APRIL 2023, 4 BY ACKYE

# Smart ZYNQ (SP&SL VERSION) Project X Vivado's built-in simulation feature demo

When using FPGA to do the project, most of the time is spent on debugging, debugging in addition to the board actually running to see the results and using ILA for debugging, you can also use software for simulation, this article will briefly demonstrate the simulation function that comes with XILINX.

**This article is demonstrated on vivado2018.3, please research for other versions**

**(Note: The content of this section applies to the boards of Smart ZYNQ SP and SL Edition, if it is Smart ZYNQ Standard Edition, please refer to the corresponding board directory)**

Simulation is very important in our design, when we write more complex logic, with vivado wiring synthesis tens of minutes or even hours, sometimes it is not good to synthesize, generate bit to download in, and find that our logic has problems, at this time to modify, and then download may take a lot of time, and even not necessarily find the problem point, this time the importance of simulation is highlighted.

The simulation function is different from the ILA, the ILA function can be understood as the logic analyzer inside the FPGA, which is equivalent to collecting the signal waveform inside the FPGA and uploading it to the software for display, and the simulation function is equivalent to simulating the results and the signal waveform in the middle according to the logic we wrote, through the simulation we can judge whether the work of each module is expected in the process of program design, whether there are unexpected problems, etc.

Another advantage of simulation is that simulation can start observing waveforms from 0 moment, and can add a large number of observation signals, which cannot be compared to

downloading to the board to observe the results, or capturing waveforms with ILA.

Novices will be more resistant to simulation, but if you are proficient in simulation skills, you can do more with less.

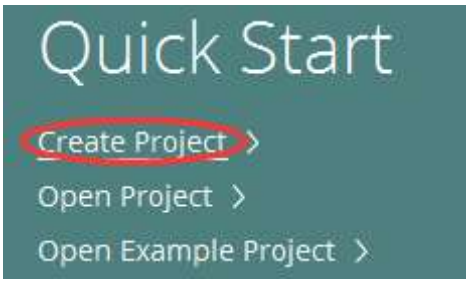
## Simulation in action

This article uses a simple project of counters to demonstrate the use of ILA function modules.

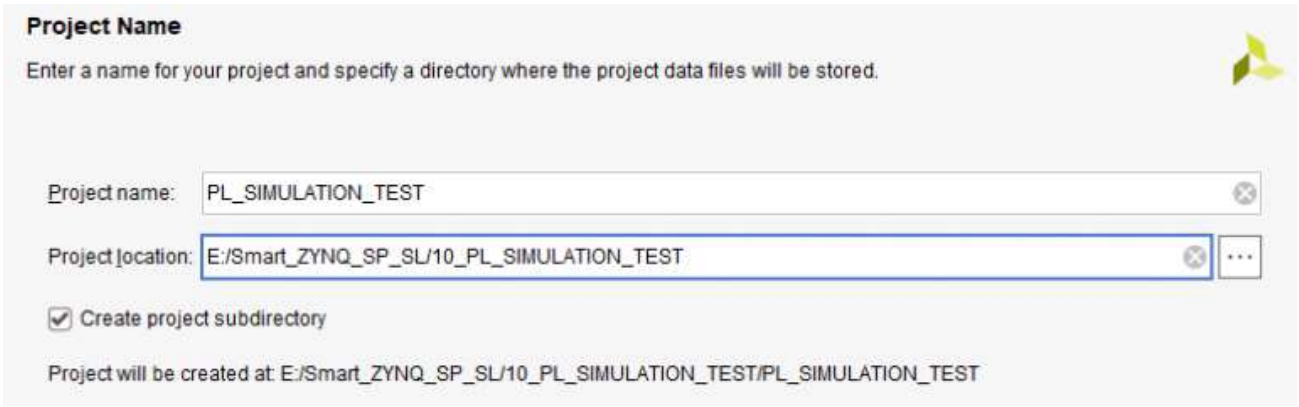
This article is demonstrated on vivado2018.3, please research for other versions

# 1. Specific steps

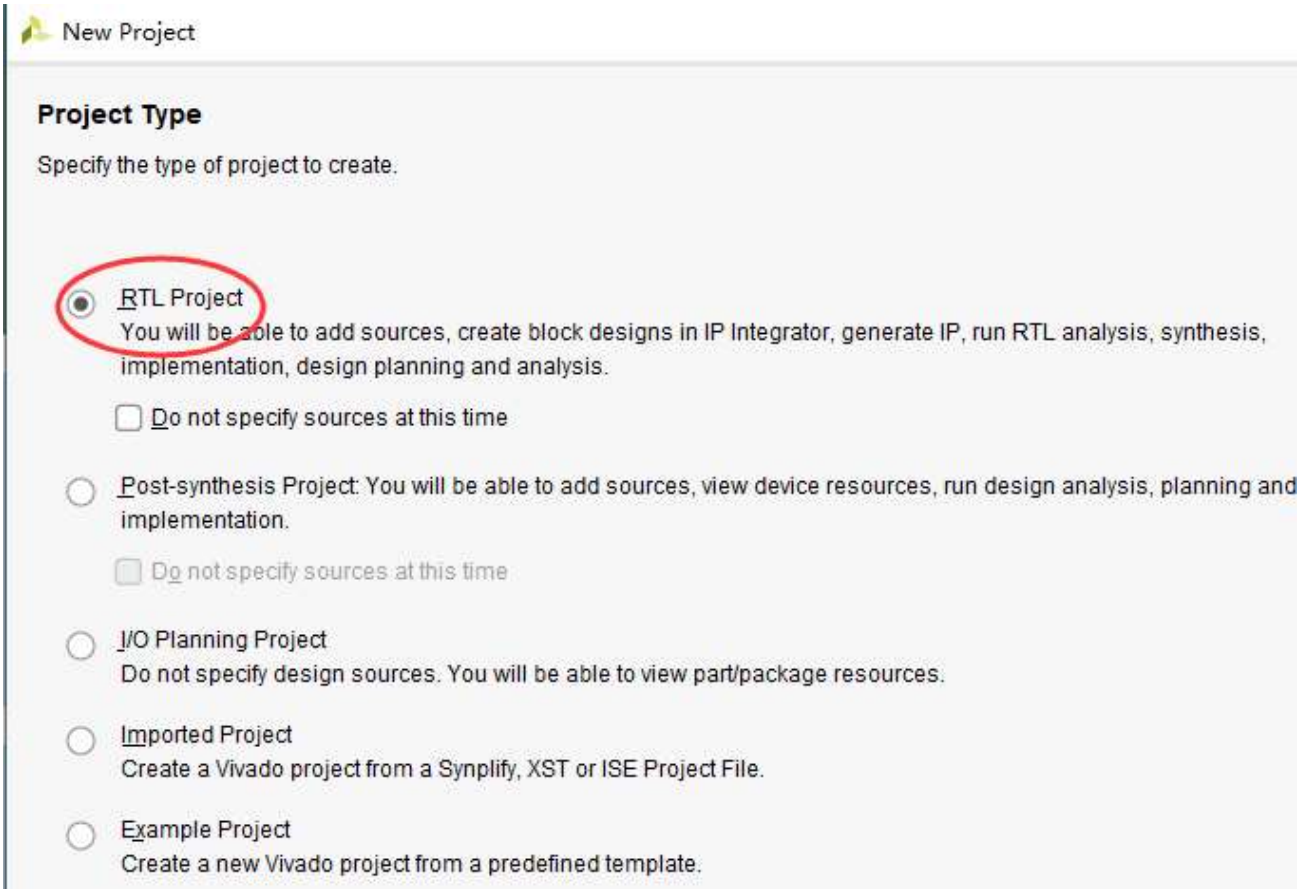
1) Specific steps to create a VIVADO project, open the software and select Create Project, as shown in the following figure




