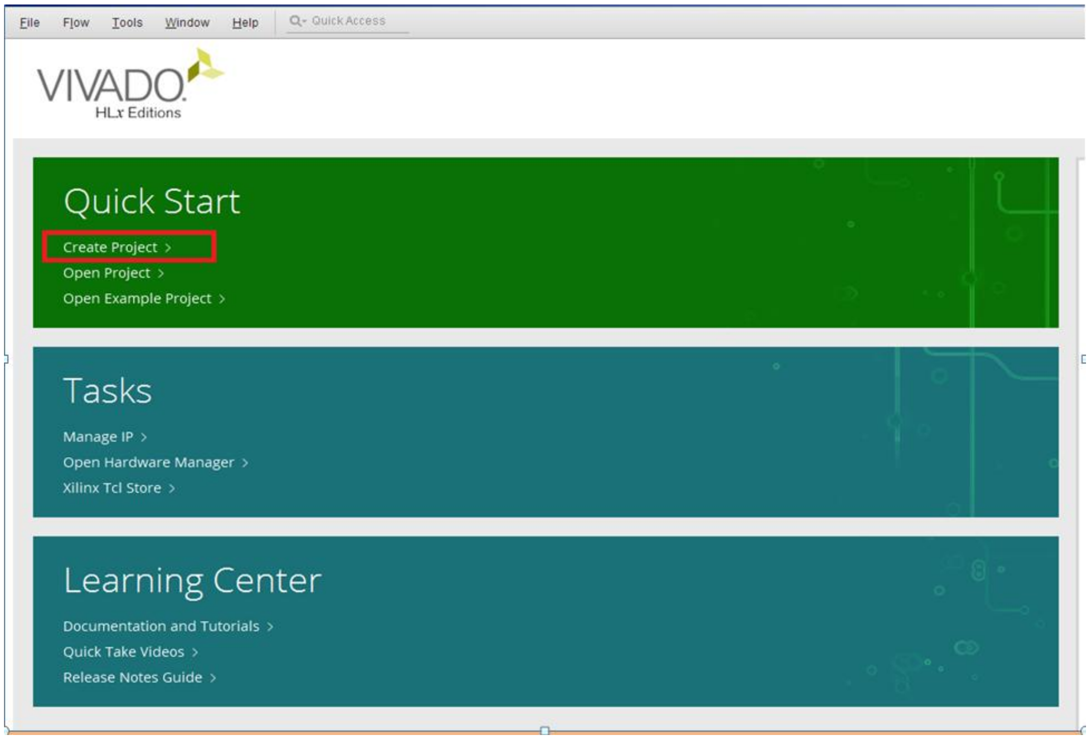
4 Vivado Project

4.1 Create a project

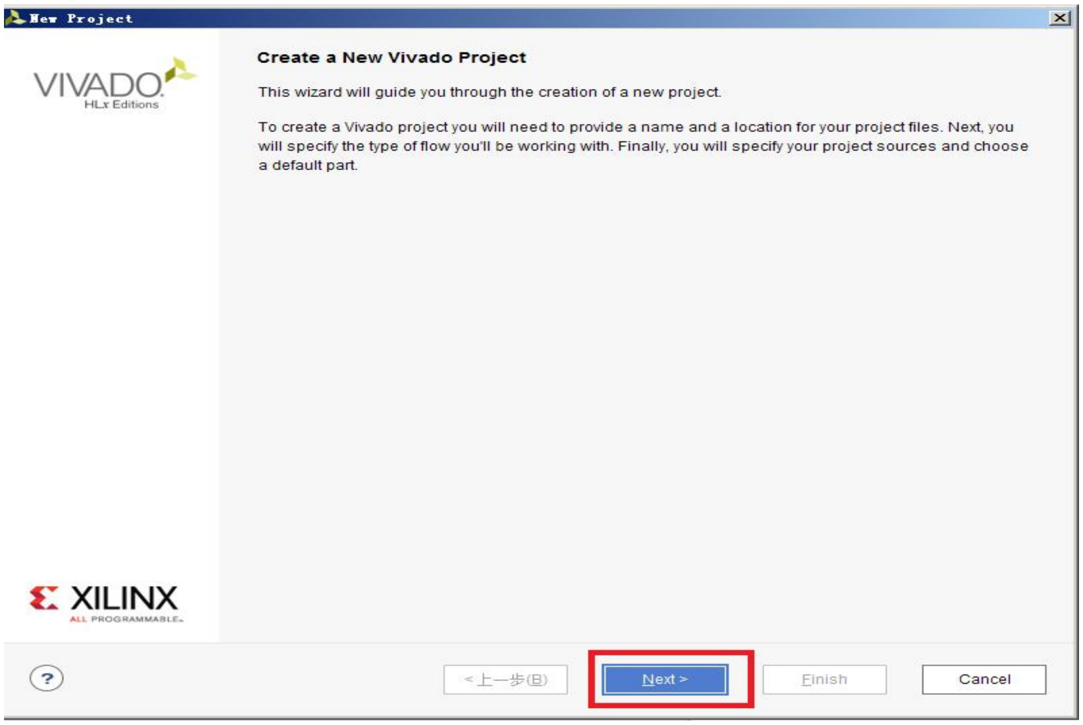
1. Start the Vivado 2017.4 development environment (Select Xilinx Design Tools->Vivado 2017.4->Vivado 2017.4 in the Start menu or double-click the icon of Vivado 2017.4 on the desktop to open the software directly)



2. Double-click Create Project in the Vivado 2017.4 development environment, as shown below:



3. Pop up a Vivado project wizard and click the Next button.

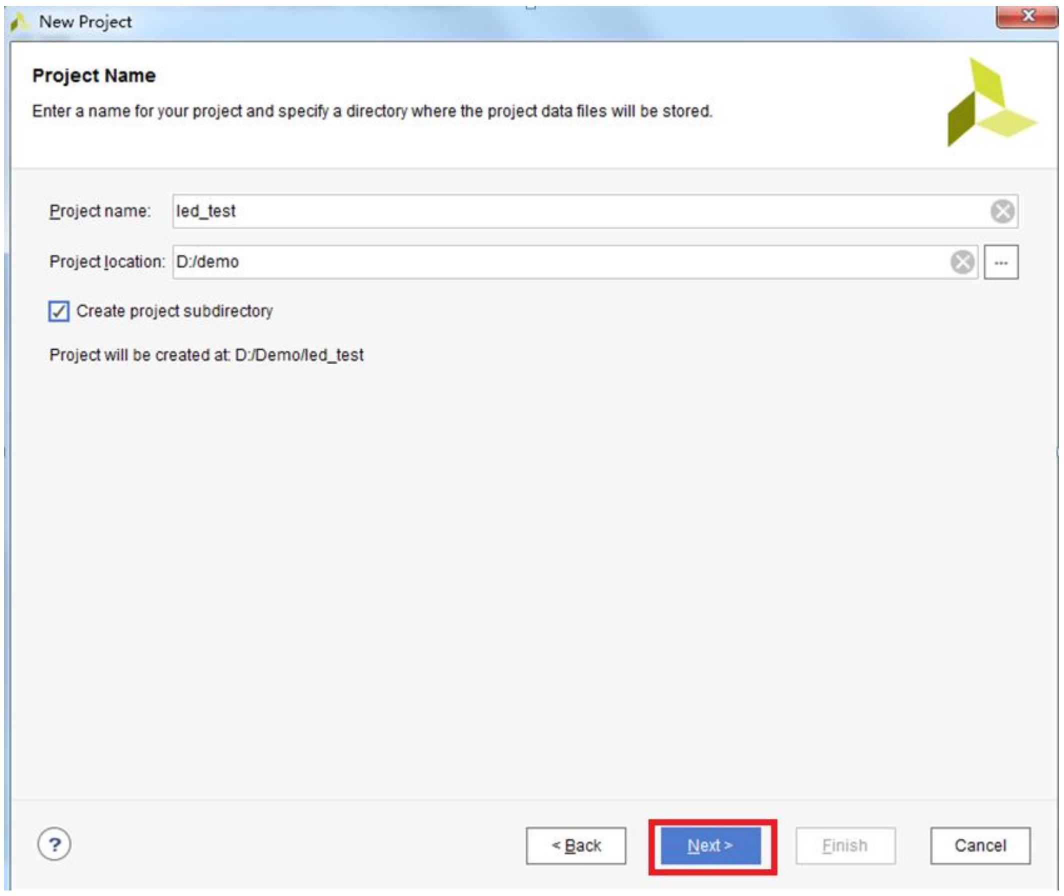

New Project

Create a New Vivado Project

This wizard will guide you through the creation of a new project.

To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.

4. In the pop-up dialog box, enter the project name and the directory where the project is stored. Here, take a project name of led_test and click Next.


New Project

Project Name

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

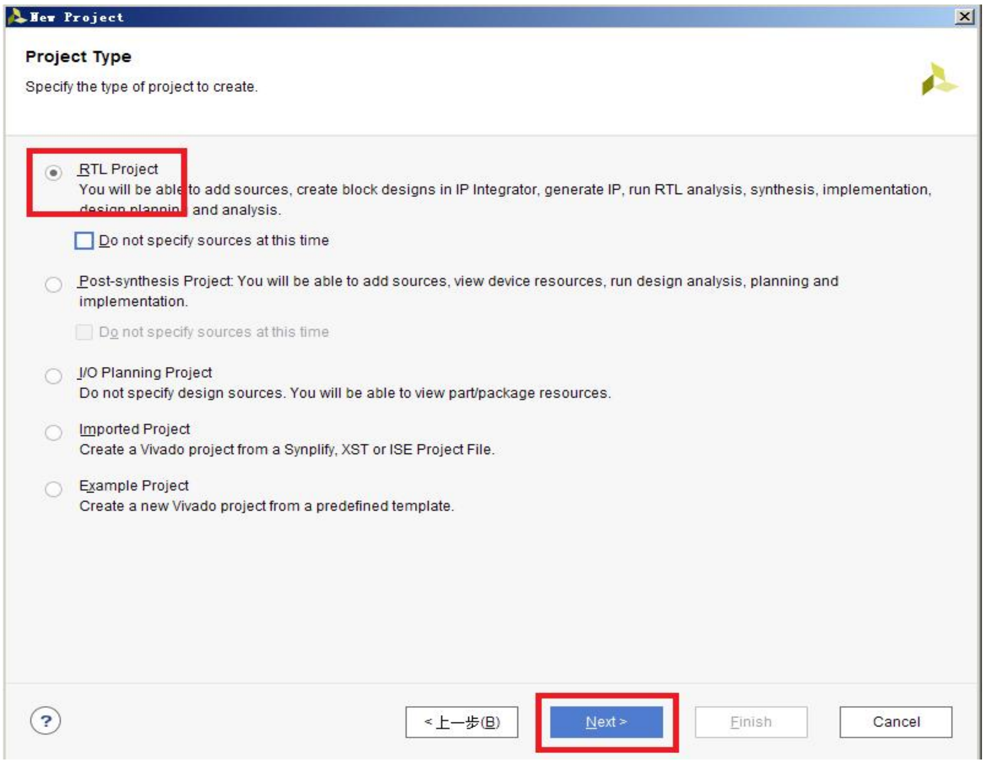
Project name: led_test

Project location: D:/demo

☑ Create project subdirectory

Project will be created at: D:/Demo/led_test

5. The RTL Project is selected by default in the dialog below, as we are programming here using the verilog behavior description language. The following “Do not specify source at this time ”can also be checked. If you do not check the box, the next step will enter to add the source file interface.


New Project

Project Type

Specify the type of project to create.

(●) RTL Project
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.

☐ Do not specify sources at this time

( ) Post-synthesis Project: You will be able to add sources, view device resources, run design analysis, planning and implementation.

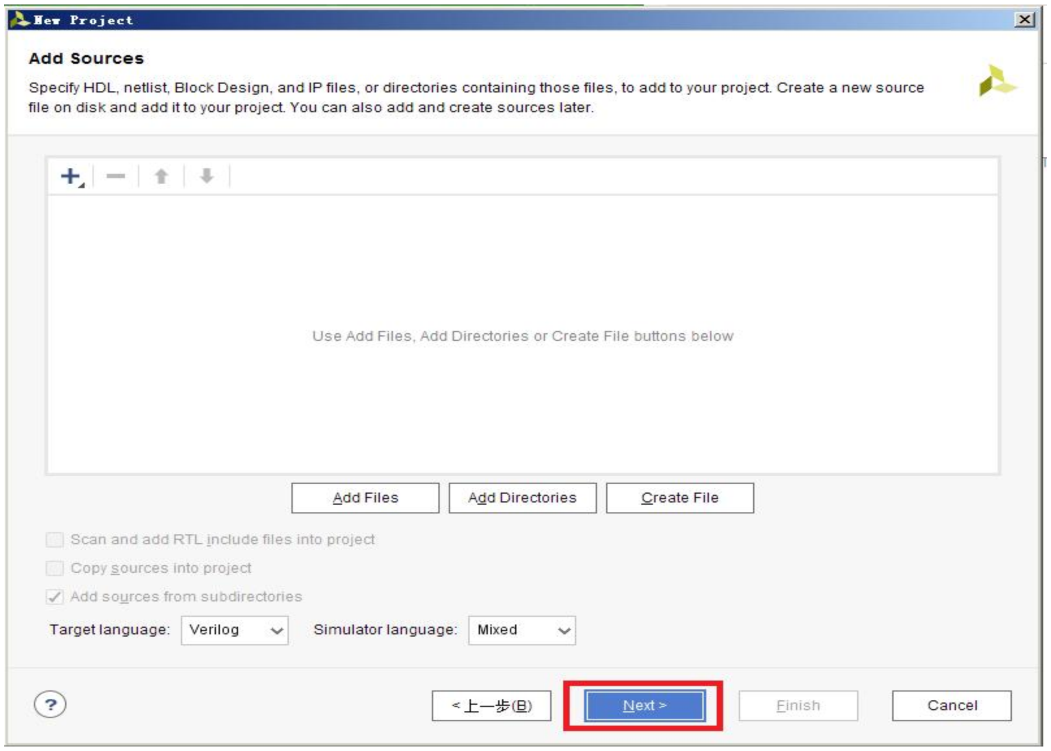
☐ Do not specify sources at this time

( ) I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

( ) Imported Project
Create a Vivado project from a Synplify, XST or ISE Project File.

( ) Example Project
Create a new Vivado project from a predefined template.

6. Go to the add “source file” interface, here do not add any design files. Click Next


New Project

Add Sources

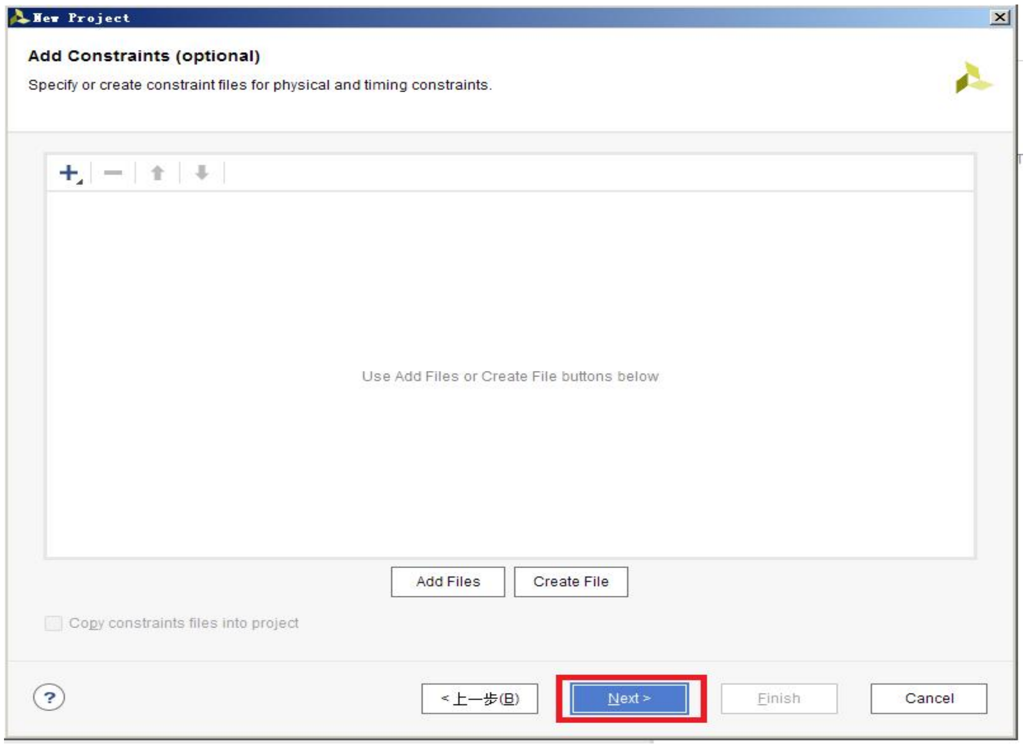
Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.

Use Add Files, Add Directories or Create File buttons below

☐ Scan and add RTL include files into project
☐ Copy sources into project
☑ Add sources from subdirectories

Target language: Verilog   Simulator language: Mixed

7. Prompt whether to add an existing constraint file, here the constraint file we have not designed, do not add



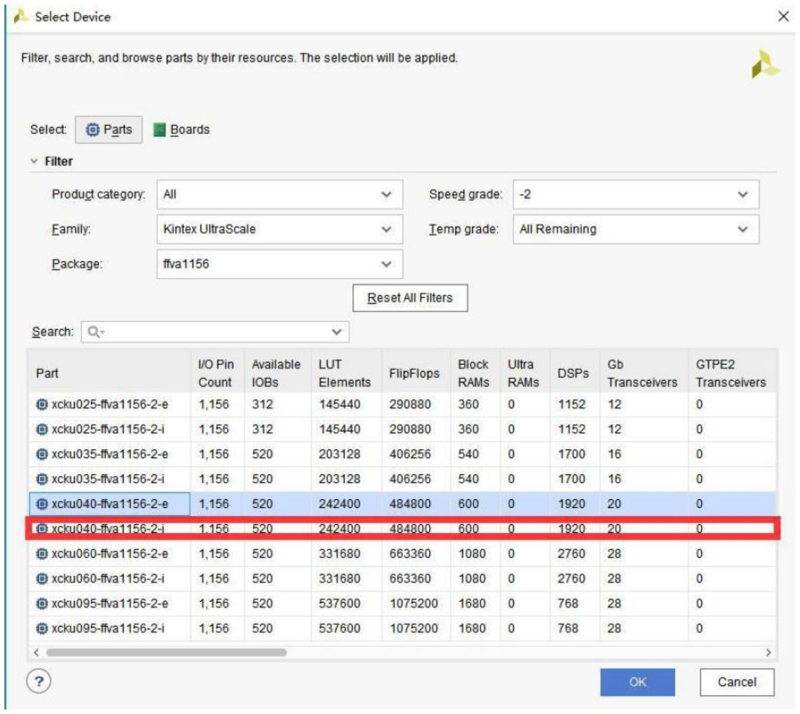
Add Constraints (optional)

Specify or create constraint files for physical and timing constraints.