2) Click NEXT and enter the project name in the second dialog box "Project name" that appears; Select the save path in Project location; Check "Create project subdirectory" and click "Next" **Note, all paths cannot appear Chinese name**



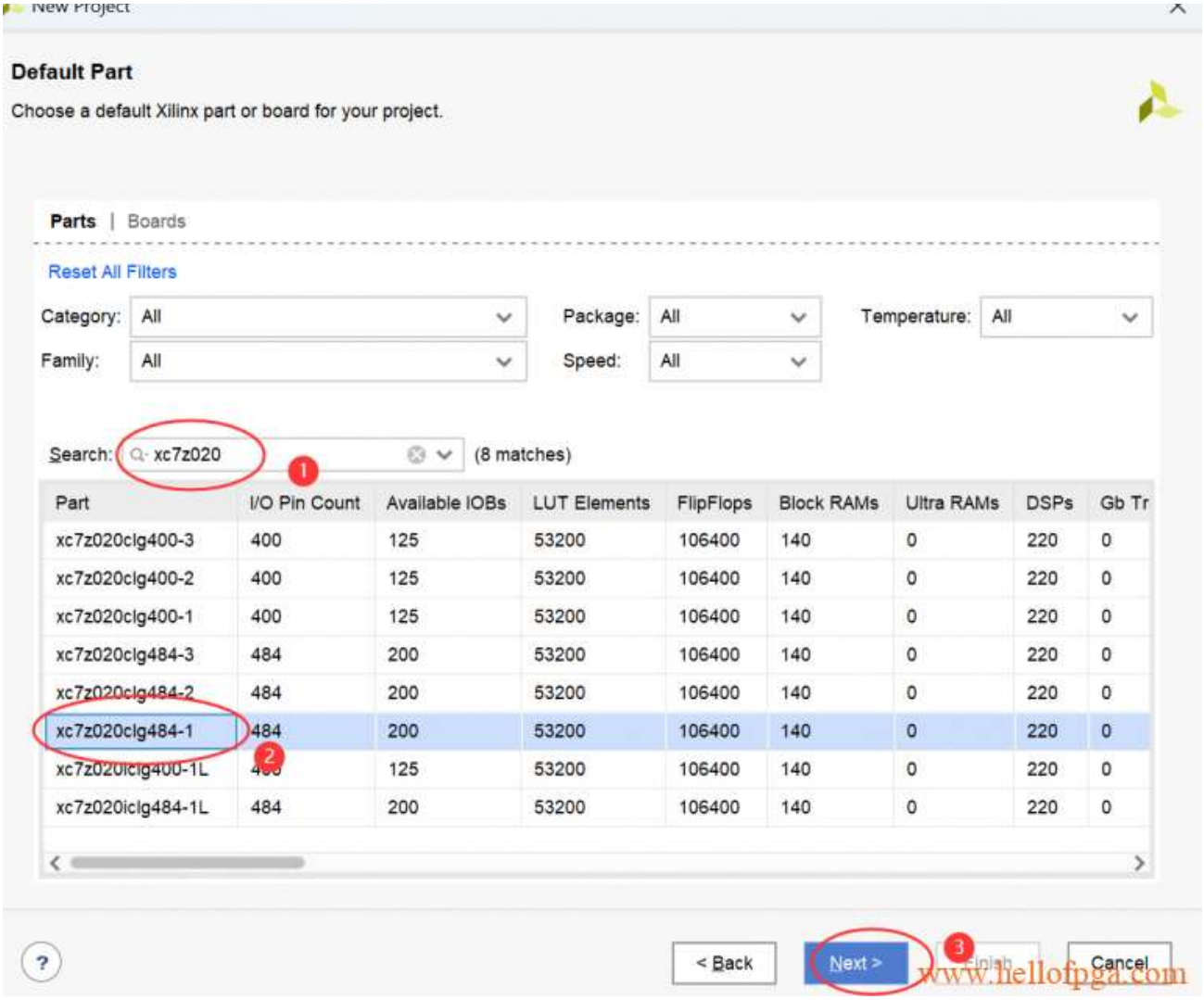
3) Click the RTL PROJECT option and click NEXT