Use Add Files or Create File buttons below

Copy constraints files into project

8. In the following dialog, select the FPGA device you are using and make some configuration. The FPGA chip model must be the same as the model on the development board. First selects “Kintex UltraScale” in the Family column, selects “-2” in the Speed grade column, selects “ffva1156” in the Package column, and then selects “xcku040_ffva1156-2” in the list below; Click NEXT to go to the next screen:



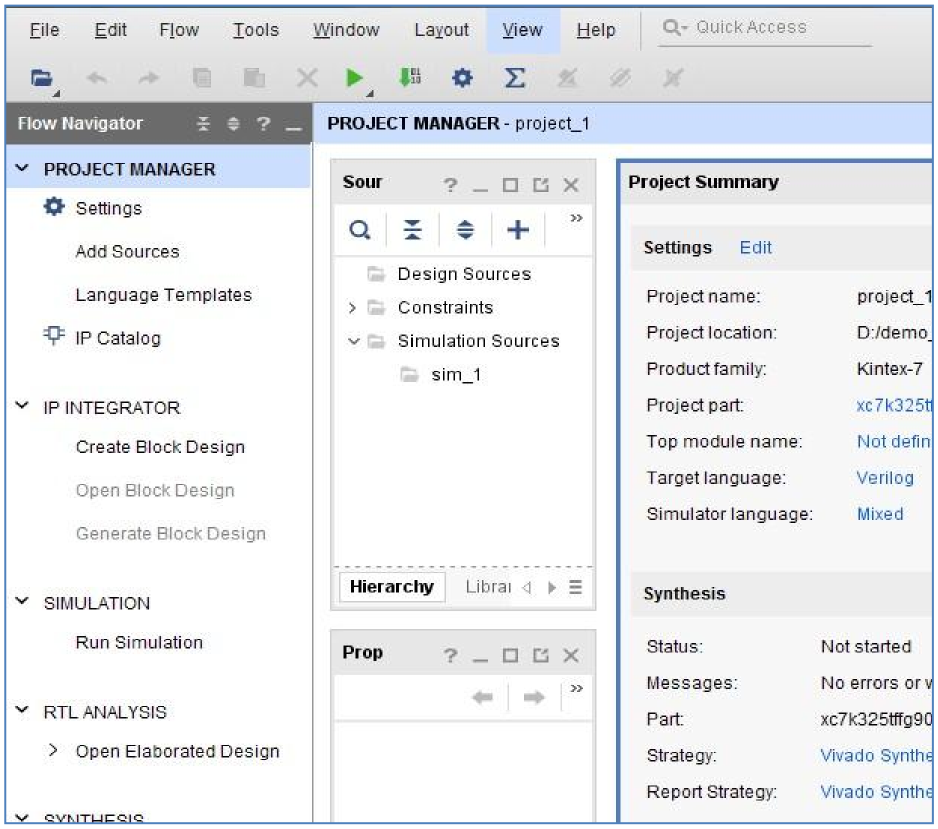
Filter, search, and browse parts by their resources. The selection will be applied.

Product category: All
Speed grade: -2
Family: Kintex UltraScale
Temp grade: All Remaining
Package: ffva1156
Search:

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gb Transceivers	GTPE2 Transceivers
xcku025-ffva1156-2-e	1,156	312	145440	290880	360	0	1152	12	0
xcku025-ffva1156-2-i	1,156	312	145440	290880	360	0	1152	12	0
xcku035-ffva1156-2-e	1,156	520	203128	406256	540	0	1700	16	0
xcku035-ffva1156-2-i	1,156	520	203128	406256	540	0	1700	16	0
xcku040-ffva1156-2-e	1,156	520	242400	484800	600	0	1920	20	0
xcku040-ffva1156-2-i	1,156	520	242400	484800	600	0	1920	20	0
xcku060-ffva1156-2-e	1,156	520	331680	663360	1080	0	2760	28	0
xcku060-ffva1156-2-i	1,156	520	331680	663360	1080	0	2760	28	0
xcku095-ffva1156-2-e	1,156	520	537600	1075200	1680	0	768	28	0
xcku095-ffva1156-2-i	1,156	520	537600	1075200	1680	0	768	28	0

9. Once again, confirm that FPGA Chip Model has been selected. If there is no problem, click "Finish" to complete the project creation.

10. After the project is created, as shown below:

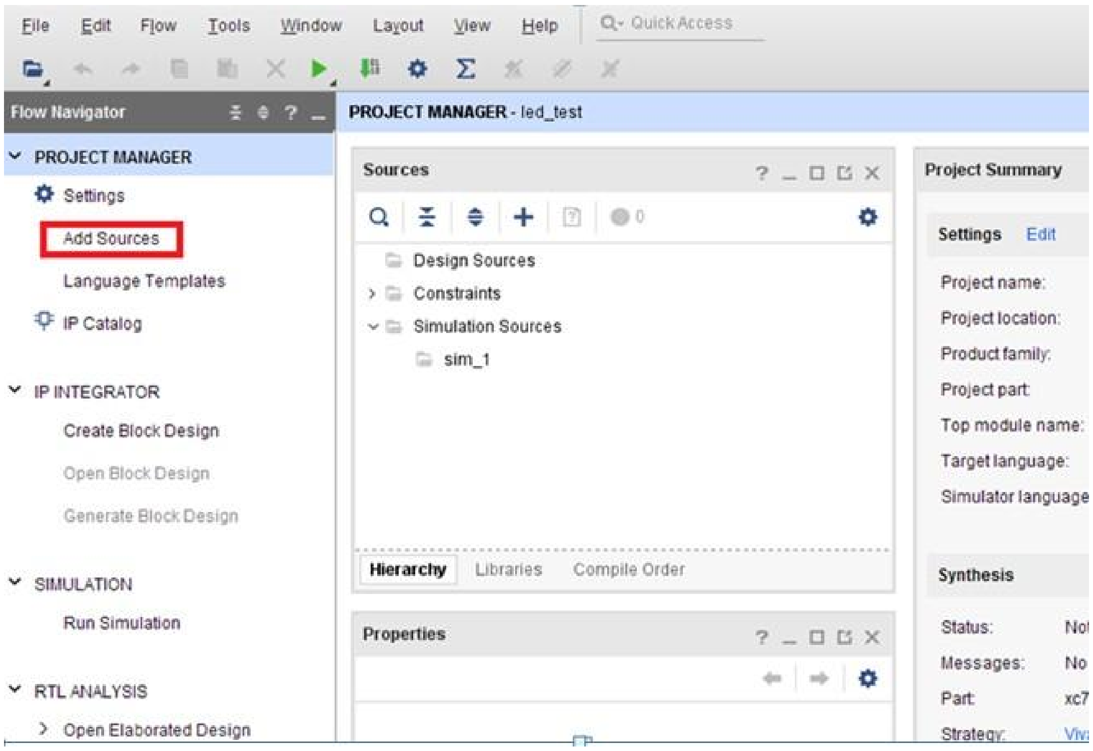


Project name:	project_1
Project location:	D:/demo_
Product family:	Kintex-7
Project part:	xc7k325t
Top module name:	Not defin
Target language:	Verilog
Simulator language:	Mixed

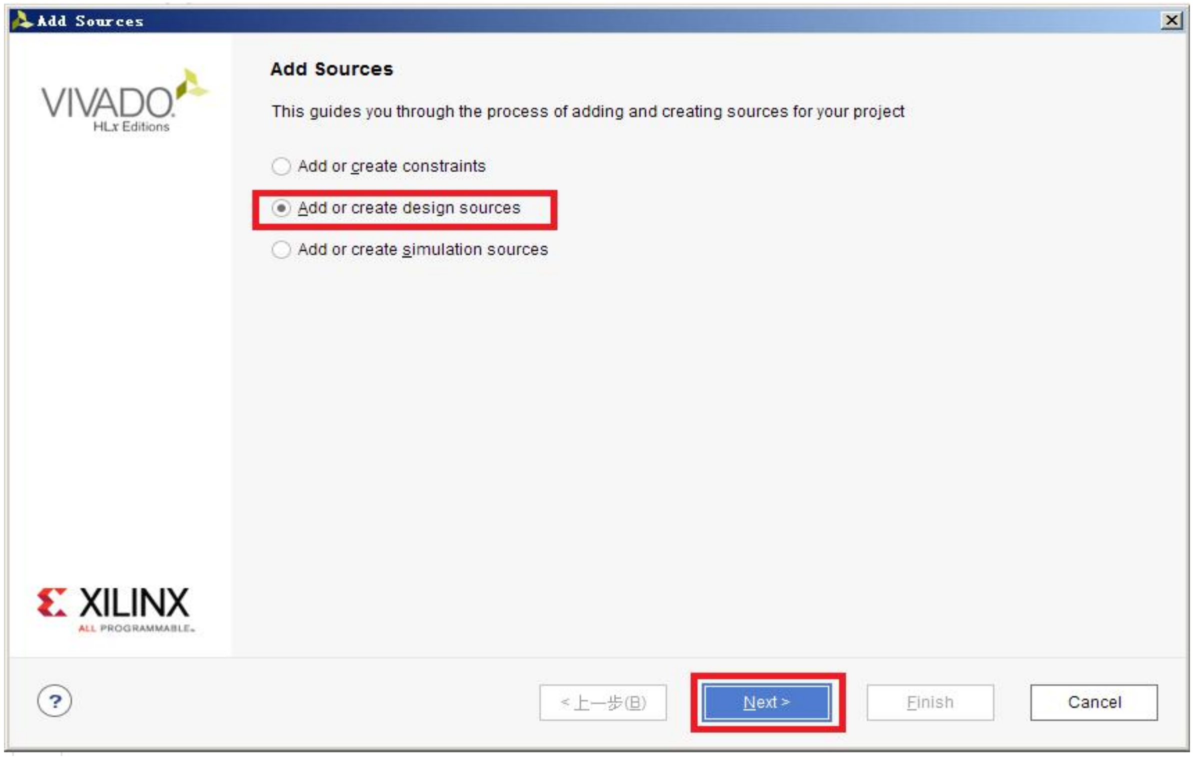
Status:	Not started
Messages:	No errors or v
Part:	xc7k325tffg90
Strategy:	Vivado Synthe
Report Strategy:	Vivado Synthe

4.2 Write the Verilog code of the running light

1. Click “Project Manager ” under the “Add Source” icon (or use hot key Alt+A)



2. Select “Add or create design source”, click “Next”

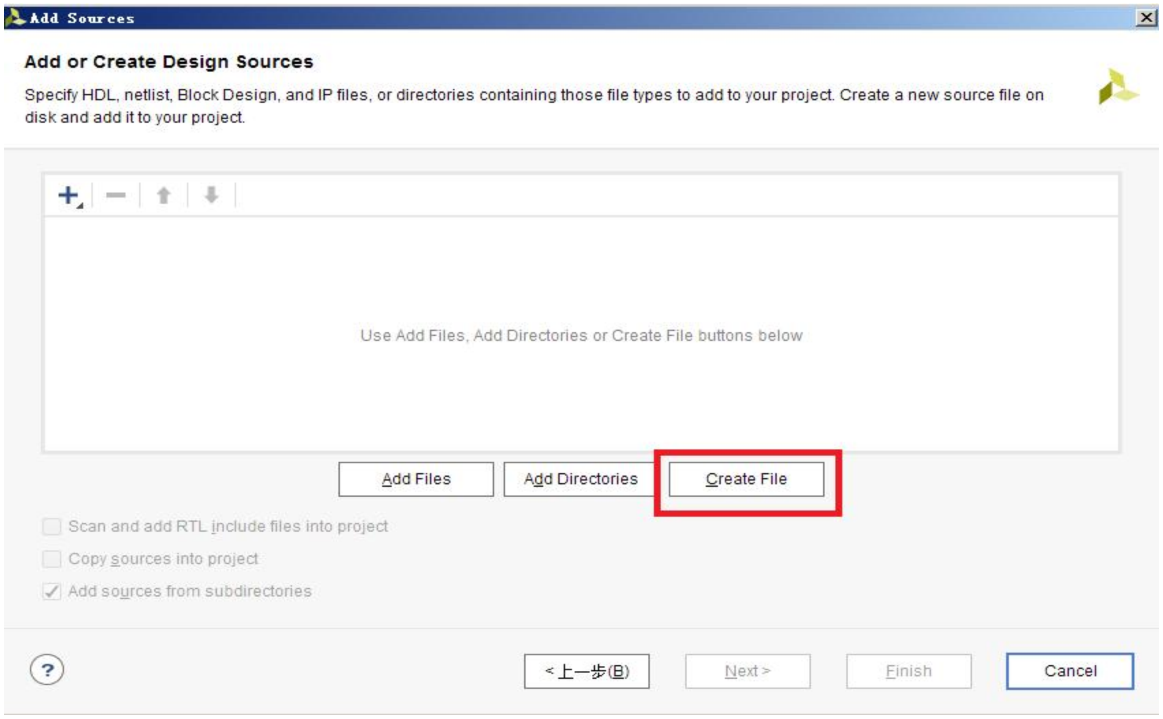


Add Sources

This guides you through the process of adding and creating sources for your project

( ) Add or create constraints
[✓ Add or create design sources]
( ) Add or create simulation sources

3. Click “Add Files” to add source files one by one. Click “Add Directories” to add source files by directory. Since we don't have a design program yet, click the “Create File button” here.



Add Sources

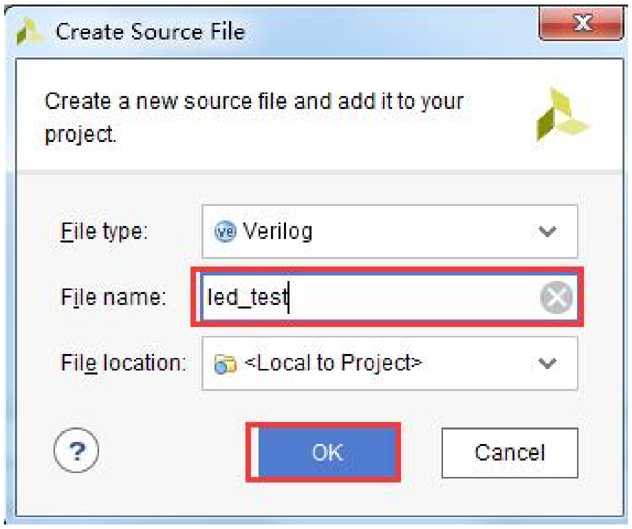
Add or Create Design Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.

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
< 上一步(B)
Next >
Finish
Cancel

In the pop-up dialog box, select “File type” is “verilog”, “File name” is “led_test”, click the OK button.



Create Source File

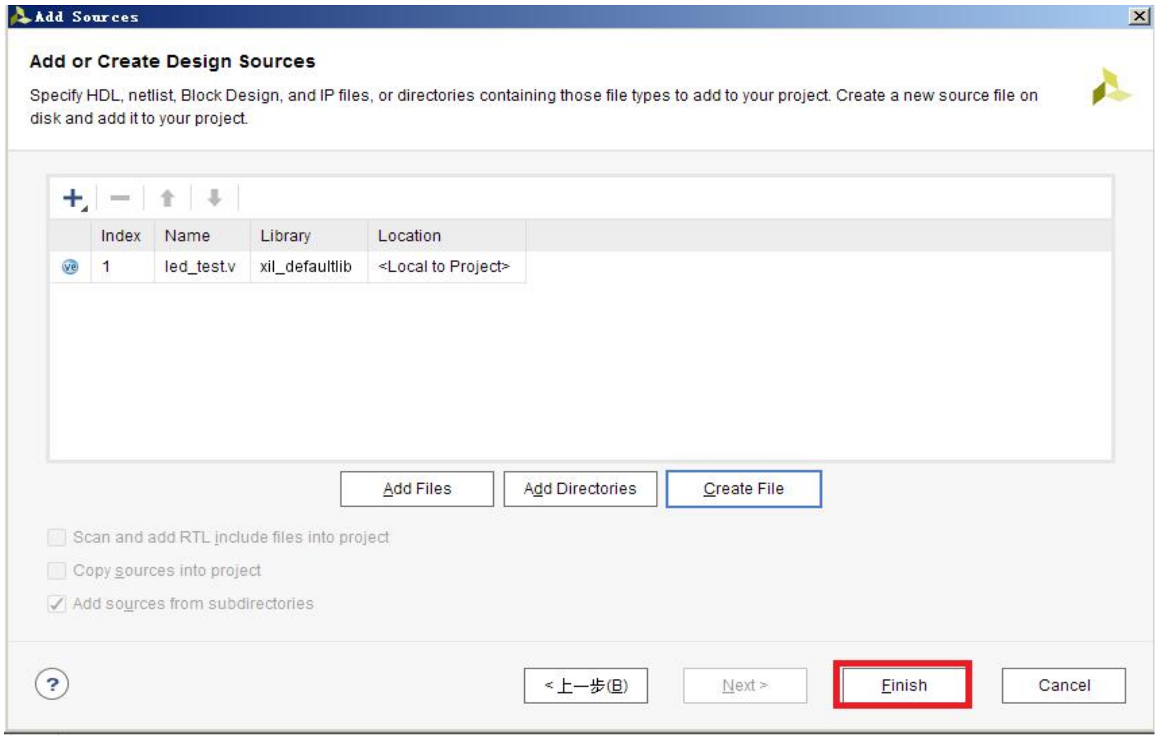
Create a new source file and add it to your project.

File type: Verilog

File name: led_test

File location: <Local to Project>

4. Click "Finish"



Add Sources

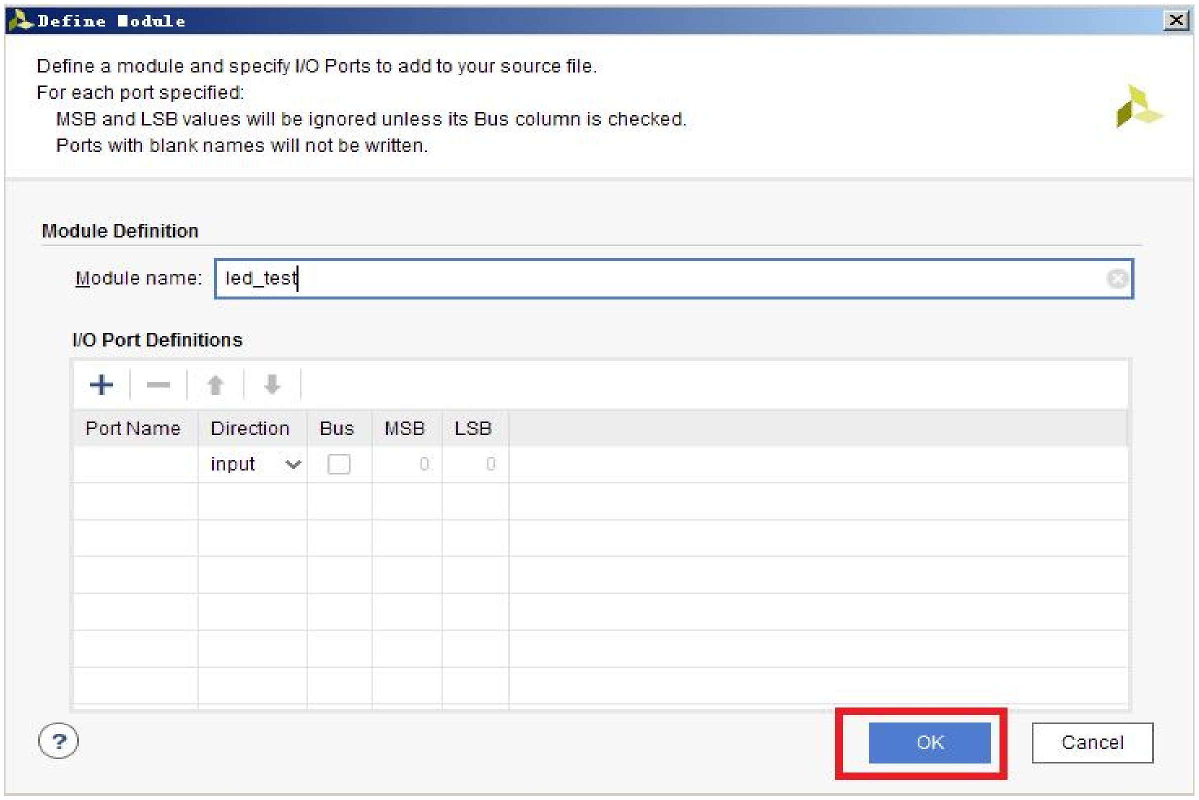
Add or Create Design Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.

	Index	Name	Library	Location
	1	led_test.v	xil_defaultlib	<Local to Project>

Add Files
Add Directories
Create File
Scan and add RTL include files into project
Copy sources into project
Add sources from subdirectories
< 上一步(B)
Next >
Finish
Cancel

The wizard will prompt you to define the I / O port, here not define, you can write it yourself in the program, click OK to complete.