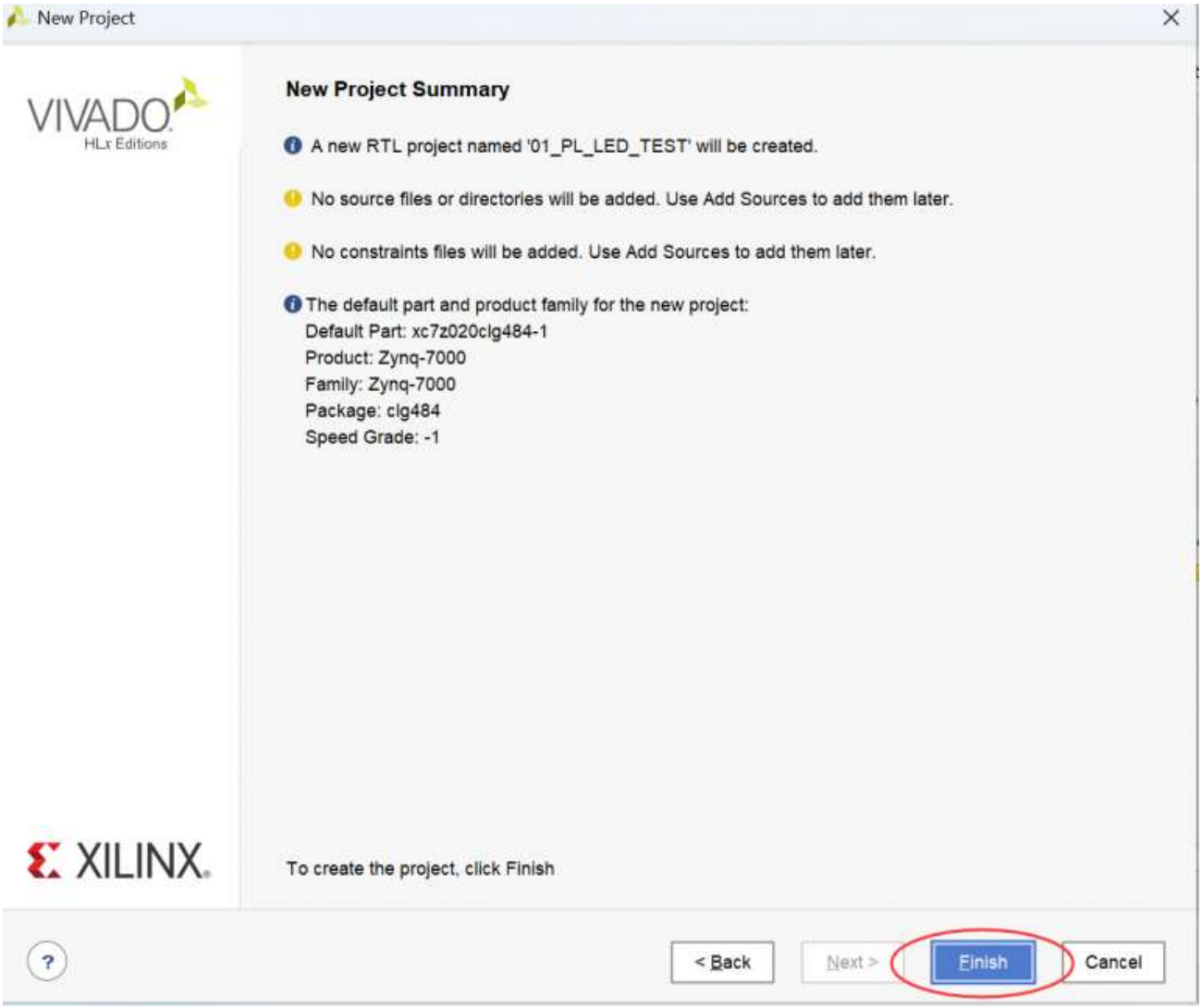

4） 第四步Add Sources 选项直接留空，NEXT

5）第五步Add Constraints 选项直接留空，NEXT

6）选择芯片型号 XC7Z020CLG484-1

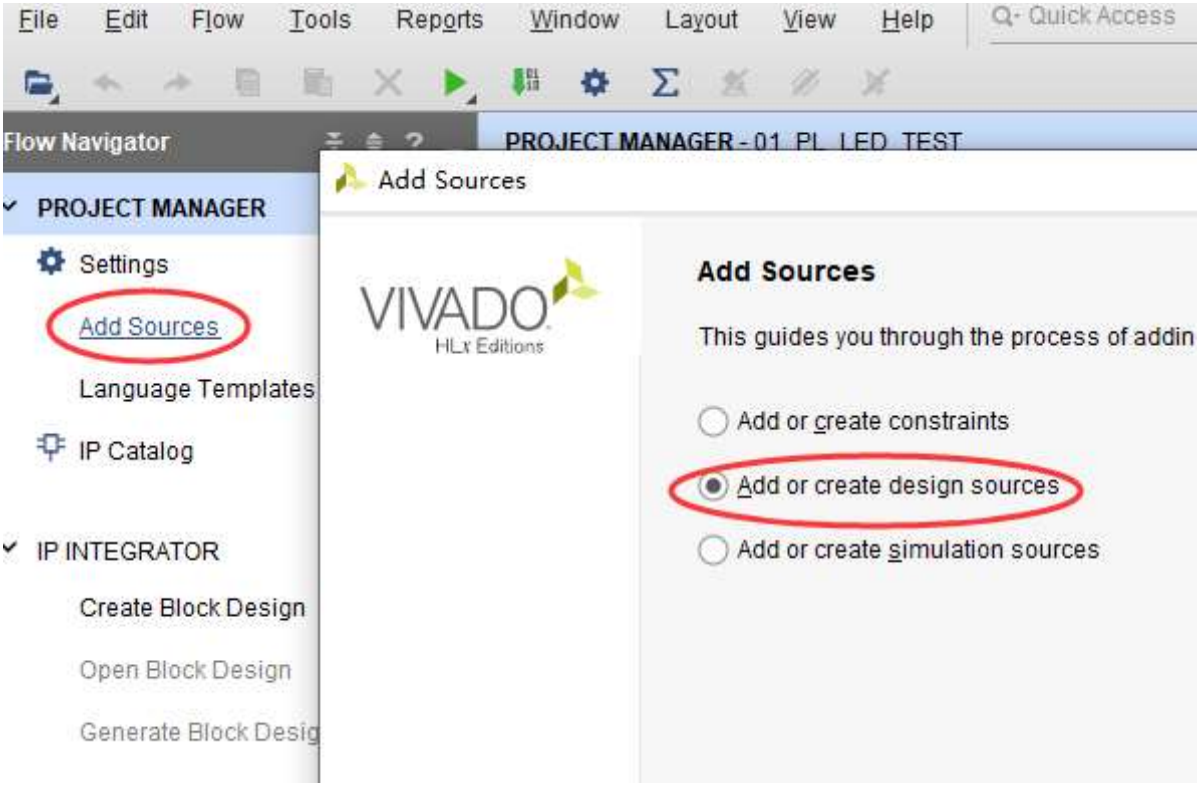



7）确认所选信息 点击“Finish”，完成vivado的工程创建

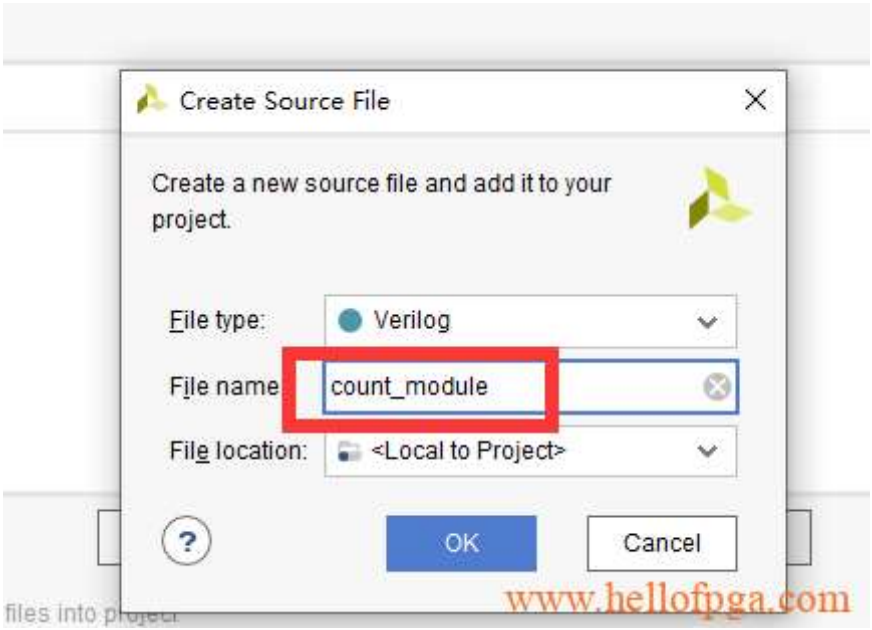



# 2 增加我们的verilog逻辑

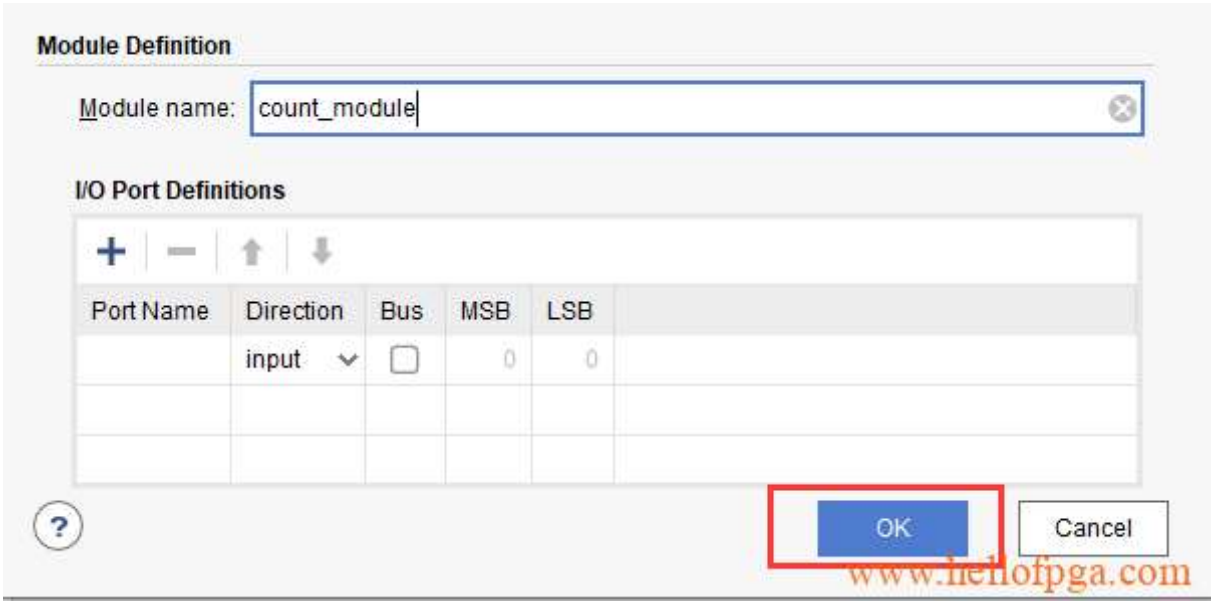
1）在主界面点击左侧 Add Sources ,点击 复选框的Add or create design sources 选项 并点击NEXT



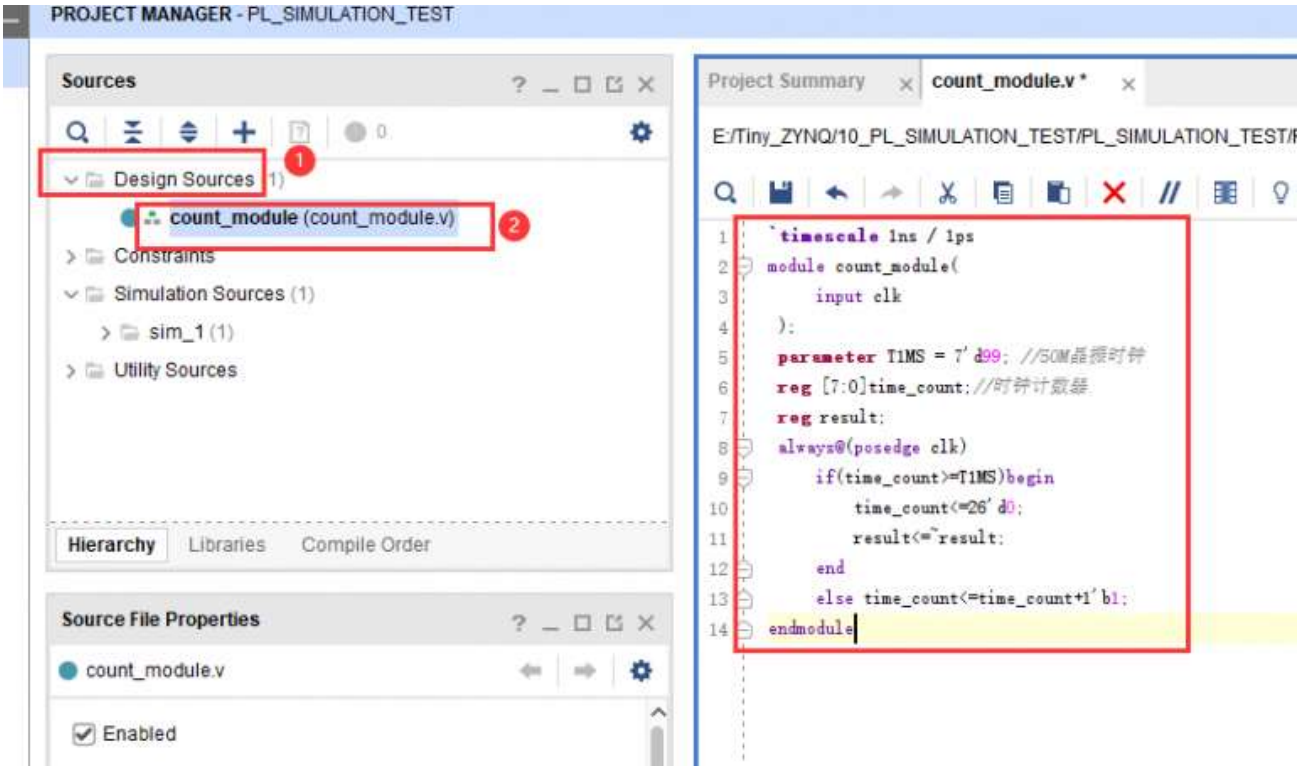

2）在出现的Add Sources 中 选择创建新文件 Create FILE 如下图所示，并在弹出的窗口中选择类别为Verilog ，在FILE name中填写文件的名称，这里用”count_module” 代替，点击OK 并点击FINISH