Define Module

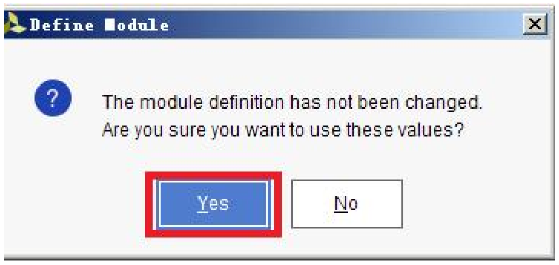
Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

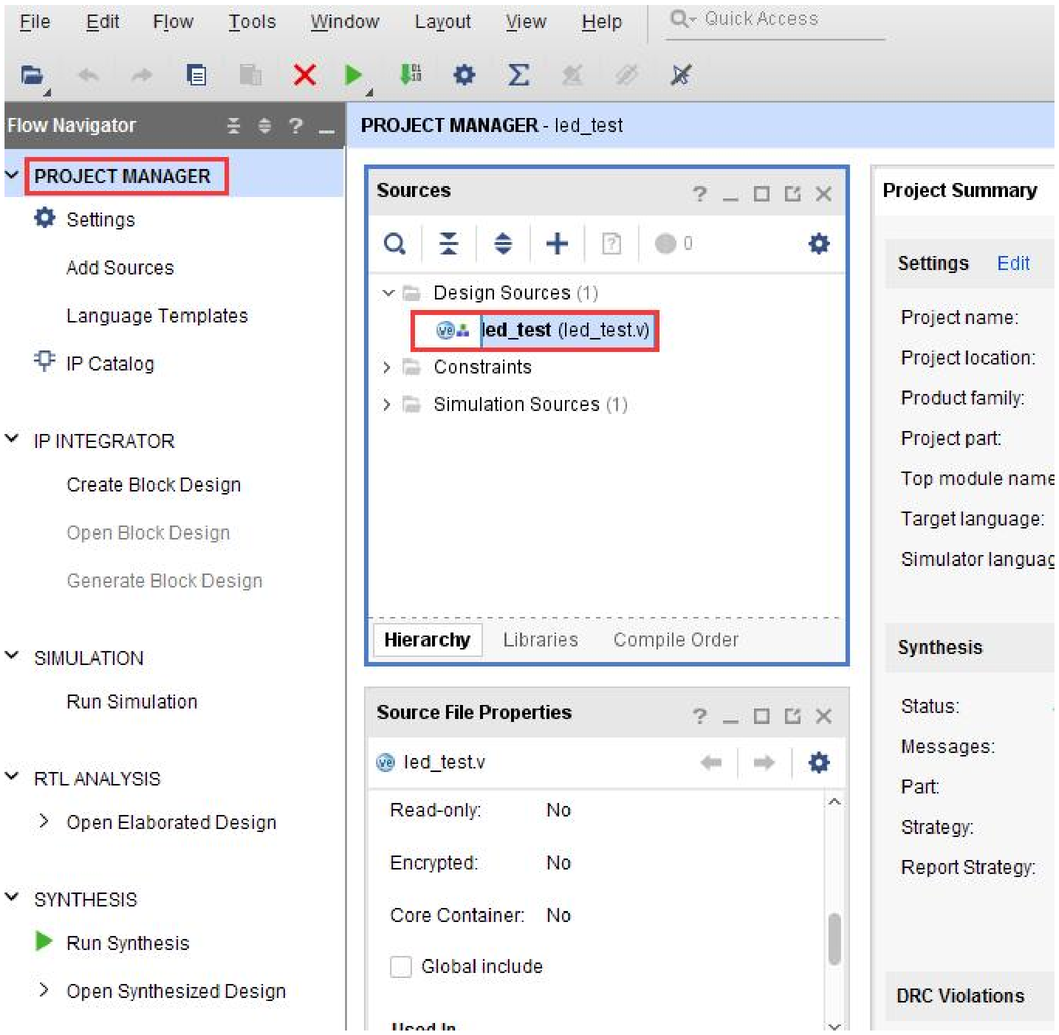
Module name: led_test

I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
	input	☐	0	0


Define Module
The module definition has not been changed.
Are you sure you want to use these values?
Yes
No

Then, there is already a “led_test.v” file in Design Sources under the “Project Manager” interface, and it automatically becomes the top module of the project.



5. Next we will write the “led_test.v” program, here we define a 32-bit register timer for loop counting 0~199_999_999 (1 second), When counting to 49_999_999 (0.25 second), the first LED is extinguished; When counting to 99_999_999 (0.5 second), the second LED is turned off; When counting to 149_999_999 (0.75 second), the third LED is turned off; When counting to 199_999_999 (1 second), the fourth LED is turned off. The counter is recounted. Look at the code directly for the specific operation.

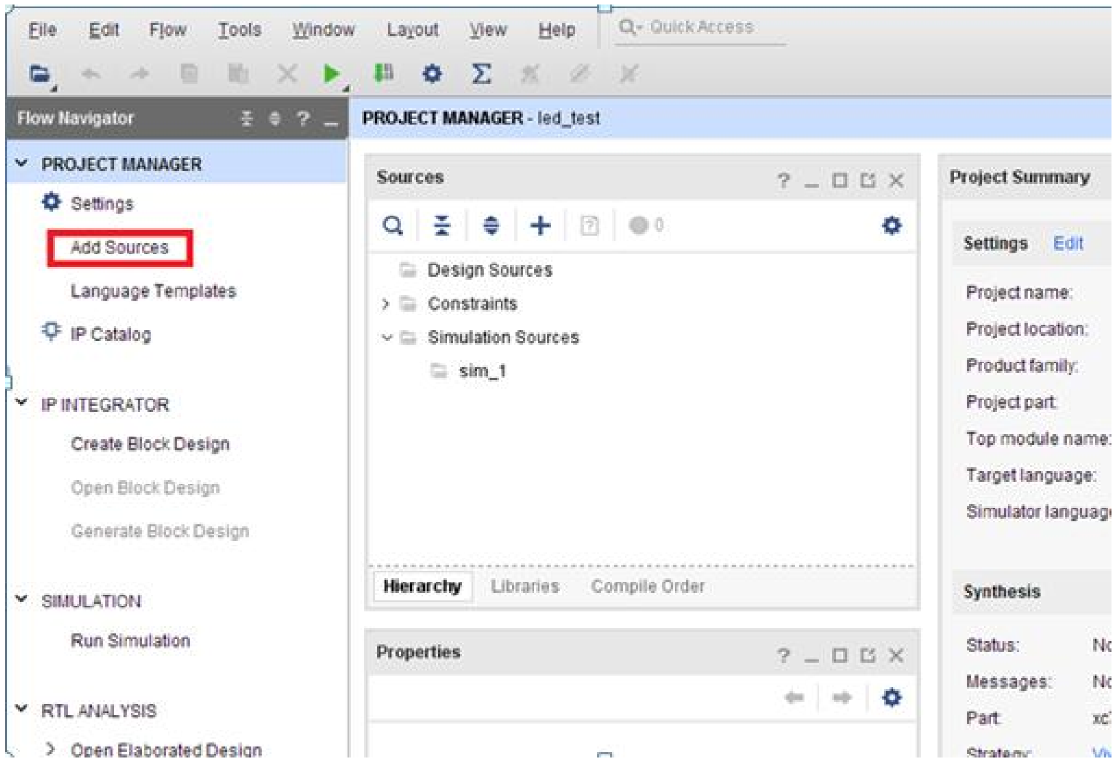
```verilog
//===========================================================================
// Module name: led_test.v
//===========================================================================
`timescale 1ns / 1ps

module led_test
(
  sys_clk_p,      // Differentia system clock 200Mhz input on board
  sys_clk_n,
  rst_n,          // reset ,low active
  led,             // LED,use for control the LED signal on board
  fan_pwm           //fan control
 );

//===========================================================================
// PORT declarations
//===========================================================================

input         sys_clk_p;
input         sys_clk_n;
input         rst_n;
output [3:0]  led;
output fan_pwm;
//define the time counter
reg [31:0]   timer;
reg [3:0]    led;
assign fan_pwm =1'b0;
//===========================================================================
//Differentia system clock to single end clock
//===========================================================================
wire        sys_clk;
 IBUFGDS u_ibufg_sys_clk
     (
      .I  (sys_clk_p),
      .IB (sys_clk_n),
      .O  (sys_clk  )
      );
//===========================================================================
// 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 <= 4'b0000;                  //when the reset signal active
      else if (timer == 32'd49_999_999)    //time counter count to 0.25 sec,LED1 lighten

          led <= 4'b0001;
      else if (timer == 32'd99_999_999)    //time counter count to 0.5 sec,LED2 lighten
      begin
          led <= 4'b0010;
        end
      else if (timer == 32'd149_999_999)   //time counter count to 0.75 sec,LED3 lighten
          led <= 4'b0100;
      else if (timer == 32'd199_999_999)   //time counter count to 1 sec,LED4 lighten
          led <= 4'b1000;
    end

endmodule
```

6. Save after writing the code. Click on the menu “File - Save All Files”

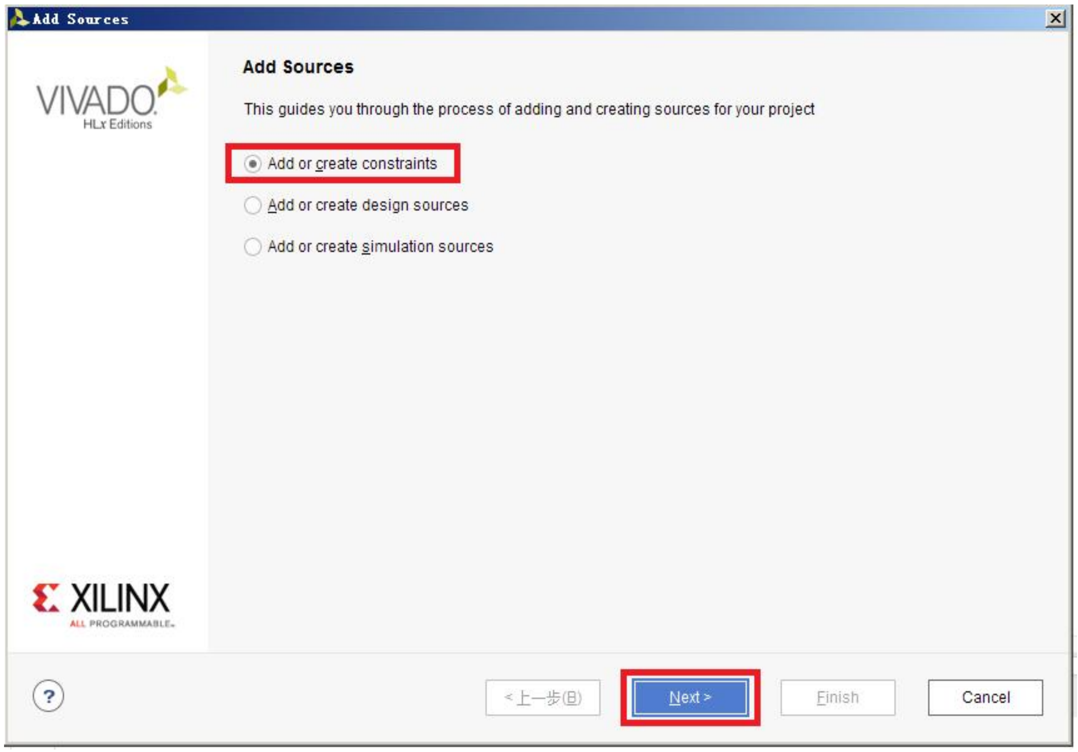
4.3 Add XDC pin constraint file

Unlike ISE software, the constraint file format used by Vivado is an xdc file. The xdc file is mainly used to complete the constraints of the pins, the constraints of the clock, and the constraints of the group. Here we need to allocate the input and output ports in the “led_test.v” program to the real pins of the FPGA. This requires preparing an FPGA pin binding file .xdc, and adding it to the project.

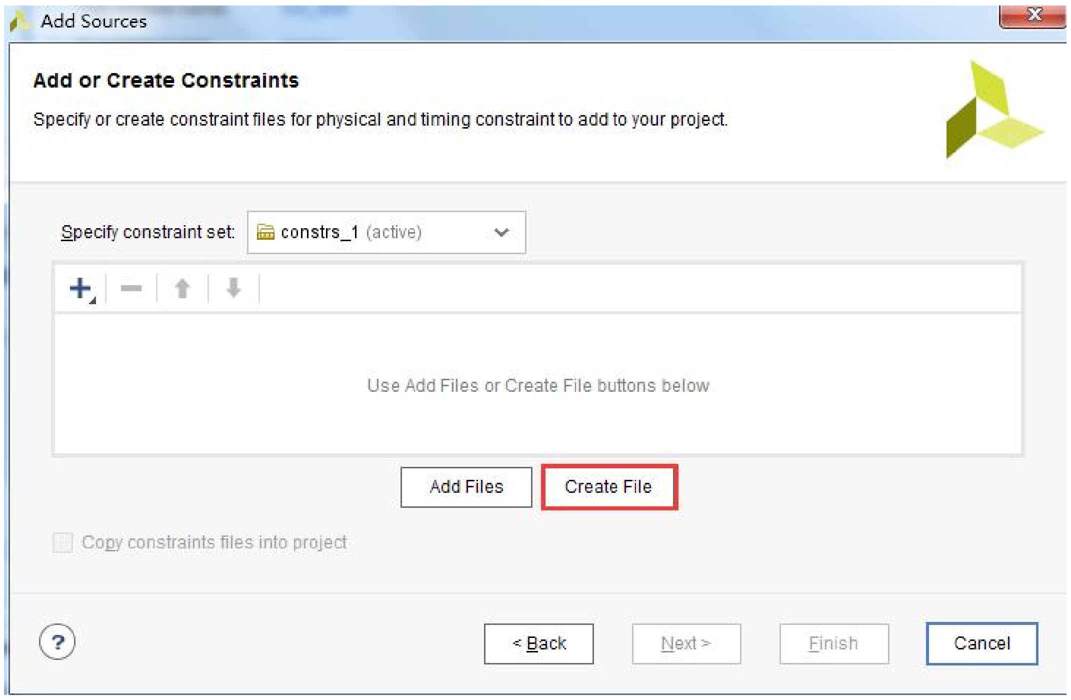
1. Click the “Add Sources” icon under “Project Manager”



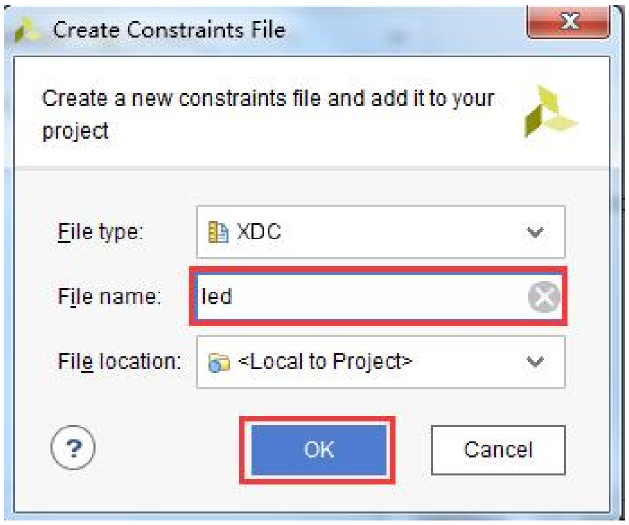
2. Select “Add or create constraints” option and click “Next”


Add Sources
VIVADO HLx Editions
XILINX ALL PROGRAMMABLE
Add Sources
This guides you through the process of adding and creating sources for your project
Add or create constraints
Add or create design sources
Add or create simulation sources
< 上一步(B)
Next >
Finish
Cancel

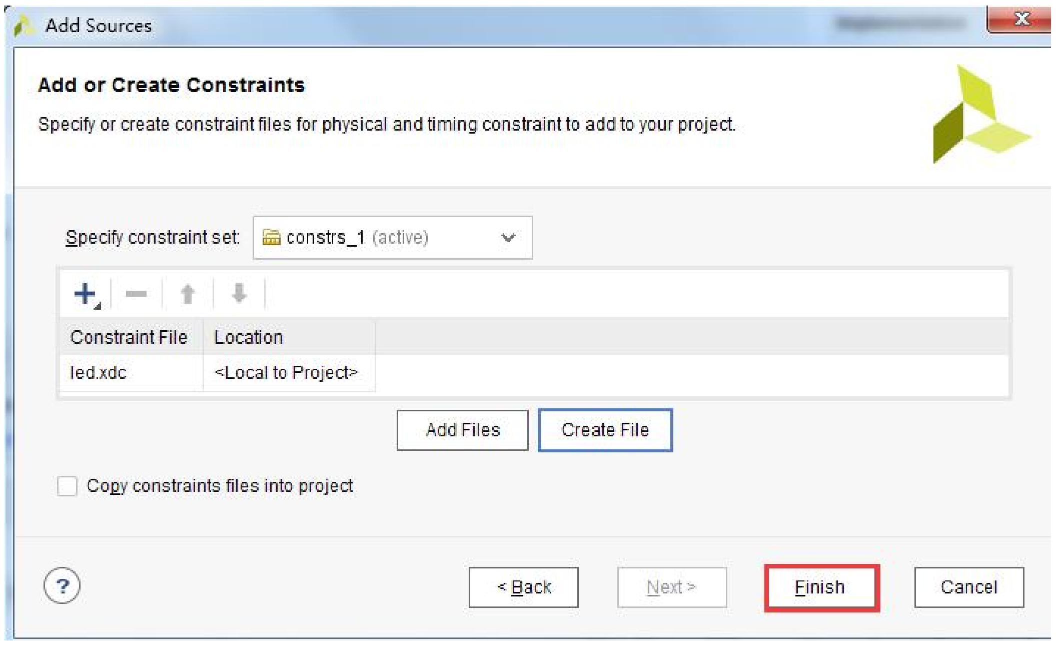
3. Click "Create File" Button


Add Sources
Add or Create Constraints
Specify or create constraint files for physical and timing constraint to add to your project.
Specify constraint set: constrs_1 (active)
Use Add Files or Create File buttons below
Add Files
Create File
Copy constraints files into project
< Back
Next >
Finish
Cancel

In the pop-up dialog box, select "File type" is "XDC", "File name" is "led", click the OK button.

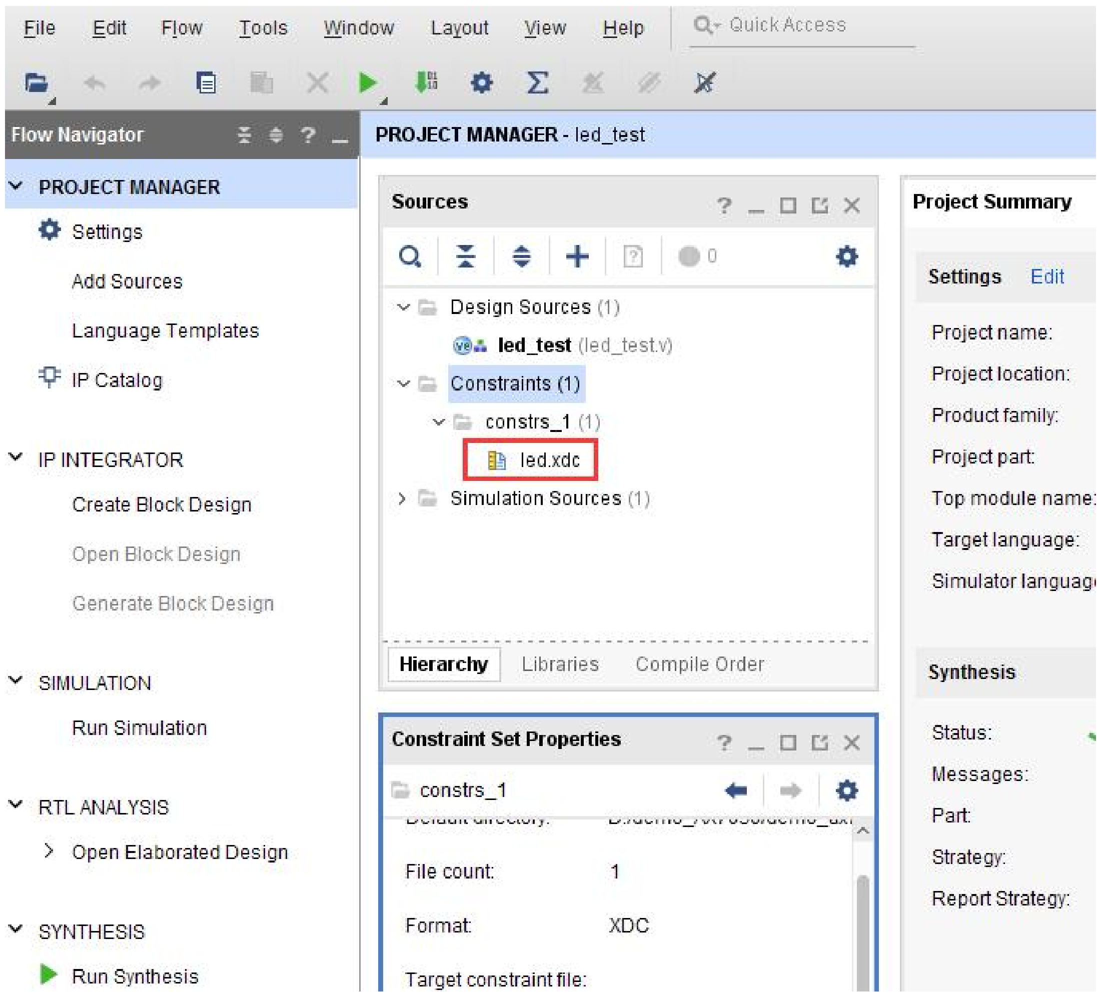


4. Click “Finish”


Add Sources
Add or Create Constraints
Specify or create constraint files for physical and timing constraint to add to your project.
Specify constraint set:
constrs_1 (active)
Add Files
Create File
Copy constraints files into project
< Back
Next >
Finish
Cancel

Constraint File	Location
led.xdc	<Local to Project>

Then, there is already a “led.xdc” file in the “constrs_1” directory of the Constraints directory under the “Project Manager” interface.