3）在跳出的窗口中可以填写模块的输入输出信号，由于这部分工作在代码中可以完成，所以这里直接点OK 完成VERILOG 文件的创建




1.2 Code: Here we do a simple function on the code, the FPGA chip internal counter time_count every 100 times the count (0-99) result register flips once (from 1 to 0, or from 0 to 1), **the program and engineering one is very similar, but for demonstration, so the counter time is shortened**

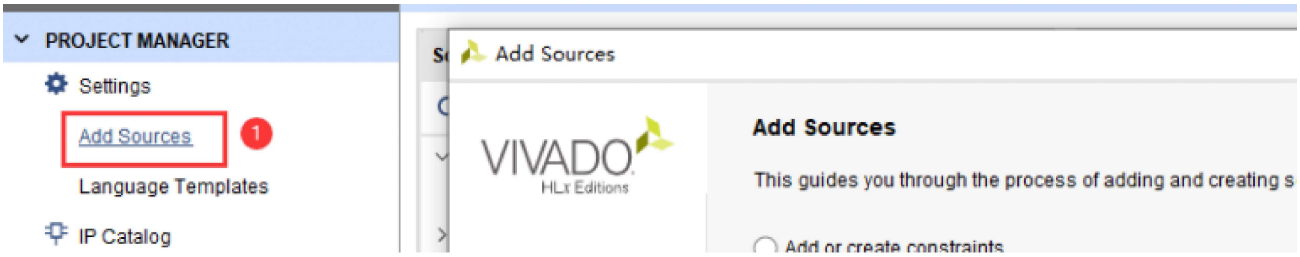



```
`timescale 1ns / 1ps
module count_module(
     input clk,
     input rst_n
 );
 parameter T1MS = 7'd99; //50M晶振时钟
```