5. Double-click to open the led.xdc file and add the following pin definitions to this file.

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

set_property CFGBVS VCCO [current_design]

set_property CONFIG_VOLTAGE 3.3 [current_design]

set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]

set_property BITSTREAM.CONFIG.UNUSEDPIN Pullup [current_design]

############## 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 AG27 [get_ports rst_n]
set_property IOSTANDARD LVCMOS25 [get_ports rst_n]

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

set_property IOSTANDARD LVCMOS25 [get_ports fan_pwm]

set_property PACKAGE_PIN AE26 [get_ports fan_pwm]

##############LED define#################

set_property PACKAGE_PIN A22 [get_ports {led[0]}]

set_property IOSTANDARD LVCMOS15 [get_ports {led[0]}]


set_property PACKAGE_PIN C19 [get_ports {led[1]}]

set_property IOSTANDARD LVCMOS15 [get_ports {led[1]}]


set_property PACKAGE_PIN B19 [get_ports {led[2]}]

set_property IOSTANDARD LVCMOS15 [get_ports {led[2]}]


set_property PACKAGE_PIN E18 [get_ports {led[3]}]

set_property IOSTANDARD LVCMOS15 [get_ports {led[3]}]
```

The first two sentences of the XDC file are the voltage of the CFGBVS pin and the voltage of the configuration circuit, because the CFGBVS pin on the development board is pulled up to 3.3V, which is the VCCIO of BANK0. In addition, the voltage of the configuration circuit is 3.3V. So here are configured as VCCIO and 3.3V respectively.

Let's introduce the syntax of the most basic XDC. The normal IO port only needs to constrain the pin number and voltage. The pin constraints are as follows:

set_property PACKAGE_PIN "Pin nuber" [get_ports “Port name”]

The constraints of the level signal are as follows:

set_property IOSTANDARD "Voltage" [get_ports “Port Name”]

Here you need to pay attention to the case of the text. If the port name is an array, enclose it with { }. The port name must be the same as the name in the source code, and the port name cannot be the same as the keyword.

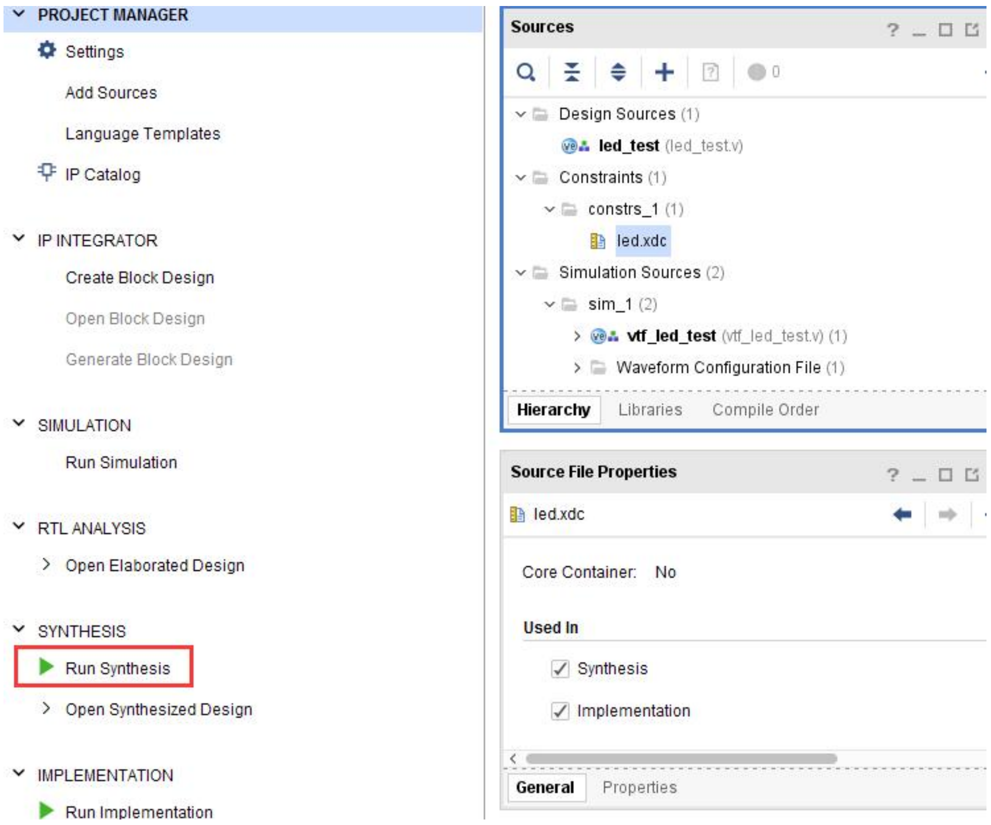
The clock port can also define clock cycle constraints. For example, we define the input differential clock in the XDC with a clock period of 20ns. The clock cycle constraint method is as follows:

create_clock -period " cycle " [get_ports "Port name"]

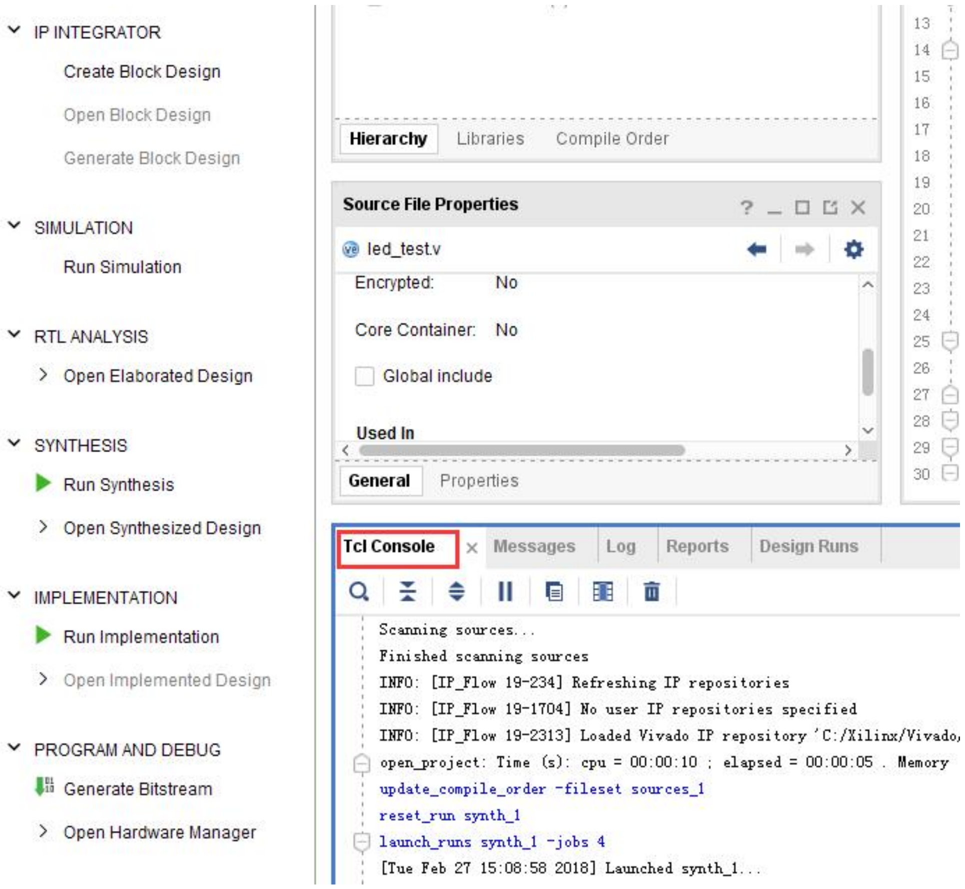
When finished, select the menu "File->Save all files" to save all files.

4.4 Compile

Click "Run Synthesis" to start synthesizing and generating netlist files:

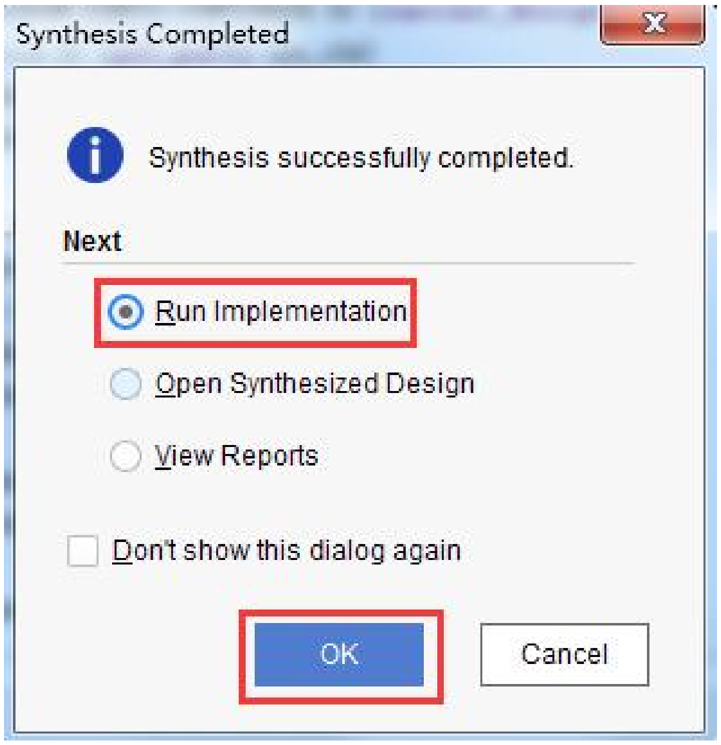


Some status information can be seen in the "Tcl Console" window or the "Messages" window.



```
Scanning sources...
Finished scanning sources
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository 'C:/Xilinx/Vivado,
open_project: Time (s): cpu = 00:00:10 ; elapsed = 00:00:05 . Memory
update_compile_order -fileset sources_1
reset_run synth_1
launch_runs synth_1 -jobs 4
[Tue Feb 27 15:08:58 2018] Launched synth_1...
```

After Synthesis is completed, a small window will pop up. You can start the Layout and Wiring by clicking "Run Implementation" here:


Synthesis Completed

Synthesis successfully completed.

Next

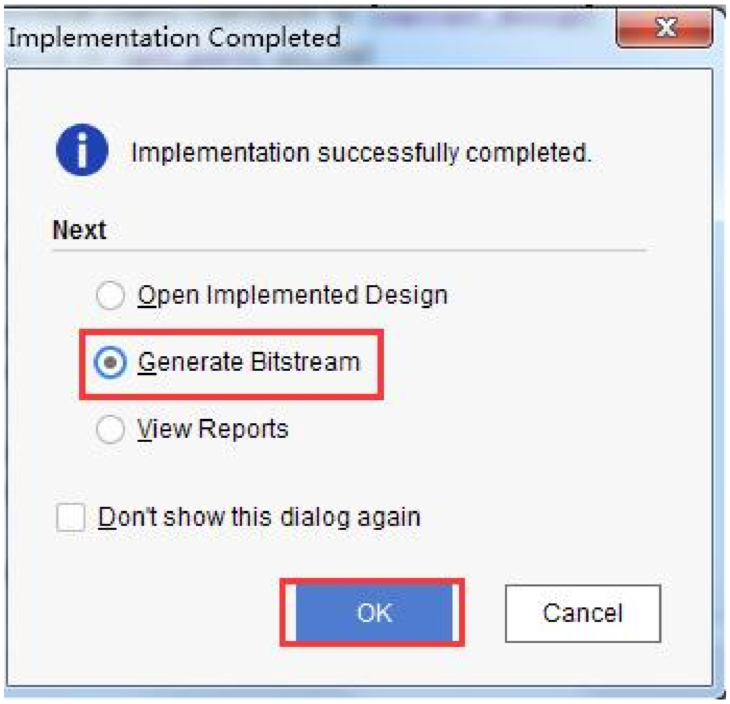
[✓ Run Implementation]

Open Synthesized Design

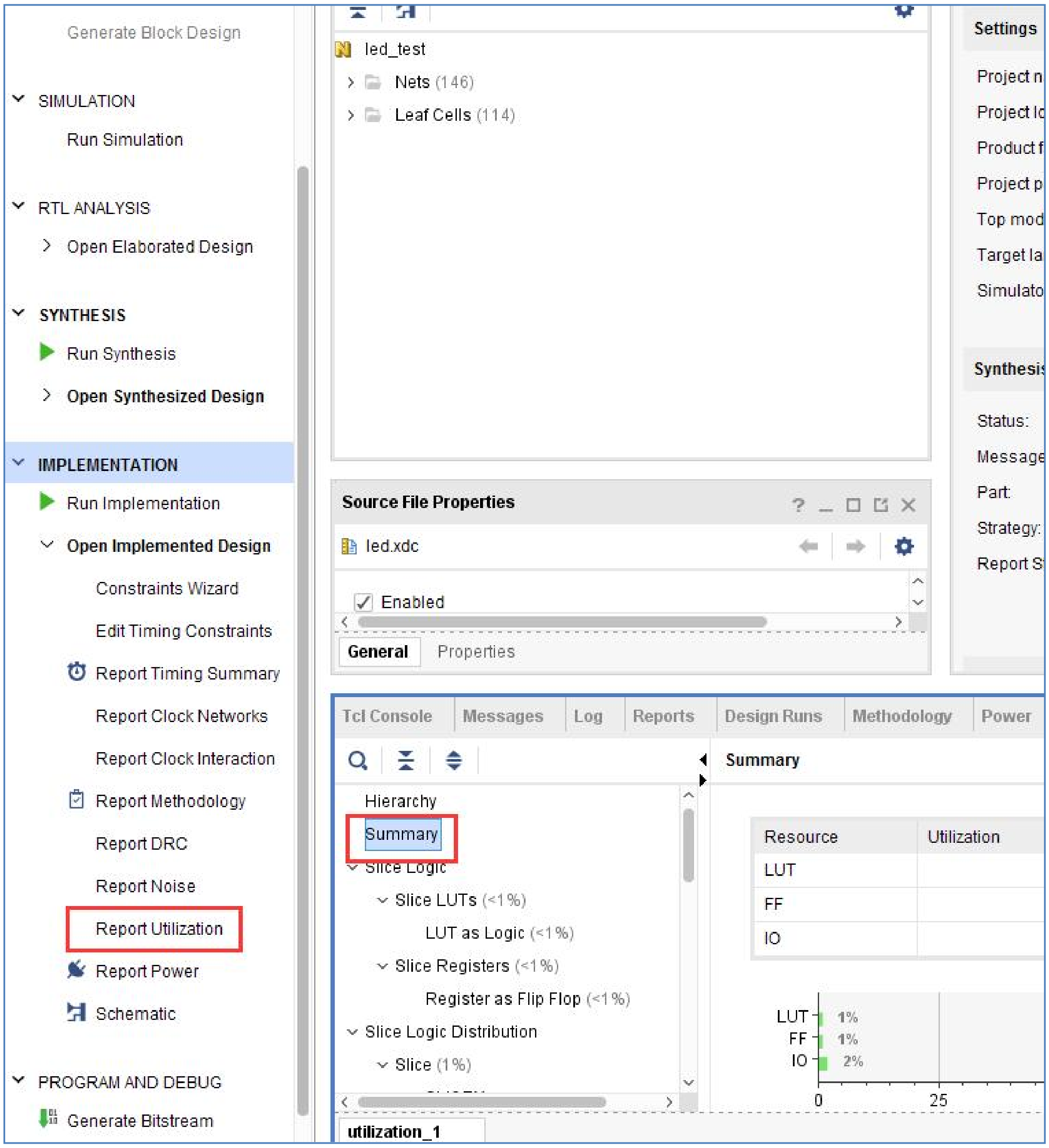
View Reports

Don't show this dialog again

A prompt window will pop up when the wiring is completed. You can generate a bit file by clicking the "Generate Bitstream" here



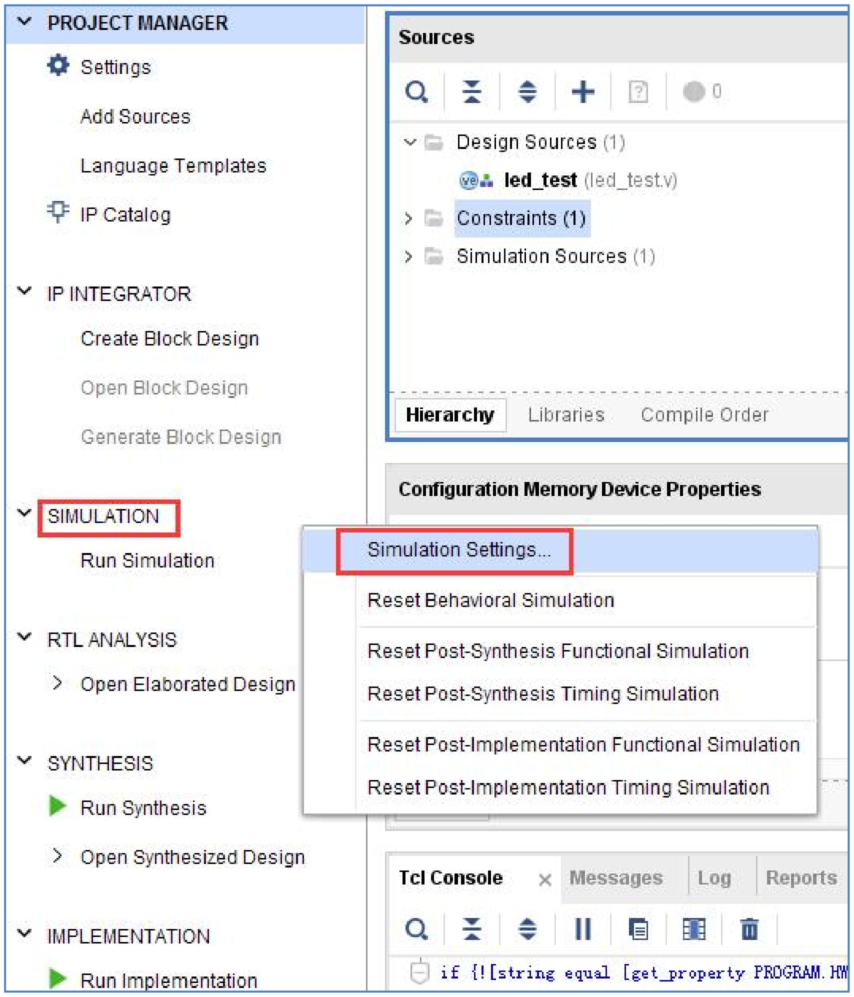
After the Bit file is generated, we can open the "Table" of the "Project Summary" page to check the actual resource usage on the FPGA board. Because our "led_test program" is relatively simple, only four resources are used: LUT (Lookup Table), FF (Flip Flop) Register), IO (pin) and BUFG (clock buffer).



4.5 Vivado simulation verification

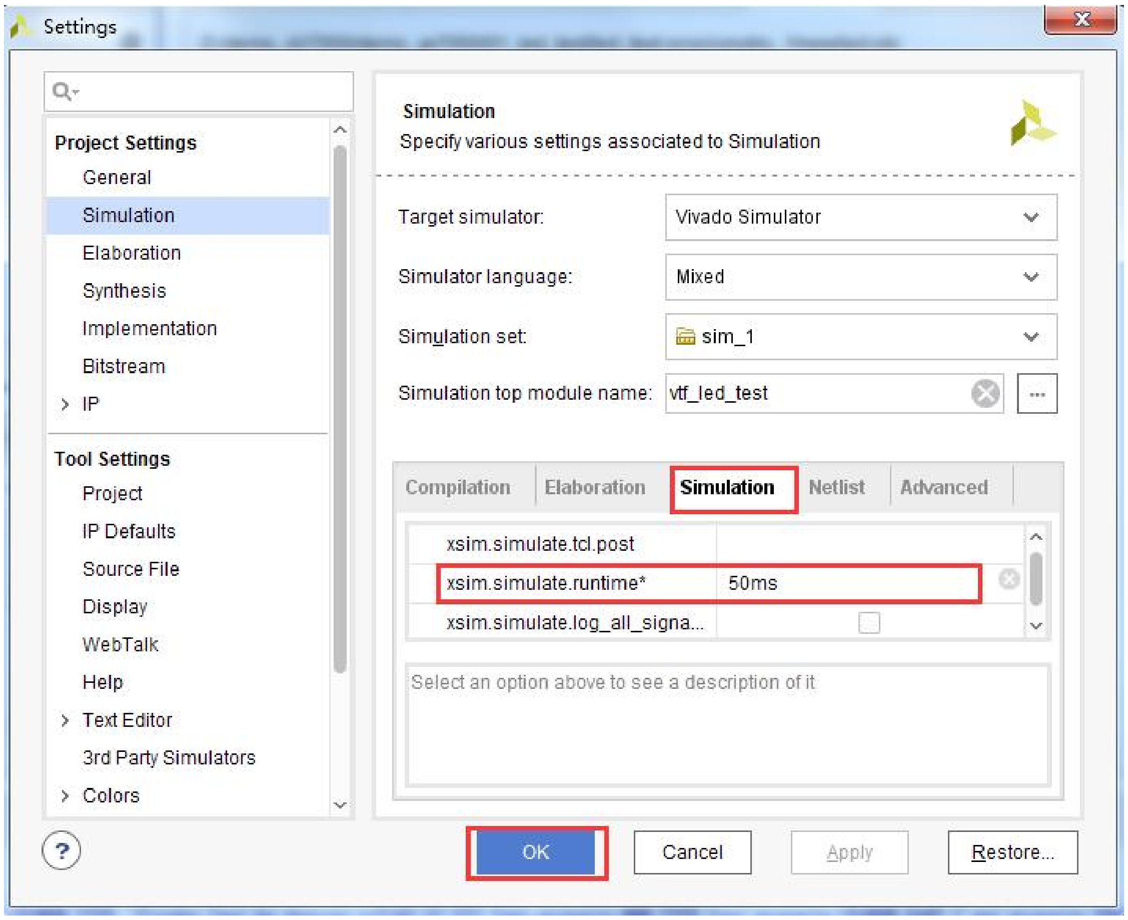
The simulation tool Vivado uses the output waveform to verify that the flow light programming results are consistent with our expectations. Specific steps are as follows:

1. Set up the simulation configuration for Vivado and right click on “Simulation Settings” in “SIMULATION”