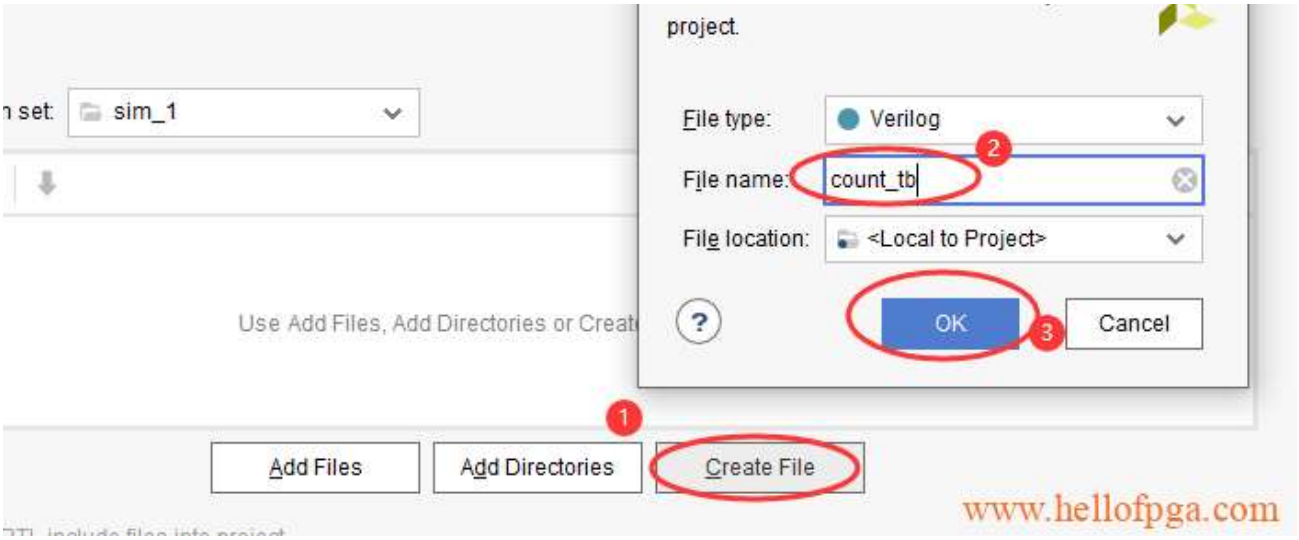
```
    reg [7:0]time_count;//时钟计数器
    reg result;
    always@(posedge clk or negedge rst_n)
        if(!rst_n)begin
           time_count<=26'd0;
           result<=1'b0;
        end
        else if(time_count>=T1MS)begin
            time_count<=26'd0;
            result<=~result;
        end
        else time_count<=time_count+1'b1;
endmodule
```

# 3. Create the simulation file *testbench*

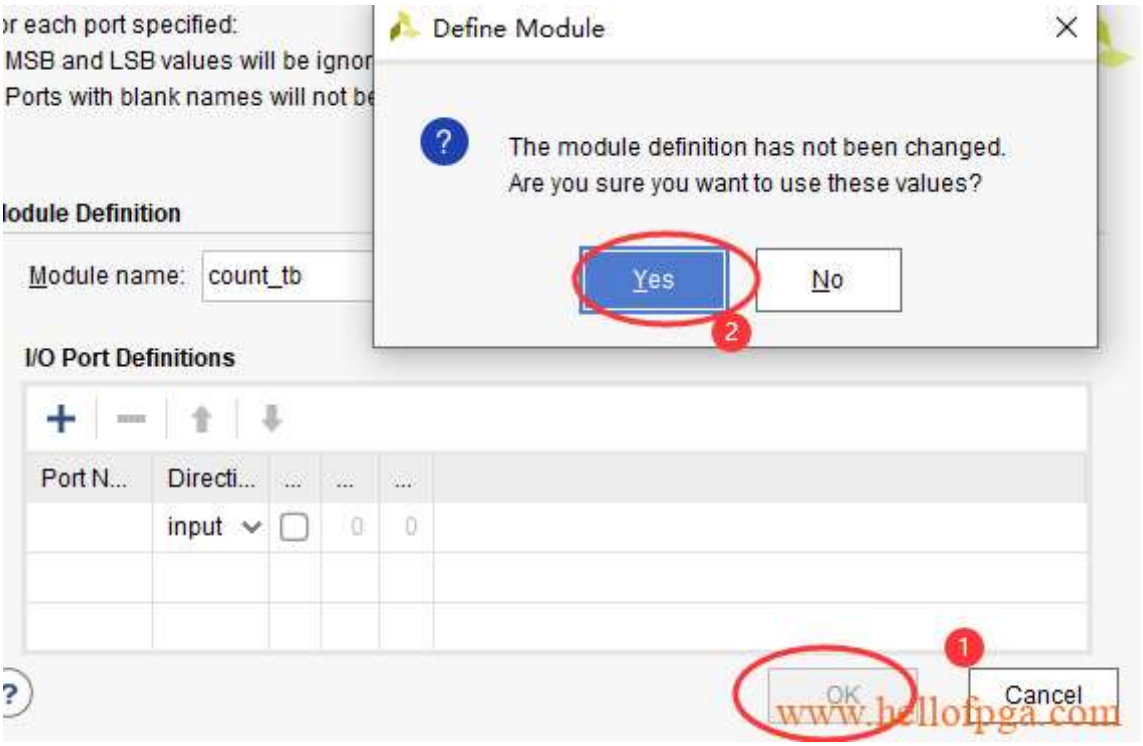
1) Go back to the main interface and click Add Sources on the left, click the Add or createsimulation sources option in the checkbox and click NEXT




2) Click Create File to enter the name of testbench in the pop-up dialog box (here replace count_tb, where tb stands for testbench abbreviation), and then click OK




3) Select OK in the next dialog box and select YES in the pop-up yellow box




4) At this time, you can find the count_tb.v file we just created in the simulation directory of the project


Double-click to open count_tb.v and add the incentive code for testbench inside

```
`timescale 1ns / 1ps
module count_tb(

    );

reg    clk_i;
reg    rst_n_i;

initial begin
      clk_i = 0;
      rst_n_i = 0;
      #200;
      rst_n_i = 'b1;
  end

always #5 clk_i = ~clk_i;

count_module u_test(
    .clk(clk_i),
    .rst_n(rst_n_i)
);



endmodule
```

**Note: The writing of testbench is a bit different from standard logic, and most testbench code is not synthetic**

**testbench** 内容注释：

initial 模块 是从0时刻开始就工作的， 只执行一次， #200相当于延迟200个单位周期（备注 这句是不可综合语句）

#200； 后再跟 rst_n_i=1'b1; 代表 200个单位时间后 rst由0置位到1

always #5 clk_i = ~clk_i; 代表每5个单位时间 翻转一次，也就是10个单位时间为一个周期

至于仿真的时间单位 则在程序的最顶端 描述 `timescale 1ns / 1ps //仿真时间单位/仿真时间精度

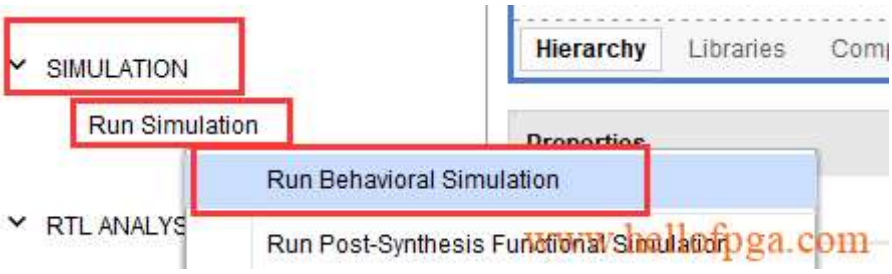
count_module u_test（）相当于是对count_module的程序实例化并且命名为u_test