```
if {![string equal [get_property PROGRAM.HW
```

2. In the “Simulation Settings” window, perform the following configuration to configure it. Set it to 50ms (set it as needed), and press OK by default.



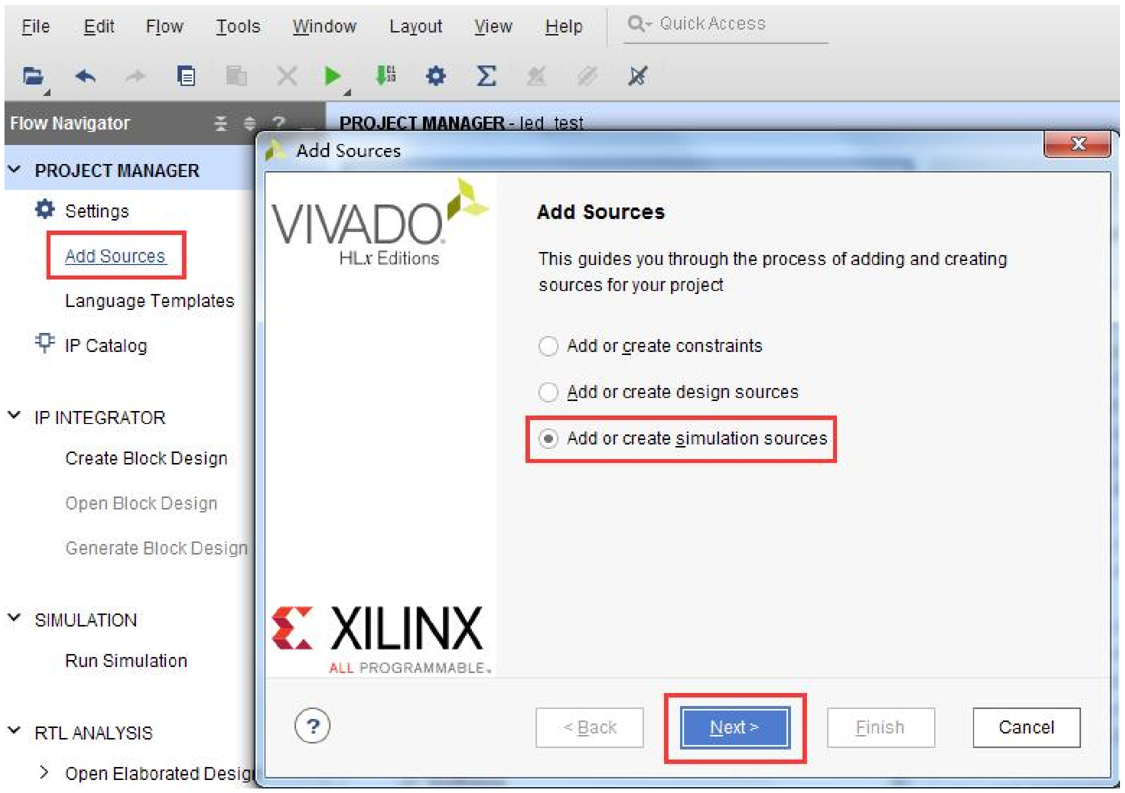
Simulation
Specify various settings associated to Simulation

Target simulator:	Vivado Simulator
Simulator language:	Mixed
Simulation set:	sim_1
Simulation top module name:	vtf_led_test

xsim.simulate.tcl.post	
xsim.simulate.runtime*	50ms
xsim.simulate.log_all_signa...	☐

Select an option above to see a description of it

3. Add the stimulus test file, click the "Add Sources" icon under "Project Manager", click on the settings below and click Next.

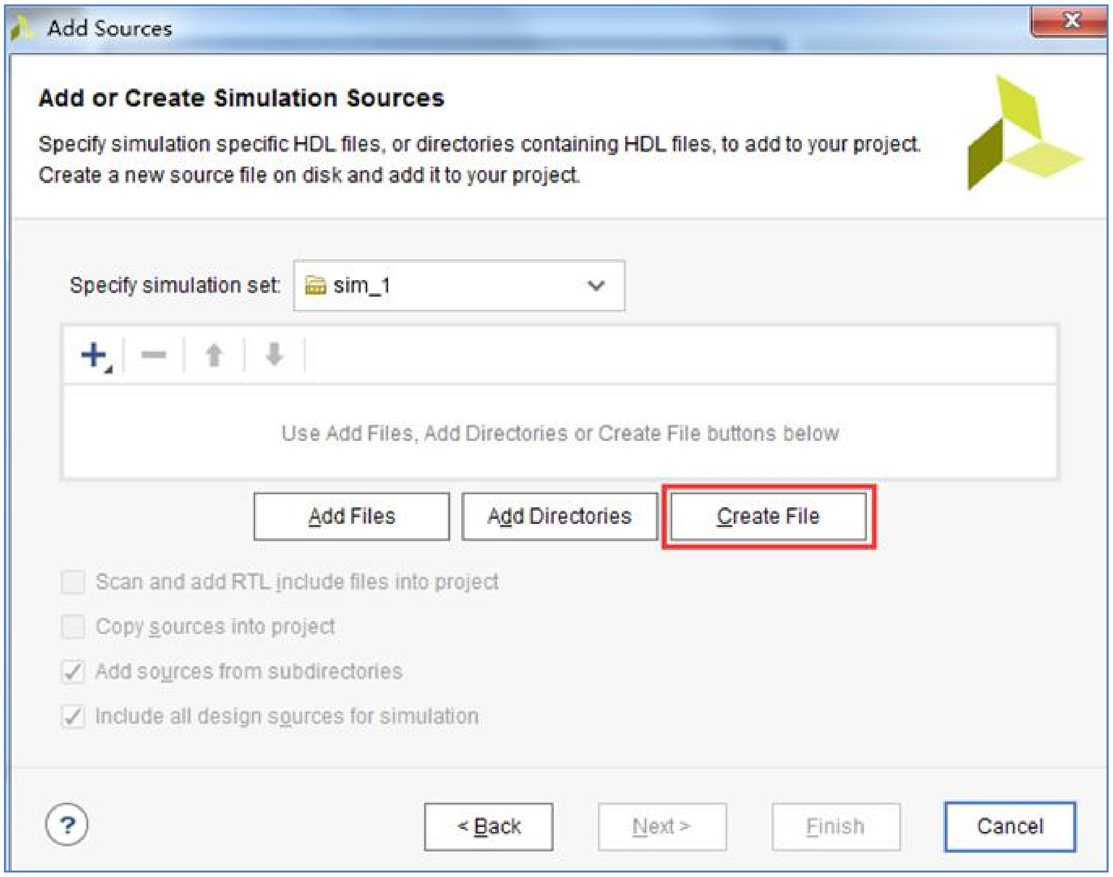


Add Sources

This guides you through the process of adding and creating sources for your project

( ) Add or create constraints
( ) Add or create design sources
[✓ Add or create simulation sources]

4. Click “Create File” to generate a simulation stimulus file.



Add or Create Simulation Sources

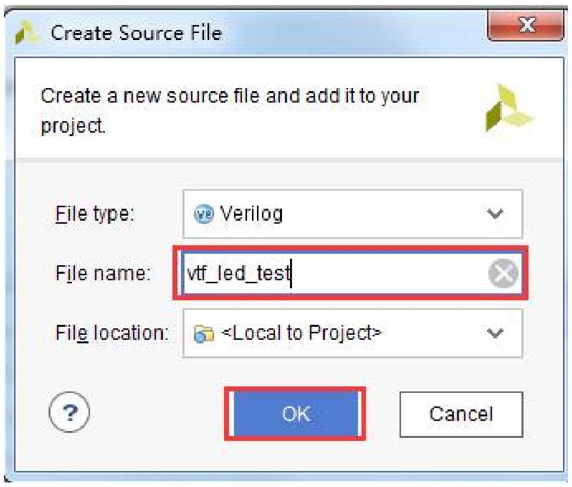
Specify simulation specific HDL files, or directories containing HDL files, to add to your project. Create a new source file on disk and add it to your project.

Specify simulation set: sim_1

Use Add Files, Add Directories or Create File buttons below

☐ Scan and add RTL include files into project
☐ Copy sources into project
✓ Add sources from subdirectories
✓ Include all design sources for simulation

5. Enter the name of the stimulus file in the pop-up dialog box, here we enter the name “vtf_led_test”.



Create Source File

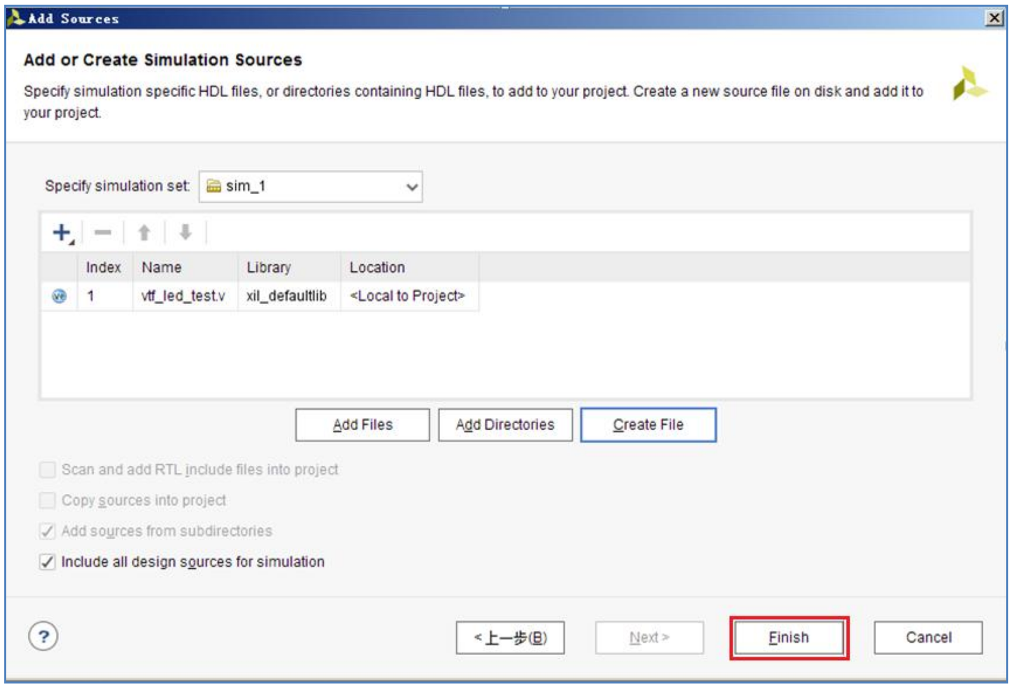
Create a new source file and add it to your project.

File type: Verilog

File name: vtf_led_test

File location: <Local to Project>

6. Click the “Finish” button to return.



Add Sources

Add or Create Simulation Sources

Specify simulation specific HDL files, or directories containing HDL files, to add to your project. Create a new source file on disk and add it to your project.

Specify simulation set: sim_1

Index	Name	Library	Location
1	vtf_led_test.v	xil_defaultlib	<Local to Project>

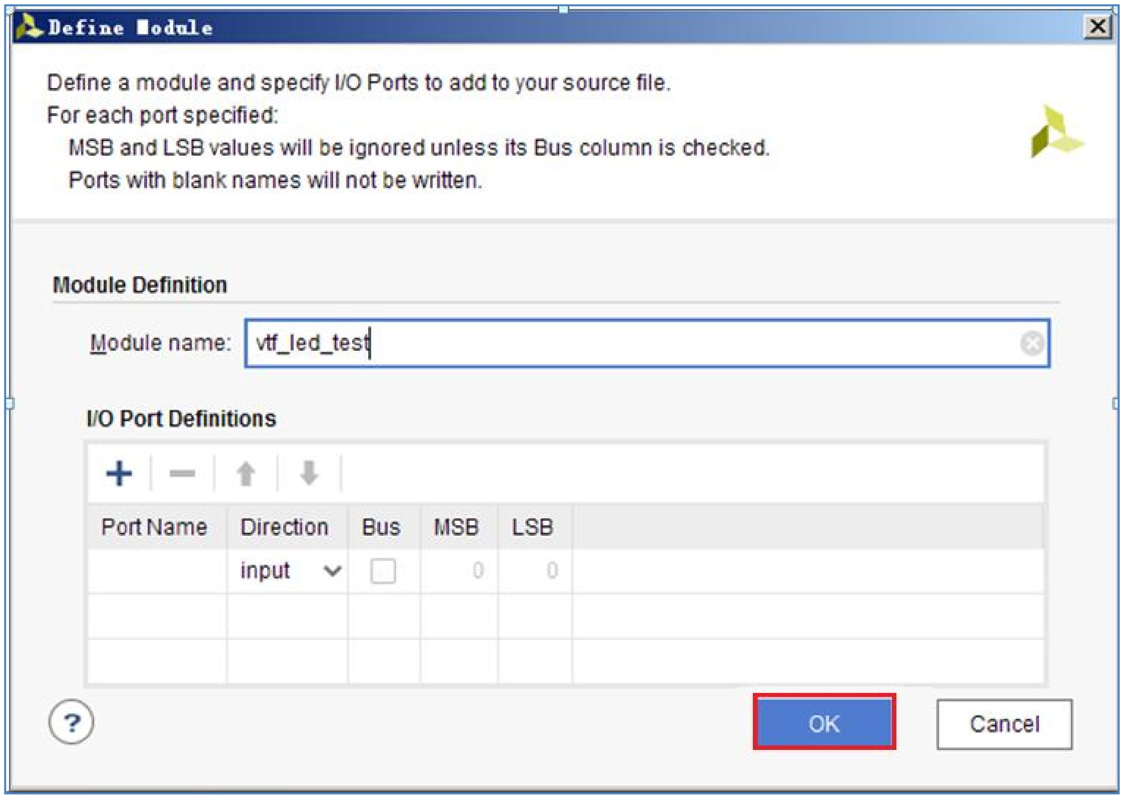
Scan and add RTL include files into project

Copy sources into project

✓ Add sources from subdirectories

✓ Include all design sources for simulation

Here we do not add “IO Ports” first, click OK.


Define Module

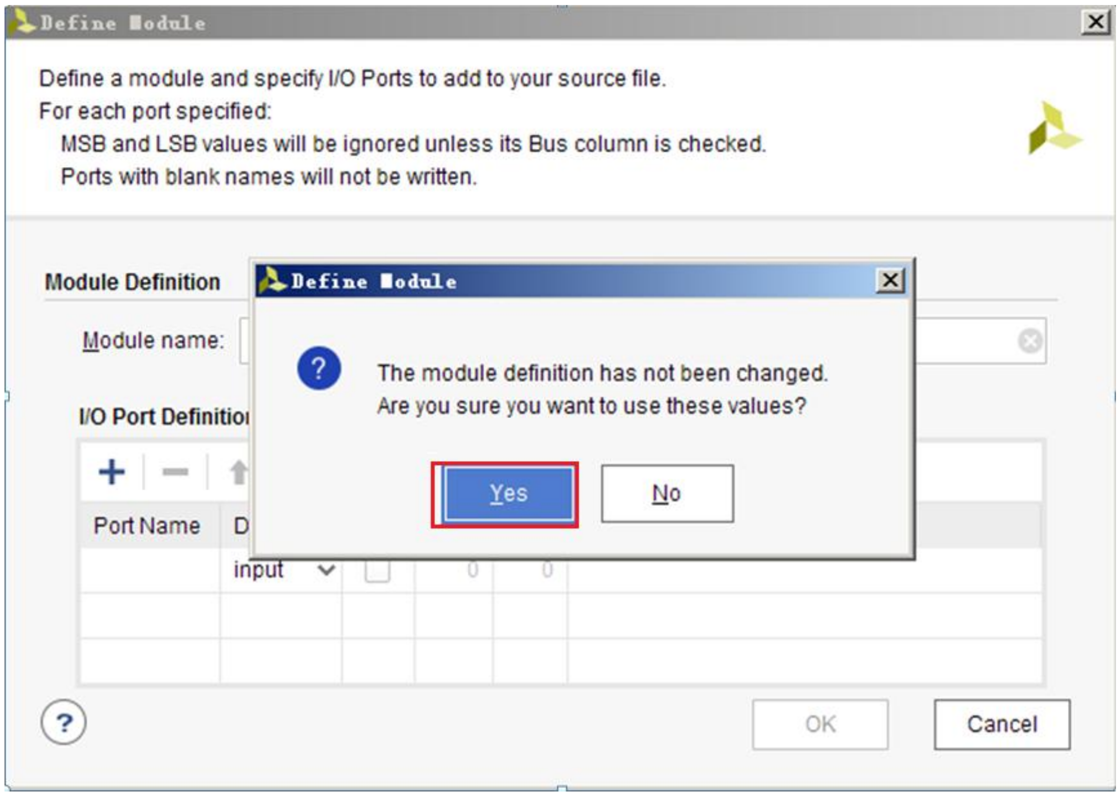
Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

Module name: vtf_led_test

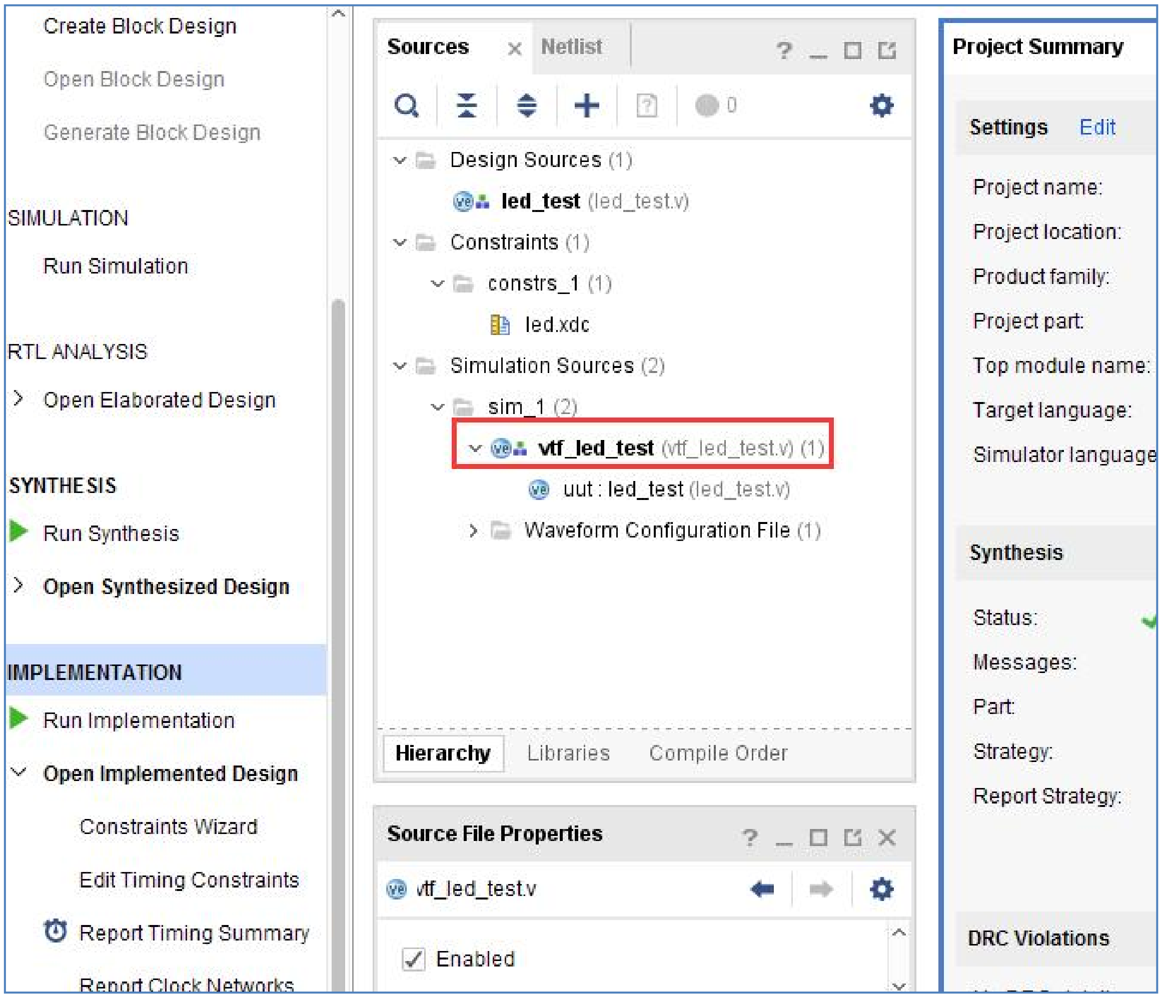
I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
	input	☐	0	0


Define Module

The module definition has not been changed.
Are you sure you want to use these values?

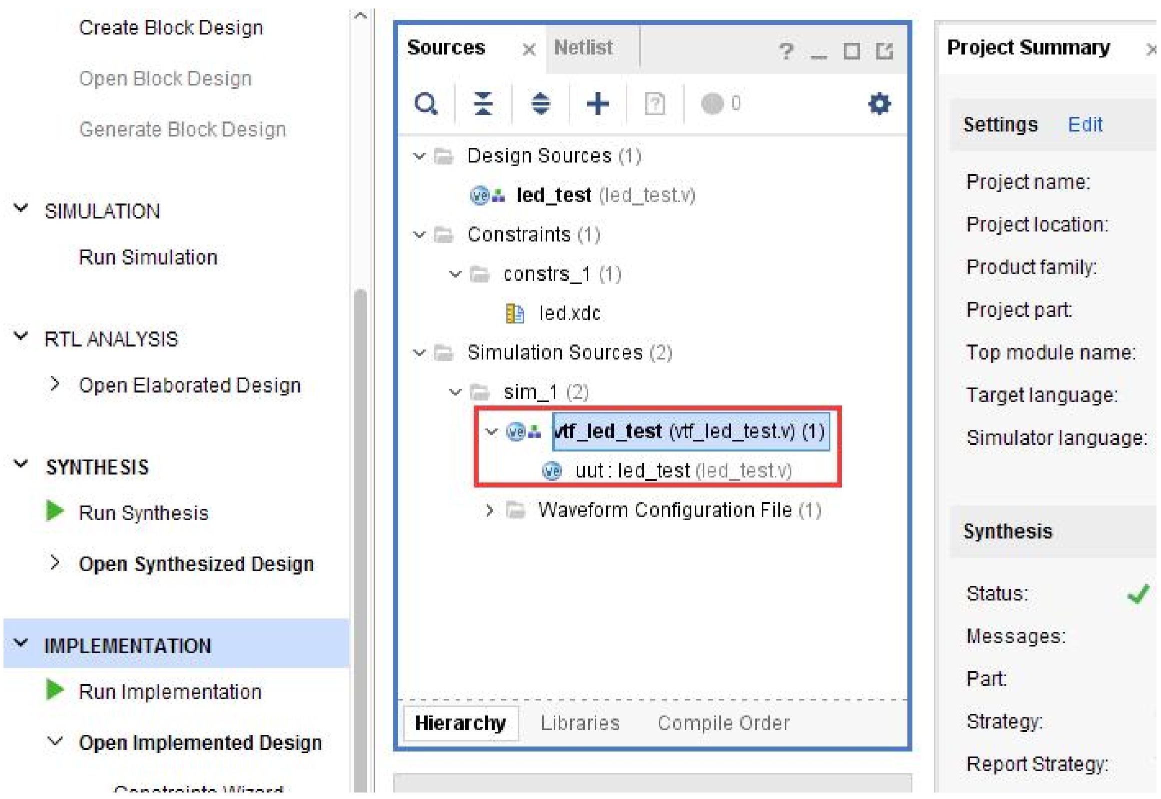
The "vtf_led_test" file just added in the "Simulation Sources" directory. Double-click to open this file, you can see that there is only the definition of the module name, nothing else.



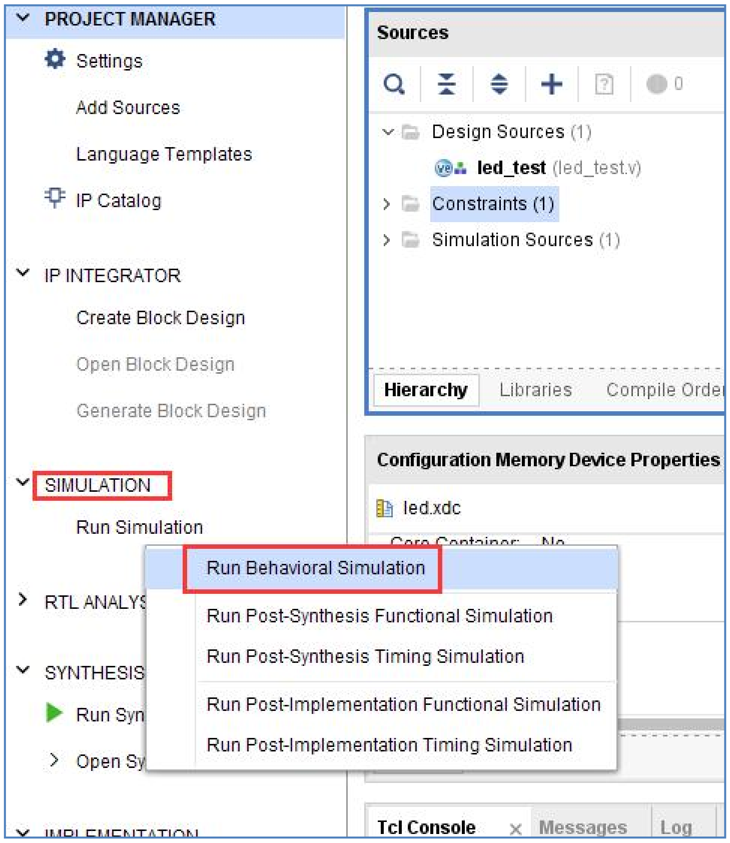
7. Next we need to write the contents of this "vtf_led_test.v" file. First define the input and output signals, then you need to instantiate the "led_test" module to make the "led_test" program part of the test program. Add the reset and clock excitation. The completed vtf_led_test.v file is as follows:

```
`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 [3:0] 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. After writing and saving, "vtf_led_test.v" automatically becomes the top layer of this simulation Hierarchy. Below it is the design file "led_test.v"

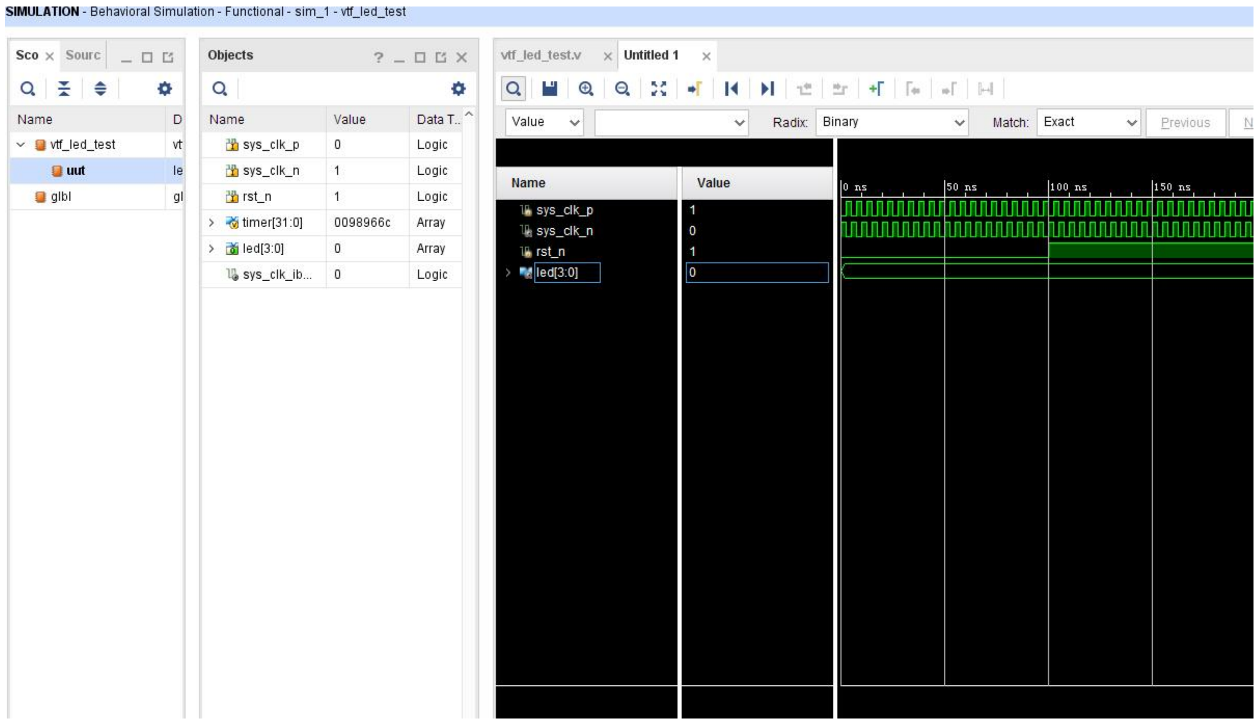


9. Click the "Run Simulation" button and select "Run Behavioral Simulation". Here we can do the behavioral level simulation.



If there are no errors, the simulation software in Vivado is working.

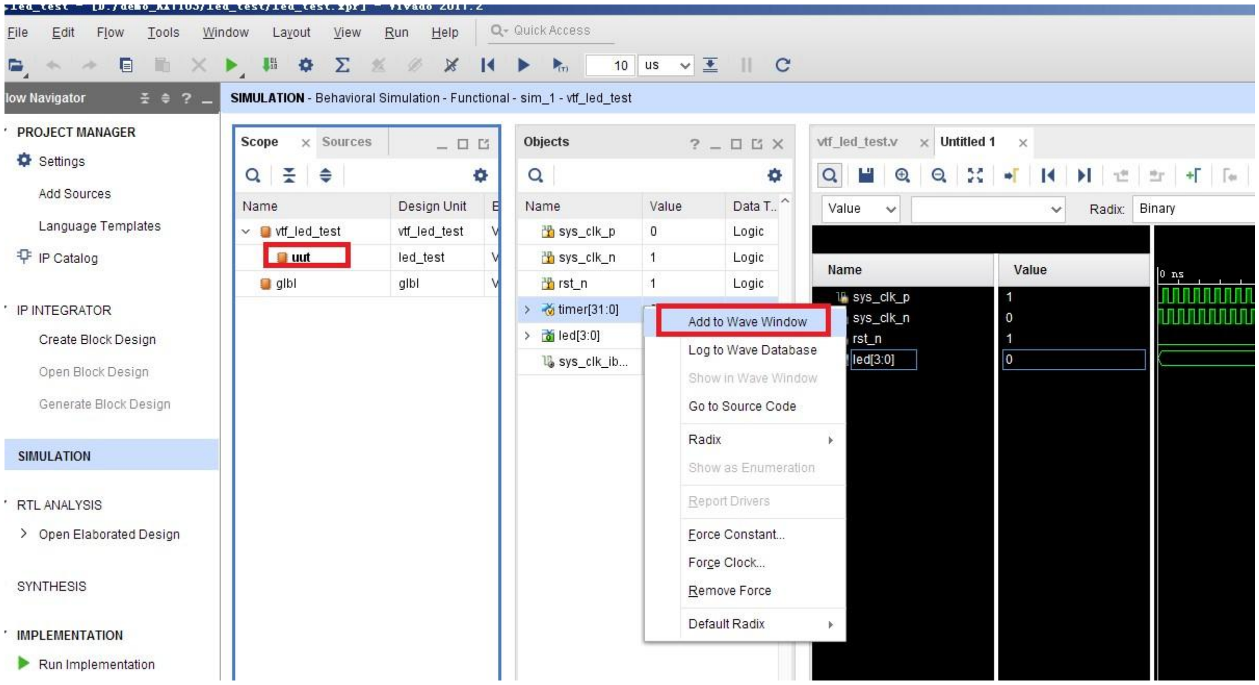
10. After popping up the simulation interface, as shown in the following figure, the interface is a 50ms waveform that the simulation software automatically runs to the simulation setup.


SIMULATION - Behavioral Simulation - Functional - sim_1 - vtf_led_test

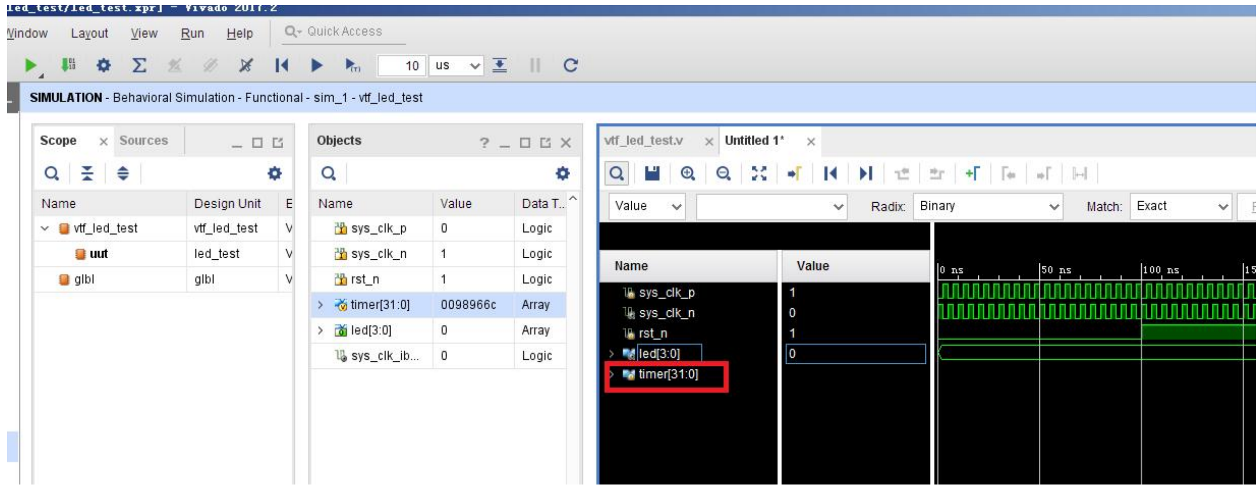
Name	Value	Data T..
sys_clk_p	0	Logic
sys_clk_n	1	Logic
rst_n	1	Logic
timer[31:0]	0098966c	Array
led[3:0]	0	Array
sys_clk_ib...	0	Logic

Name	Value
sys_clk_p	1
sys_clk_n	0
rst_n	1
led[3:0]	0

Since the state of “LED[3:0]” design in the program changes for a long time, and the simulation is time consuming, observe the “timer[31:0]” counter change here. Put it in Wave to observe (click “uut” under the “Scope” interface, then right click on the “timer” under the “Objects” interface, select “Add Wave Window” in the pop-up drop-down menu).

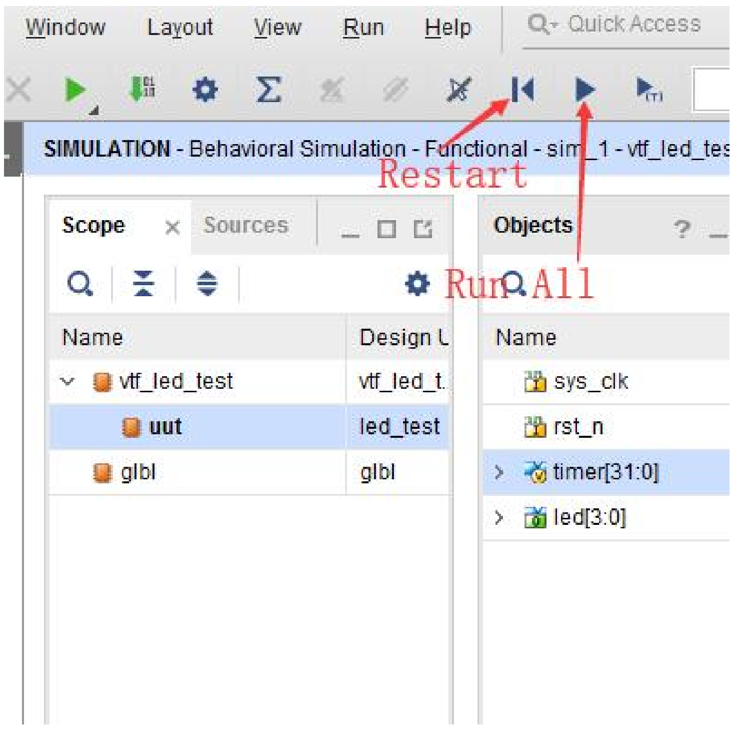


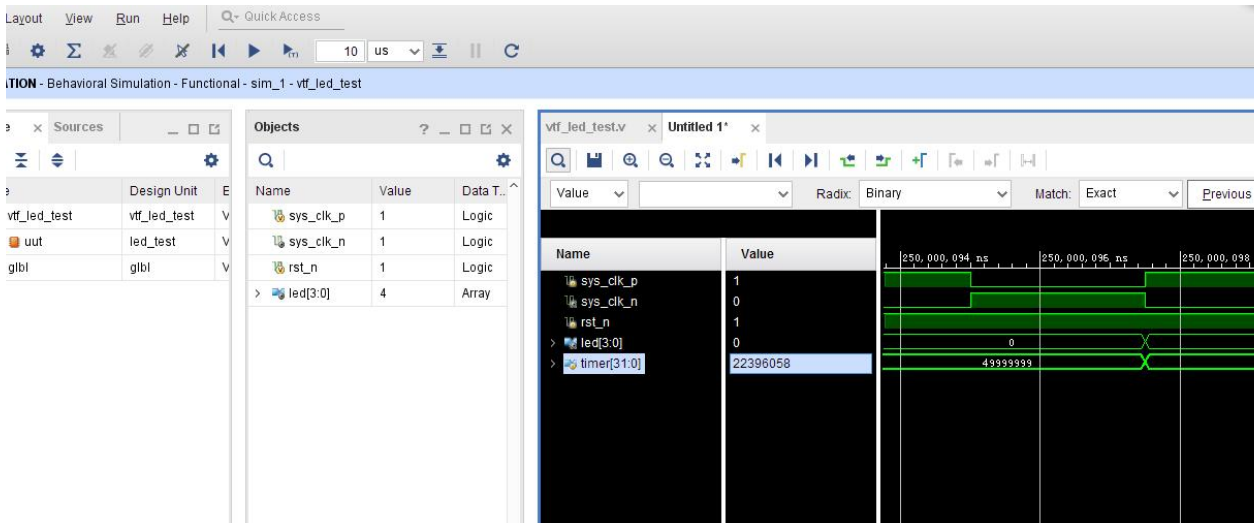
After adding, the “timer” is displayed on the waveform interface of Wave, as shown in the figure below.