5) After the simulation code is written, save the file

**You can also press the green arrow to precompile and see if the code reports an error**

# Conduct simulations

1) Click Run Behavioral simulation to start the simulation

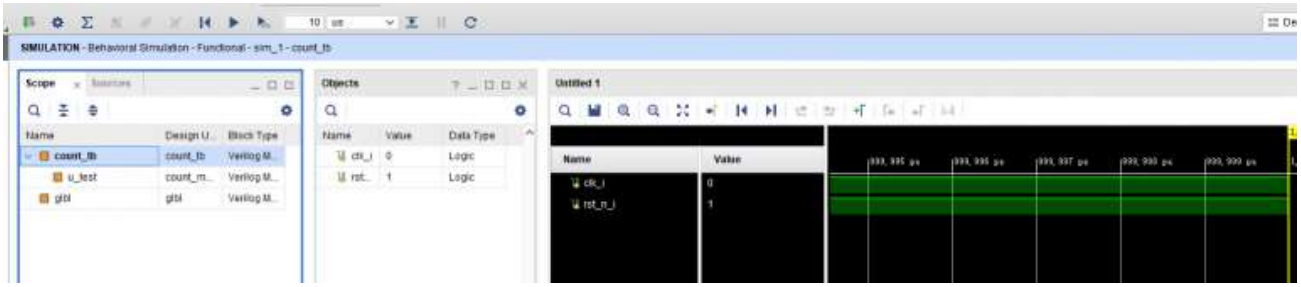



Note: The other options represent Run Behavior Simulation Functional Simulation, Run Post-Systhesis Functional Simulation Synthesis, Run Post-Systhesis Timing Simulation Synthesis, and Run Post-Implementation Functional Simulation Functional simulation after

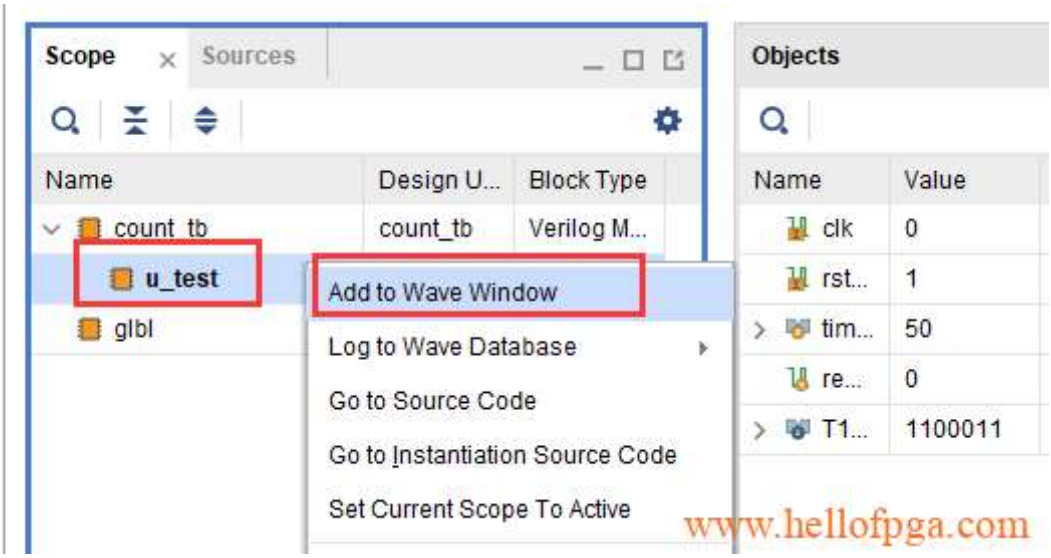
implementation/place-and-route, Run Post-Implementation Timing Simulation Timing simulation after implementation/post-placement Simulation Generally, we can choose functional simulation.

2) Then comes our imitation Allah interface

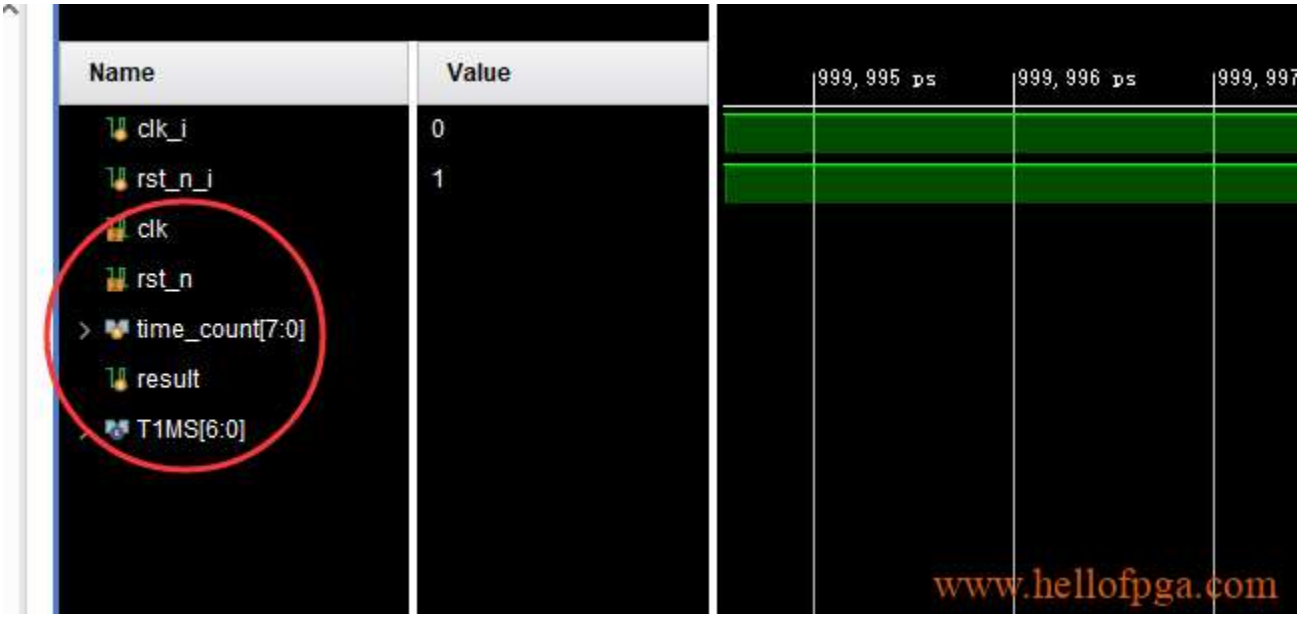
When we select any module in the Scope window, the signal registers of the corresponding module will appear in the Objects window



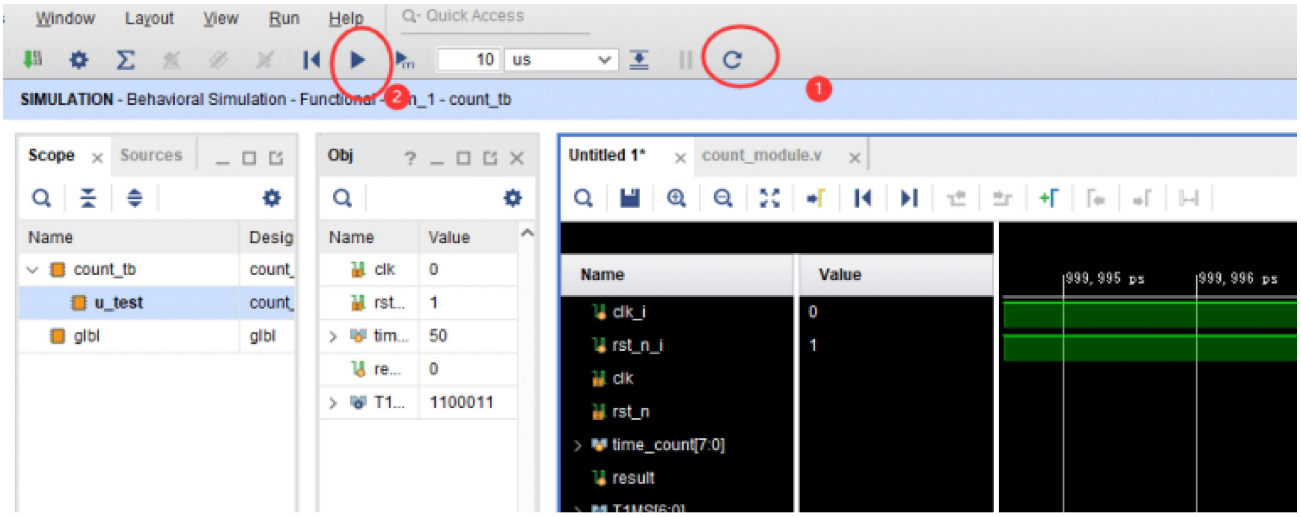
3) Here we right-click on the u_test module we instantiated, and select Add to Wave Window to add all signals under the u_test to the observation window



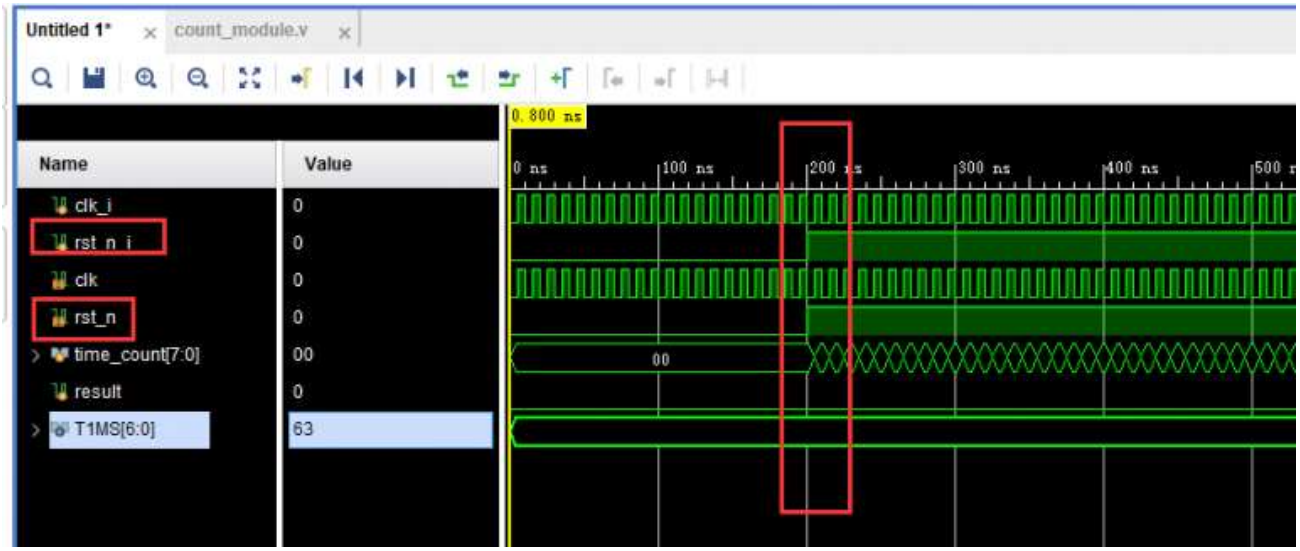

In this way, the observation window will appear with the signals and registers we need




4) Because the above signal is newly added, you need to rerun the simulation here, and then click Run all (if you do not rerun, the newly added signal line will not have a waveform)

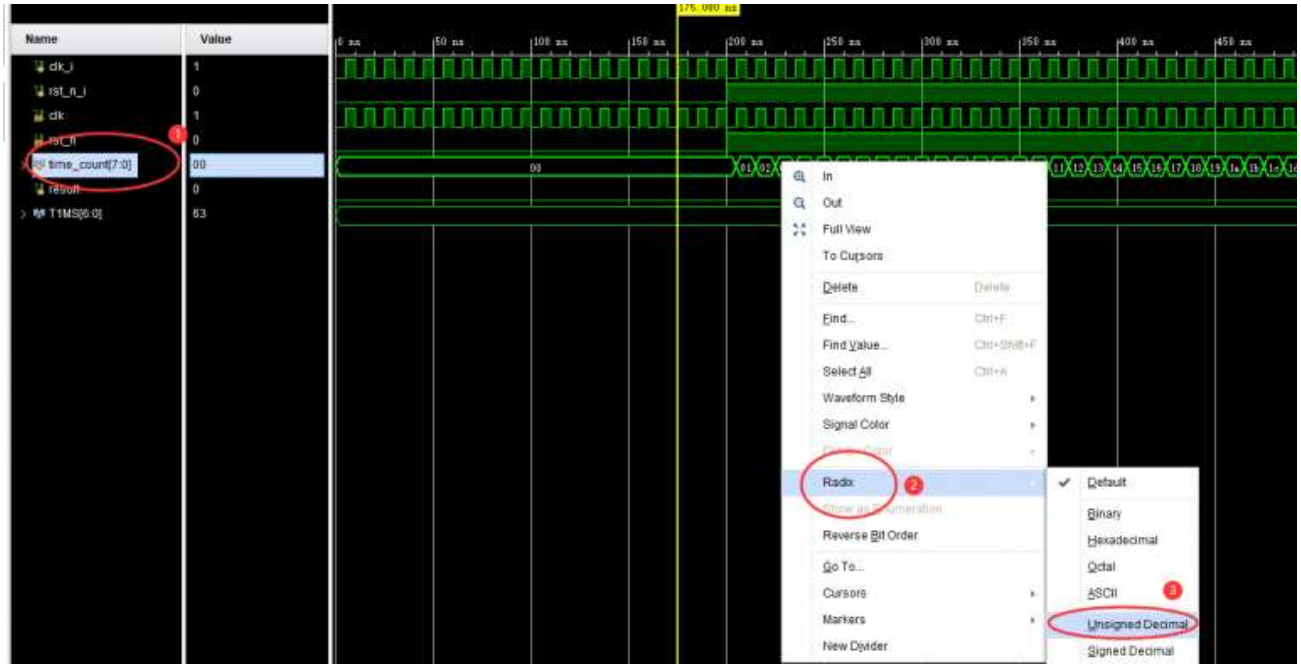



Zoom out with the mouse and pull the waveform to the left to see it at the very beginning