SIMULATION - Behavioral Simulation - Functional - sim_1 - vtf_led_test
Scope
Sources
Objects
vtf_led_test.v
Untitled 1*
Radix: Binary
Match: Exact

Name	Value	Data T...
sys_clk_p	0	Logic
sys_clk_n	1	Logic
rst_n	1	Logic
timer[31:0]	0098966c	Array
led[3:0]	0	Array
sys_clk_ib...	0	Logic

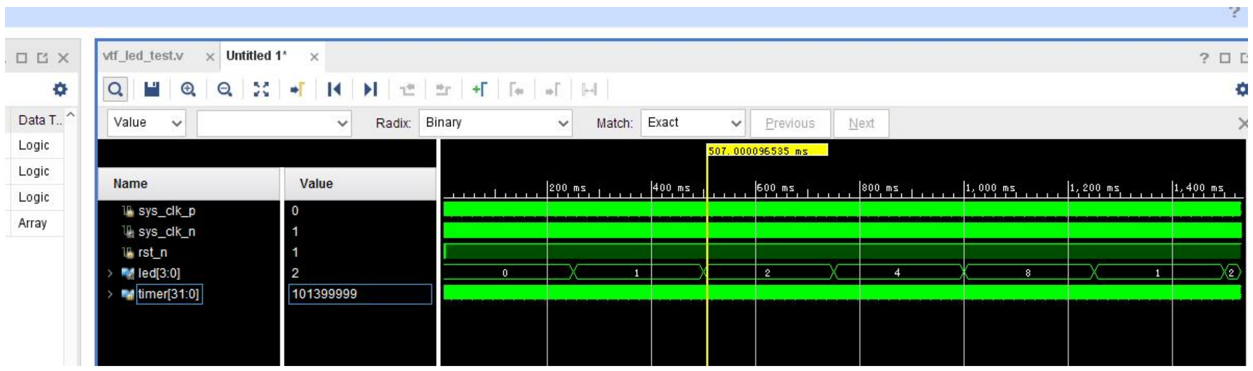
11. Click the “Restart” button labeled below to reset it, then click the “Run All” button. (Need patience!!!), you can see that the simulation waveform matches the design.





Name	Value	Data T..
sys_clk_p	1	Logic
sys_clk_n	1	Logic
rst_n	1	Logic
led[3:0]	4	Array

Name	Value
sys_clk_p	1
sys_clk_n	0
rst_n	1
led[3:0]	0
timer[31:0]	22396058

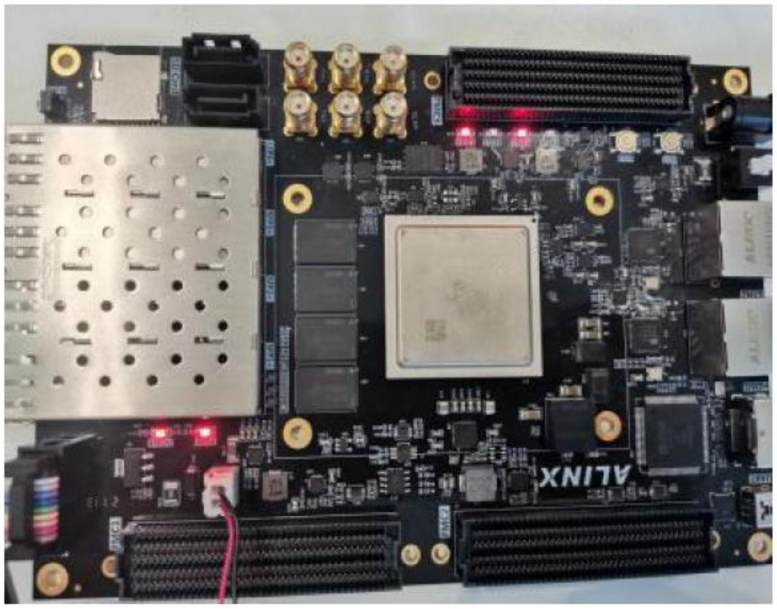


Name	Value
sys_clk_p	0
sys_clk_n	1
rst_n	1
led[3:0]	2
timer[31:0]	101399999

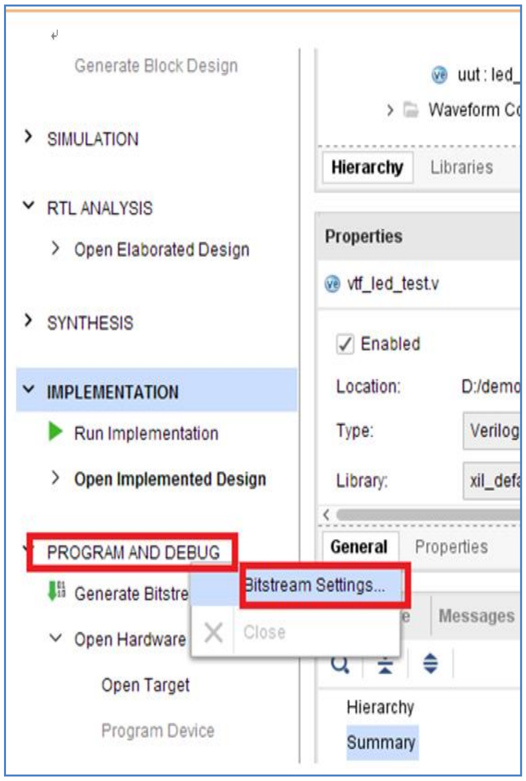
We can see that the LED signal will change to 1 one by one, indicating that the LED1~LED4 lights are extinguished one by one.

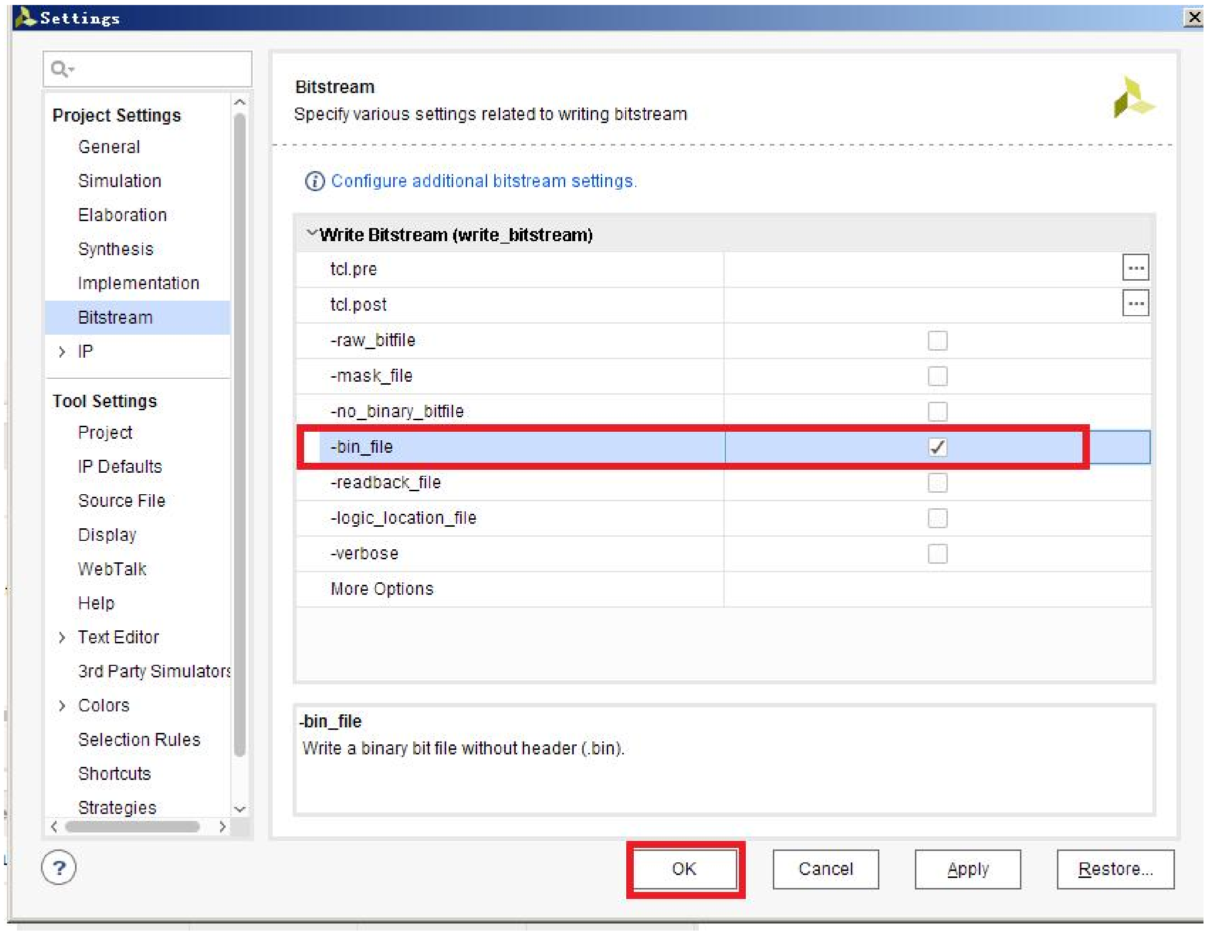
4.6 Download and debug

After the previous compilation and simulation, we can download the “bit” file to the “FPGA” chip and see the actual operation of the “LED”. Connect the hardware before downloading and debugging, connect the “JTAG” downloader to the development board, and then plug in the power adapter on the development board



1. Settings are also required before downloading: right-click on “PROGRAM AND DEBUG” and set it as shown below:





Bitstream
Specify various settings related to writing bitstream

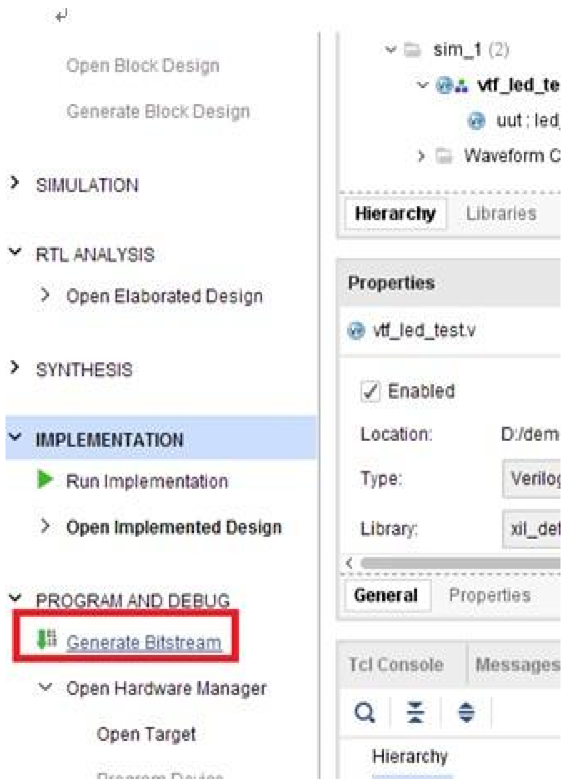
Configure additional bitstream settings.

Write Bitstream (write_bitstream)

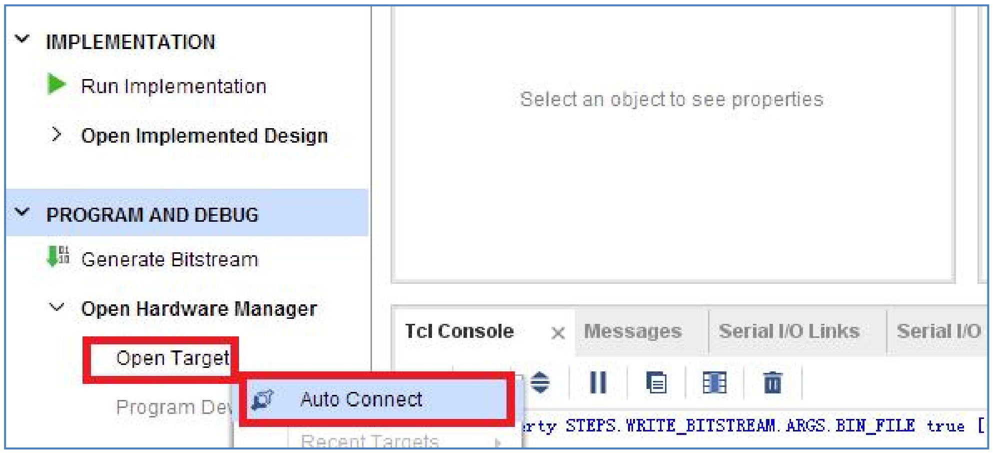
tcl.pre	
tcl.post	
-raw_bitfile	☐
-mask_file	☐
-no_binary_bitfile	☐
-bin_file	✓
-readback_file	☐
-logic_location_file	☐
-verbose	☐
More Options	

-bin_file
Write a binary bit file without header (.bin).

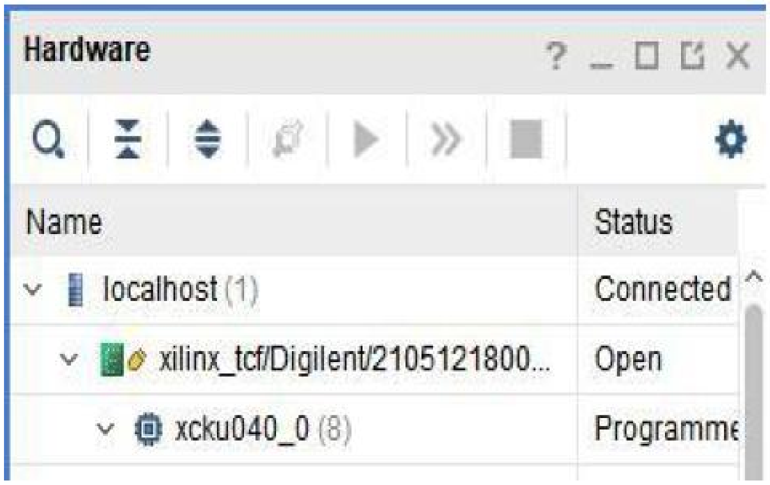
After the setup is completed, click “Generate Bitstream” to generate the “bit” and “bin files”.



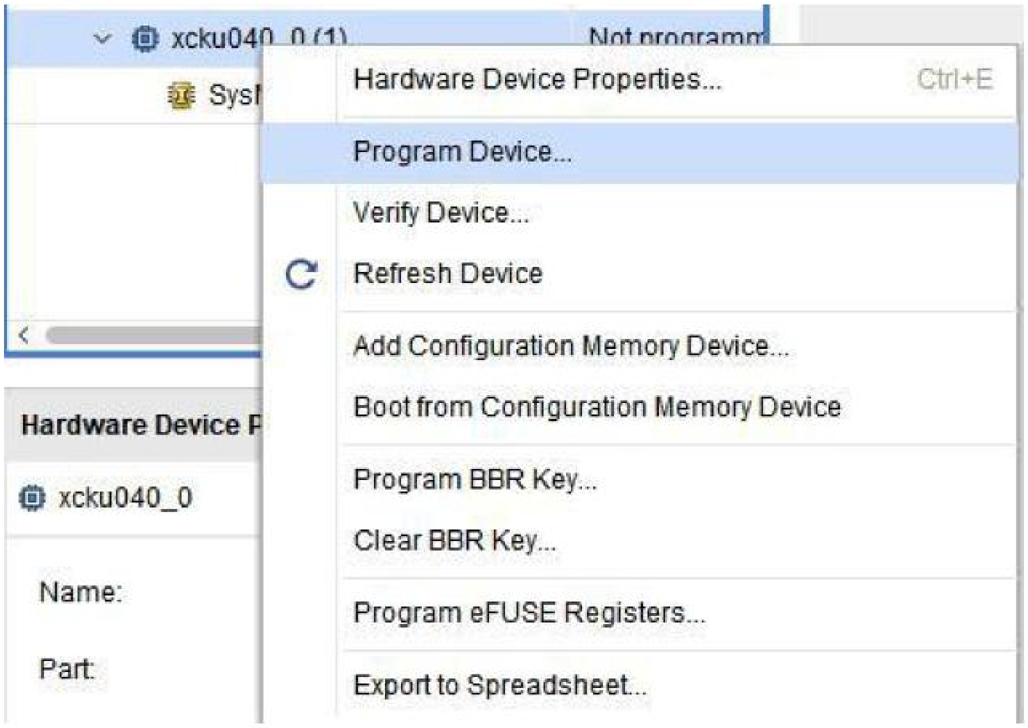
2. Click the "Open target" ->"Auto Connect". The icon of xc7k325t_0, it will be displayed under the hardware interface, indicating that the JTAG connection has been established.



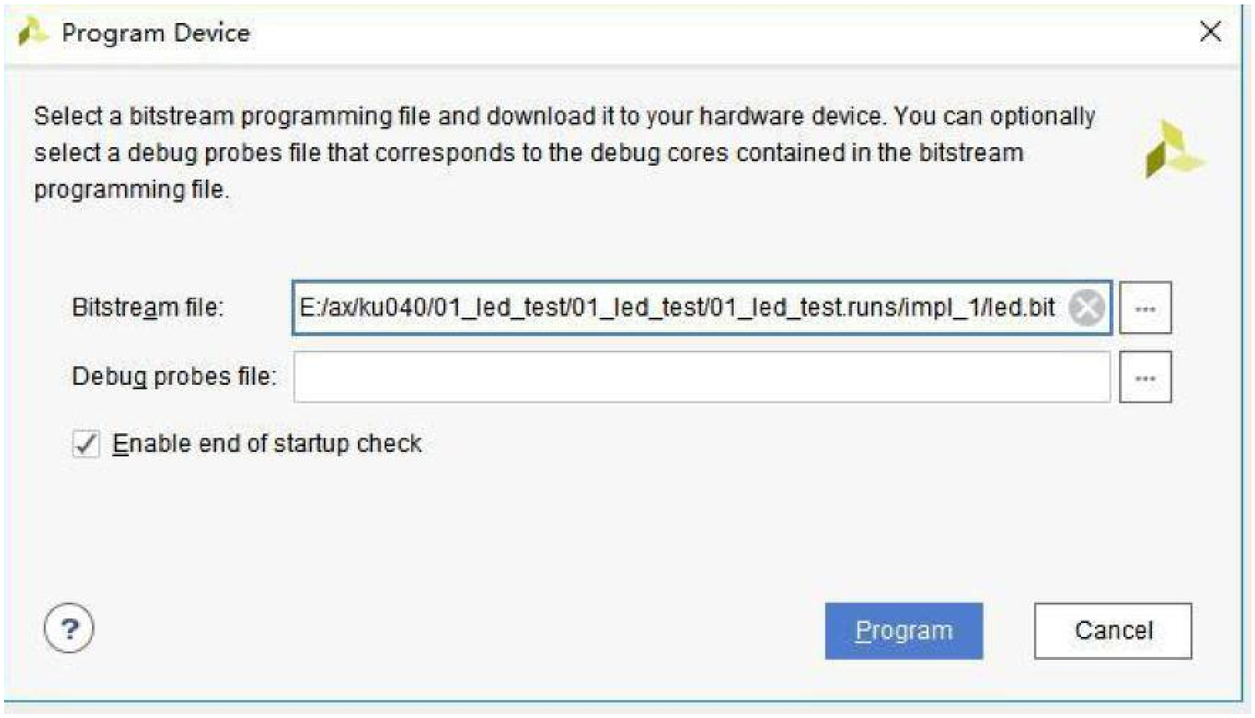
```
rty STEPS.WRITE_BITSTREAM.ARGS.BIN_FILE true [
```



Right click on xcku040, select "Program Device" item in the pop-up options.



In the "Program Device" dialog box that pops up, select the "bit file" generated by the "led_test" project and click the "Program" button to program the FPGA.



Program Device

Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.

Bitstream file: E:/ax/ku040/01_led_test/01_led_test/01_led_test.runs/impl_1/led.bit

Debug probes file:

✓ Enable end of startup check

After the programming is completed, Then, we can see that the two LED lights on the development board are already working as a water light

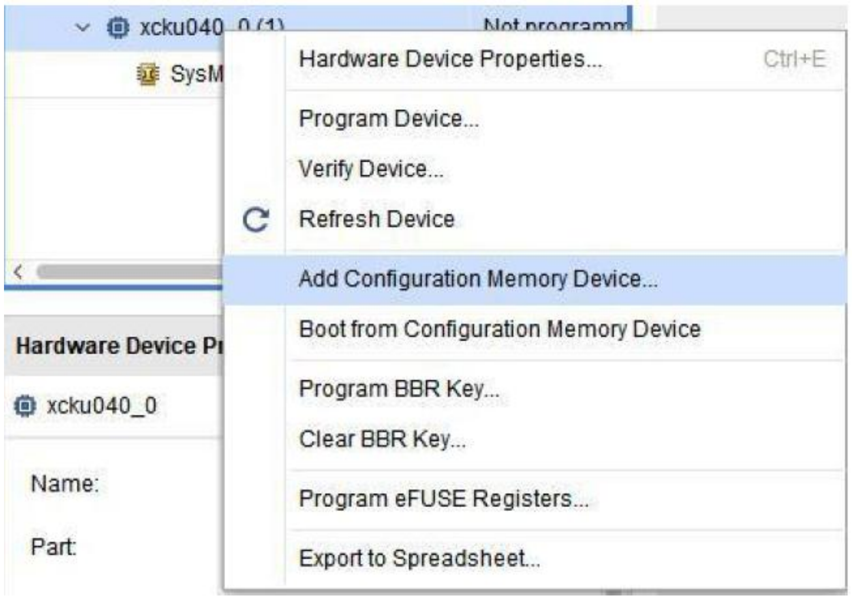
You can also try other patterns to light the LEDs, for example, let the lights run faster, or when several lights are on and off at the same time, etc., depending on your imagination, you can achieve more by writing your own program.

4.7 FLASH program curing

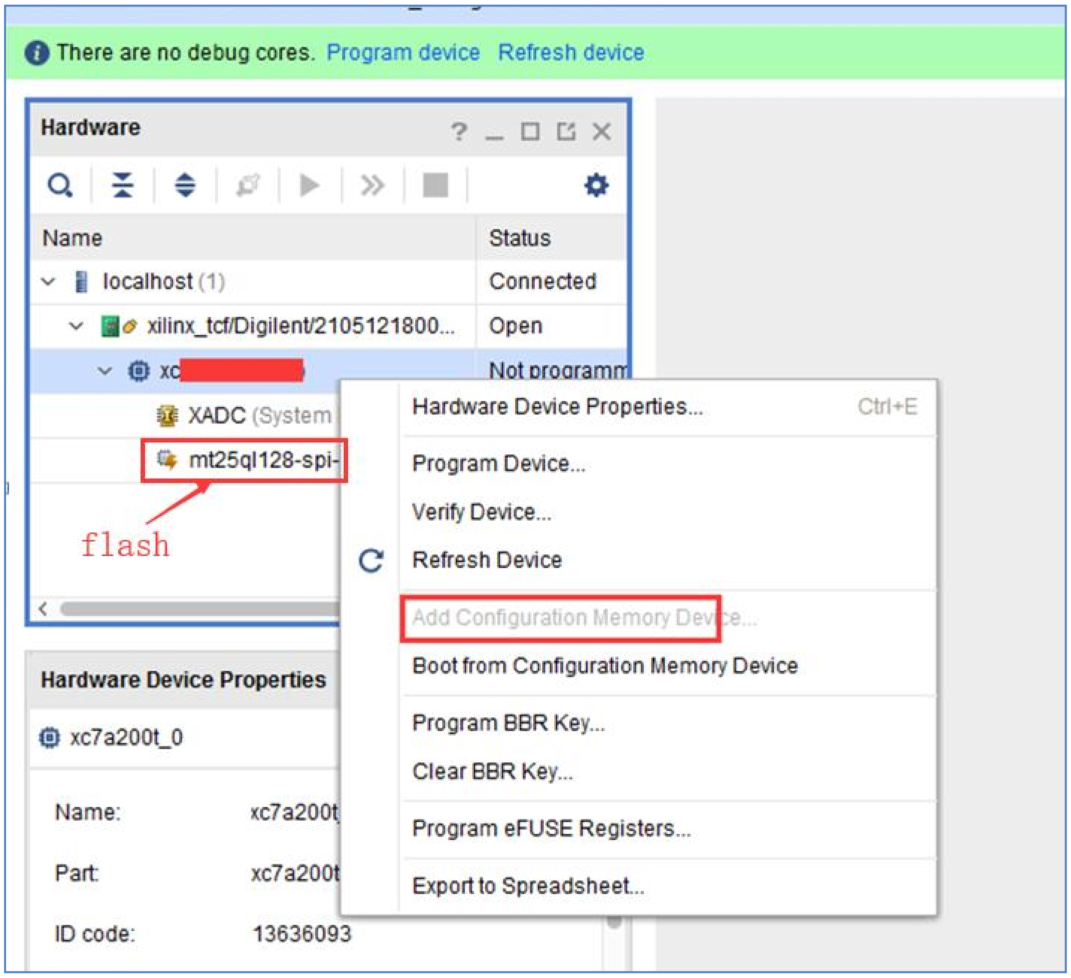
Some friends may have found that after downloading the Bit file to the FPGA, the configuration program has been lost after the development board is powered on, and JTAG download is required. This is no trouble! Well, in this section we will introduce how to solidify the configuration program into the FLASH on the development board, so that you don't have to worry about the program being lost after power off

On our development board there is an 8Pin 128Mbit FLASH for storing configuration programs. We can't directly download the “bit file” to this FLASH. We can only download the “Bin file” or the “MCS file”. In the previous we generated the bit file and also generated the Bin file. Let's take a look at the Bin file as an example to introduce the curing of the FLASH program.

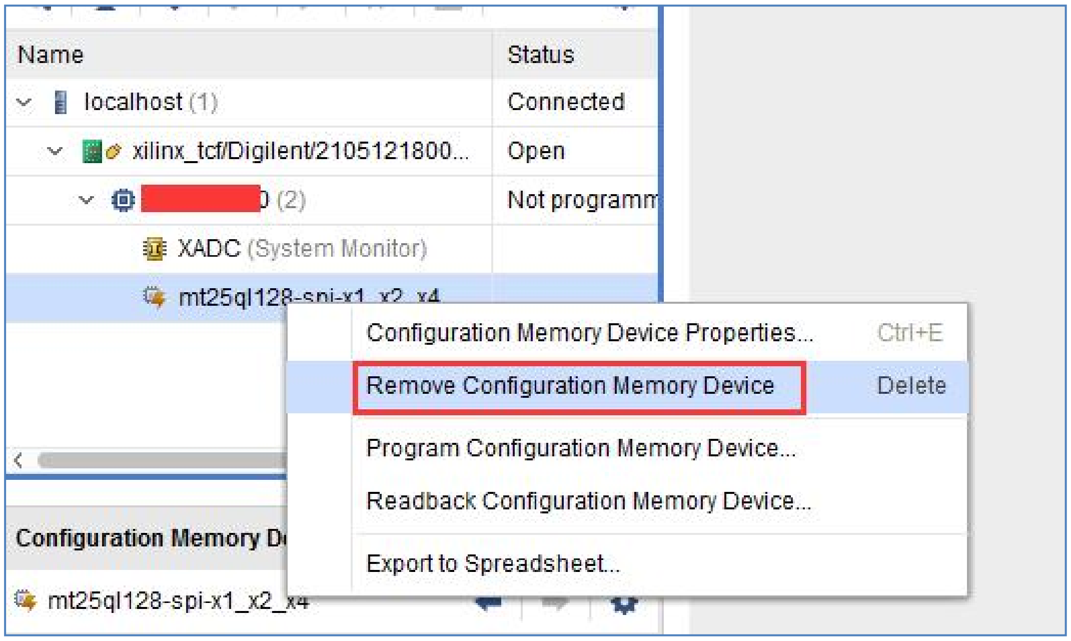
1. In the following figure, right click to select xc7a325t_0 chip, select “Add Configuration Memory Device... “ in the pop-up list



Note: If you find that this item is grayed out, it is because the FLASH configuration has been added to the project we provided, and FLASH cannot be added, as follows:



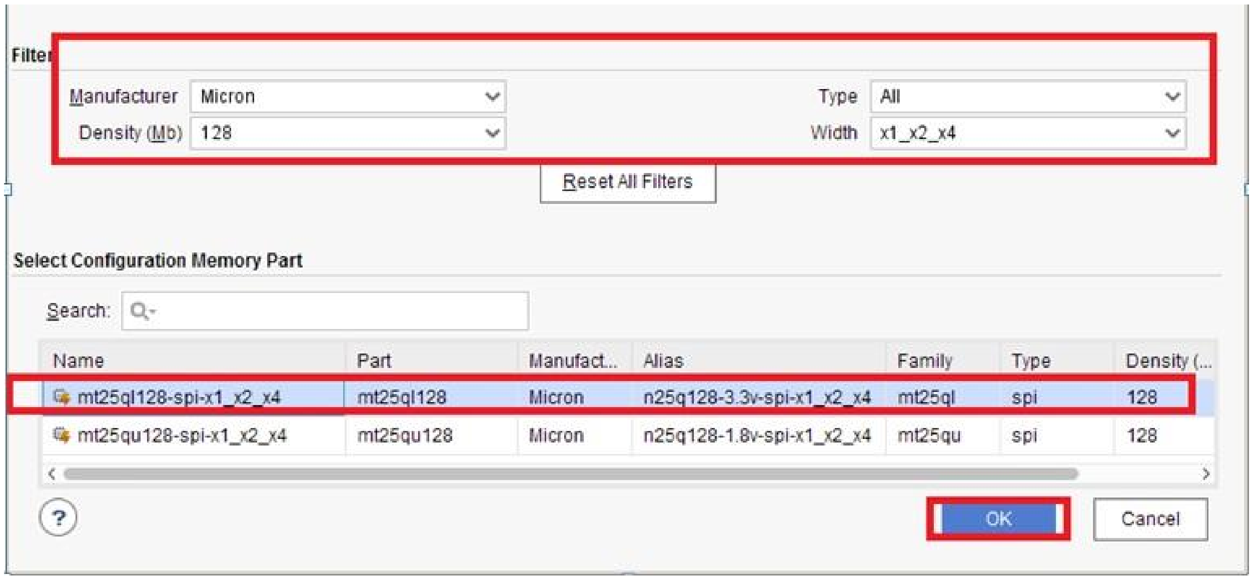
Of course, if you want to add experiments again in the existing flash project, you can remove the flash as shown below, and then follow the steps of adding flash above:



Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/2105121800...	Open
[redacted]0 (2)	Not programm
XADC (System Monitor)	
mt25ql128-spi-x1_x2_x4	

Configuration Memory Device Properties... Ctrl+E
Remove Configuration Memory Device Delete
Program Configuration Memory Device...
Readback Configuration Memory Device...
Export to Spreadsheet...
Configuration Memory D
mt25ql128-spi-x1_x2_x4

2. Select the correct FLASH model in the configuration interface of "Add configuration Memory Device", as shown below:



Filter

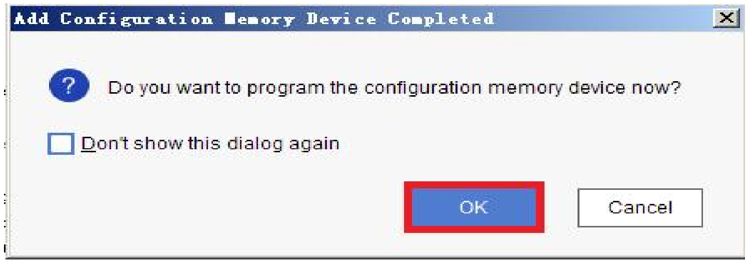
Manufacturer: Micron
Type: All
Density (Mb): 128
Width: x1_x2_x4

Select Configuration Memory Part

Search:

Name	Part	Manufact...	Alias	Family	Type	Density (...
mt25ql128-spi-x1_x2_x4	mt25ql128	Micron	n25q128-3.3v-spi-x1_x2_x4	mt25ql	spi	128
mt25qu128-spi-x1_x2_x4	mt25qu128	Micron	n25q128-1.8v-spi-x1_x2_x4	mt25qu	spi	128

3 . Prompt whether to program SPI FLASH, click OK.

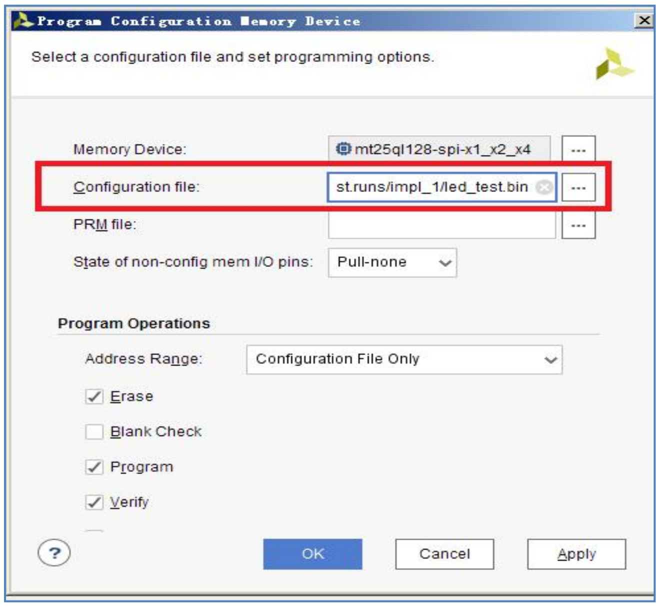


Add Configuration Memory Device Completed

Do you want to program the configuration memory device now?

Don't show this dialog again

In the pop-up “ Program Configuration Memory Device” window, the “ Configure file ” item selects the “led_test.bin” file generated by Vivado (this file defaults to the “imp1_1” directory). The “PRM File” item is not selected. In addition, in this window, users can also configure I/O for pull-up, pull-down or no pull-down. The configuration action options are left to the default.



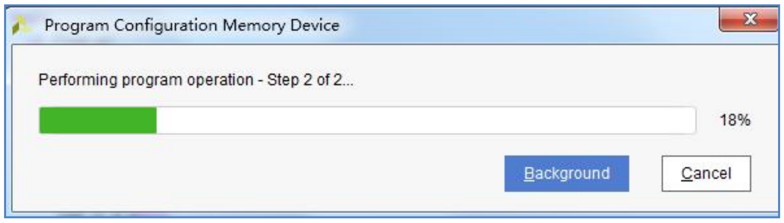
Program Configuration Memory Device

Select a configuration file and set programming options.

Memory Device: mt25ql128-spi-x1_x2_x4
Configuration file: st.runs/impl_1/led_test.bin
PRM file:
State of non-config mem I/O pins: Pull-none

Program Operations
Address Range: Configuration File Only
✓ Erase
Blank Check
✓ Program
✓ Verify

Click OK to start programming FLASH.

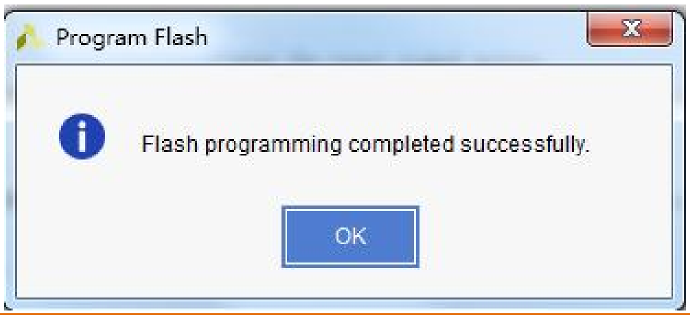


Program Configuration Memory Device

Performing program operation - Step 2 of 2...

18%

After FLASH is programmed, the following successful interface will pop up



Program Flash

Flash programming completed successfully.

At this point, the SPI FLASH is burned and the “led_test program” has been solidified into “SPI FLASH”. Let's verify, turn off the power and Power on the development board again. Wait a while and you can see that the LED lights on the development board are already run as water light

As you may have noticed, turning off the FPGA power and then powering it back on, it takes a long time for the LED lights on the development board to start run as water light. This is definitely not enough for some projects that are going to work soon after power-on. Is there any way to solve it? Of course, we can improve the read and write clock of SPI FLASH, and use “x4” to read and write. “QSPI” supports 4 data lines to read and write! The method is very simple, we just add the following three statements to the xdc file:

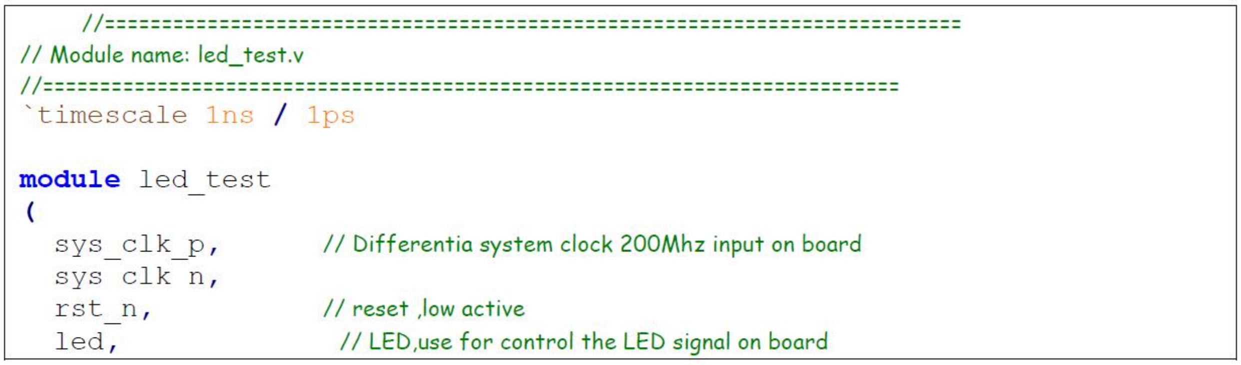
```
#############SPI Configurate Setting##################

set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [current_design]

set_property CONFIG_MODE SPIx4 [current_design]

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

The first two sets the data width and configuration mode of “QSPI FLASH ”, and the latter one is the configuration speed. The larger is the value, the faster is the speed. After modifying the xdc file, you need to recompile, then regenerate the bit and bin files, and then burn the “SPI FLASH” chip again according to the previous method. After the completion of the development of the board, this is not a power-up, LED lights began to exercise, right? After the completion, the development board is powered off and re-powered on. Is the development board powered up, and the LED lights start to exercise immediately?

So far, our first project has been successfully completed. I believe that you have mastered the entire process of Vivado's FPGA development, and it is no longer a layman of that FPGA! The use and mastery of some vivado software skills need to be familiar with everyone in the long-term practice and exploration.

5 Appendix

led_test.v (verilog Code)



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

module led_test
(
  sys_clk_p,          // Differentia system clock 200Mhz input on board
  sys_clk_n,
  rst_n,              // reset ,low active
  led,                  // LED,use for control the LED signal on board
  fan_pwm              //fan control
 );

//========================================================================
// PORT declarations
//========================================================================

input           sys_clk_p;
input           sys_clk_n;
input           rst_n;
output [3:0]    led;
output fan_pwm;
//define the time counter
reg [31:0]    timer;
reg [3:0]     led;
assign fan_pwm =1'b0;
//========================================================================
//Differentia system clock to single end clock
//========================================================================
wire          sys_clk;
 IBUFGDS #
   (
    .DIFF_TERM    ("FALSE"),
    .IBUF_LOW_PWR ("FALSE")
    )
   u_ibufg_sys_clk
     (
      .I  (sys_clk_p),
      .IB (sys_clk_n),
      .O  (sys_clk  )
      );
//========================================================================
// 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 <= 4'b0000;                    //when the reset signal active
      else if (timer == 32'd49_999_999)      //time counter count to 0.25 sec,LED1
lighten

          led <= 4'b0001;
      else if (timer == 32'd99_999_999)      //time counter count to 0.5 sec,LED2 lighten
      begin
```

```
            led <= 4'b0010;
          end
        else if (timer == 32'd149_999_999)     //time counter count to 0.75 sec,LED3
lighten
            led <= 4'b0100;
        else if (timer == 32'd199_999_999)     //time counter count to 1 sec,LED4 lighten
            led <= 4'b1000;
    end

    endmodule
```

Note: When defining a register, if the register is used in the "always" block, it must be defined as the "reg" type. If it is only used for wiring or the direct assignment needs to be defined as the "wire" type, the input signal type cannot be defined as "reg" type, whether it is "reg" type signal or a "wire" type signal. The defined register width must meet the needs of use, but must be slightly larger or equal to the bit width to be used. If the defined register bit width is much larger than the usage requirement, it will waste resources. If the defined bit width is smaller than the usage requirement, the data bit will be truncated, resulting in a program error. There are other types of signals and usage, please refer to the Verilog syntax tutorial.