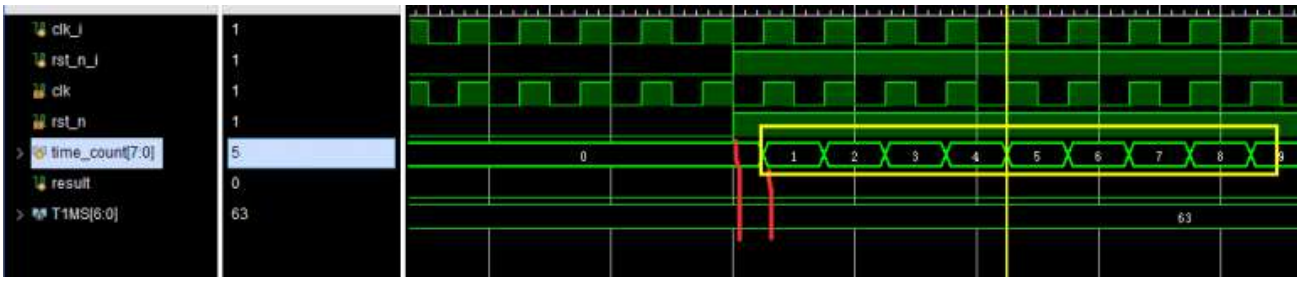



Our simulation is already working normally, and we can see that rst_n starts to go from low to high at 200ns, the same as we set, and clk_i keeps working at 10ns

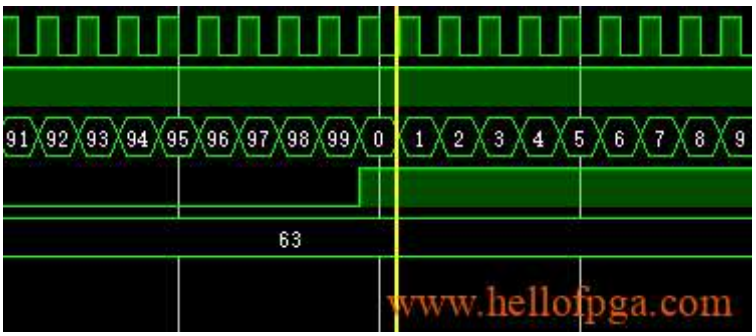

Expand the waveform Right click time_count counter in Radix change the format of our data to unsigned Decimal, unsigned decimal



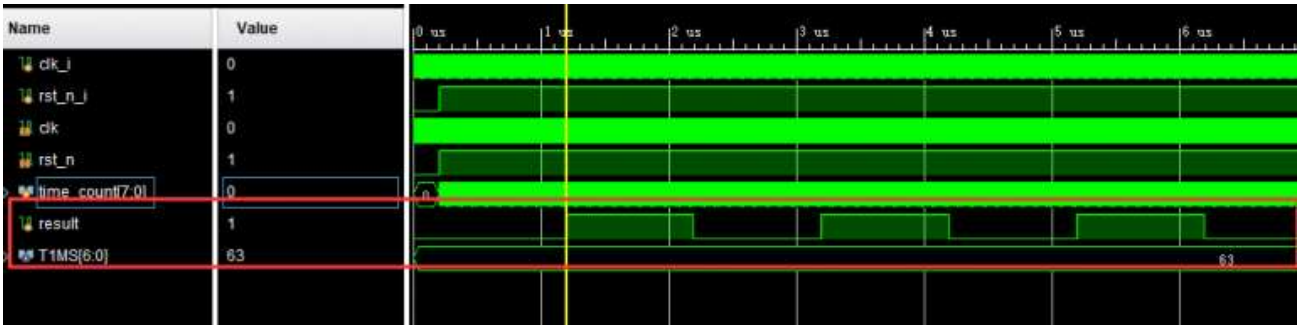

It can be observed that our counter starts from the second rising edge after the reset signal is pulled high, and the counter self-increments from 0-1 and increases itself with each rising edge



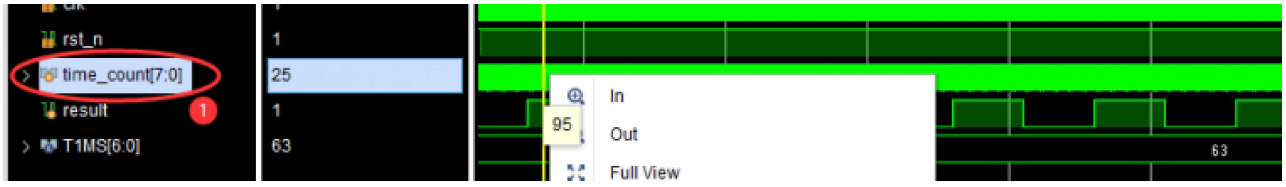

When the counter increases to 99, the counter changes from 99 to 0 on the next rising edge, and the result signal flips once



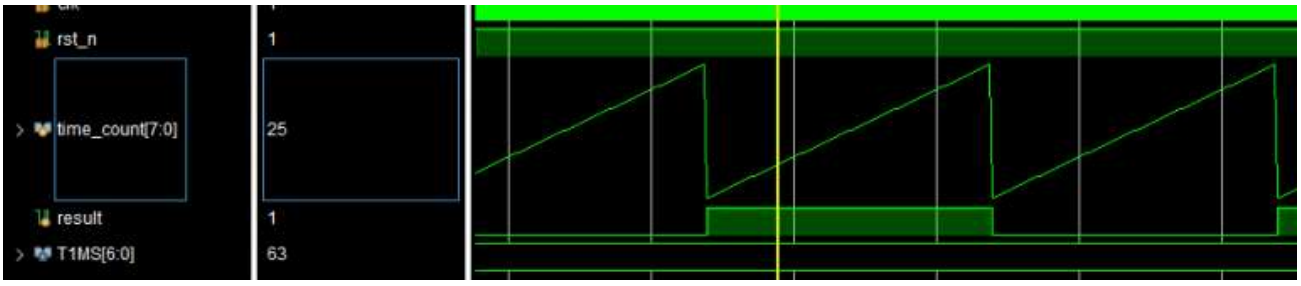

Zooming out of the waveform, you can see that the result register is oscillating back and forth between 1-100 at intervals of 10us (0 99ns because the counter counts 0-1) with a period of 2us




Of course, we can also right-click our time_count counter and click Waveform Style –> Analog to display the analog waveform corresponding to the value (such as sine wave, triangle wave, etc.)




Here, because count is from 0-99 to 0, the shape looks a bit like a triangle wave, as shown in the figure below




Here is the complete project:

10_PL_SIMULATION_TEST_XC7Z020 **Download**

**The above is a simple call and introduction of vivado's own simulation function, of course, the function of vivado simulation is far more than what I introduced, you**

## need to dig it yourself, do not expand

SMART ZYNQ SP & SL