VIVADO 下的仿真入门

软件版本:VIVADO2017.4
操作系统:WIN10
硬件平台: ZYNQ-7 系列开发板
米联客(MSXBO)论坛 www.osrc.cn 答疑解惑专栏开通，欢迎大家给我提供!!!

1.1 FPGA 设计仿真验证简介

严格来讲，FPGA 设计验证包括功能仿真、时序仿真和电路验证，它们分别对应整个开发流程的每一个步骤。仿真是指使用设计软件包对已实现的设计进行完整的测试，并模拟实际物理环境下的工作情况。
功能仿真是指仅对逻辑功能进行模拟测试，以了解其实现的功能是否满足原设计的要求，仿真过程没有加入时序信息，不涉及具体器件的硬件特性，如延时特性等，因此又叫前仿真，它是对 HDL 硬件描述语言的功能实现情况进行仿真，以确保 HDL 语言描述能够满足设计者的最初意图。时序仿真则是在 HDL 可以满足设计者功能要求的基础上，在布局布线后，提取有关的器件延迟、连线延时等时序参数信息，并在此基础上进行的仿真，也成为后仿真，它是接近于器件真实运行状态的一种仿真。

1.2 仿真文件

Vivado 仿真需要两个文件，一个是我们的源文件.V 文件，另外一个就是 testbench 文件。我们最后仿真的时候，其实仿真的是 testbench 文件。本节课程就以 CH06 vivado 仿真实验之多路分配器实验为例

1.2.1 程序源码

```
`timescale 1ns / 1ps
//----------------------------------------------------------------------------------
// Target Devices:
// Tool versions:    VIVADO2017.4
// Description:        Divider_Multiple
// Revision:
// Additional  Comments:
//1) _i PIN input
//2) _o PIN output
//3) _n PIN active low
//4) _dg debug signal
//5) _r    reg delay
//6) _s state machine
*/
```

```
//----------------------------------------------------------------------------
module Divider_Multiple(
    input clk_i,
    input rst_n_i,
    output div2_o,
    output div3_o,
    output div4_o,
    output div8_o,
    output div2hz_o
    );

reg div2_o_r;
always@(posedge clk_i or negedge rst_n_i)
begin
    if(!rst_n_i)
        div2_o_r<=1'b0;
    else
        div2_o_r<=~div2_o_r;
end

reg [1:0] div_cnt1;
always@(posedge clk_i or negedge rst_n_i)
begin
    if(!rst_n_i)
        div_cnt1<=2'b00;
    else
        div_cnt1<=div_cnt1+1'b1;
end

reg div4_o_r;
reg div8_o_r;
```

```
always@(posedge clk_i or negedge rst_n_i)
begin
    if(!rst_n_i)
        div4_o_r<=1'b0;
    else if(div_cnt1==2'b00 || div_cnt1==2'b10)
        div4_o_r<=~div4_o_r;
    else
        div4_o_r<=div4_o_r;
end

always@(posedge clk_i or negedge rst_n_i)
begin
    if(!rst_n_i)
        div8_o_r<=1'b0;
    else if((~div_cnt1[0]) && (~div_cnt1[1]))
        div8_o_r<=~div8_o_r;
    else
        div8_o_r<=div8_o_r;
end

reg [1:0] pos_cnt;
reg [1:0] neg_cnt;
always@(posedge div2_o_r or negedge rst_n_i)
begin
    if(!rst_n_i)
        pos_cnt<=2'b00;
    else if(pos_cnt==2'd2)
        pos_cnt<=2'b00;
    else
        pos_cnt<=pos_cnt+1'b1;
end
```

```
always@(negedge div2_o_r or negedge rst_n_i)
begin
    if(!rst_n_i)
        neg_cnt<=2'b00;
    else if(neg_cnt==2'd2)
        neg_cnt<=2'b00;
    else
        neg_cnt<=neg_cnt+1'b1;
end

reg div3_o_r0;
reg div3_o_r1;
always@(posedge div2_o_r or negedge rst_n_i)
begin
    if(!rst_n_i)
        div3_o_r0<=1'b0;
    else if(pos_cnt<2'd1)
        div3_o_r0<=1'b1;
    else
        div3_o_r0<=1'b0;
end

always@(negedge div2_o_r or negedge rst_n_i)
begin
    if(!rst_n_i)
        div3_o_r1<=1'b0;
    else if(neg_cnt<2'd1)
        div3_o_r1<=1'b1;
    else
        div3_o_r1<=1'b0;
end
reg div2hz_o_r;
```

```
reg [25:0] div2hz_cnt;
always@(posedge clk_i or negedge rst_n_i)
begin
    if(!rst_n_i)
        div2hz_cnt<=0;
    else if(div2hz_cnt<26'd25_000000)
        div2hz_cnt<=div2hz_cnt+1'b1;
    else
        div2hz_cnt<=0;
end

always@(posedge clk_i or negedge rst_n_i)
begin
    if(!rst_n_i)
        div2hz_o_r<=0;
    else if(div2hz_cnt==26'd12_999999 || div2hz_cnt==26'd24_999999)
        div2hz_o_r<=~div2hz_o_r;
    else
        div2hz_o_r<=div2hz_o_r;
end
assign div2_o=div2_o_r;
assign div3_o=div3_o_r0 | div3_o_r1;
assign div4_o=div4_o_r;
assign div8_o=div8_o_r;
assign div2hz_o=div2hz_o_r;

ila_0 ila_0_0 (
    .clk(clk_i), // input wire clk
    .probe0(div2hz_o), // input wire [0:0]    probe0
    .probe1({div2_o,div3_o,div4_o,div8_o}) // input wire [3:0]    probe1
);
endmodule
```

1. 2. 2 Testbench 文件

```
module Divider_Multiple_top_TB;
    // Inputs
    reg clk_i;
    reg rst_n_i;

    // Outputs
    wire div2_o;
    wire div3_o;
    wire div4_o;
    wire div8_o;
    wire div2hz_o;

    // Instantiate the Unit Under Test (UUT)
    Divider_Multiple_top  uut (
        .clk_i(clk_i),
        .rst_n_i(rst_n_i),
        .div2_o(div2_o),
        .div3_o(div3_o),
        .div4_o(div4_o),
        .div8_o(div8_o),
        .div2hz_o(div2hz_o)
    );

    initial
        begin
        // Initialize  Inputs4
        clk_i = 0;
        rst_n_i = 0;

        // Wait 100 ns for global reset to finish
        #96;
        rst_n_i=1;
    end
        always
            begin
                #5 clk_i=~clk_i;
```

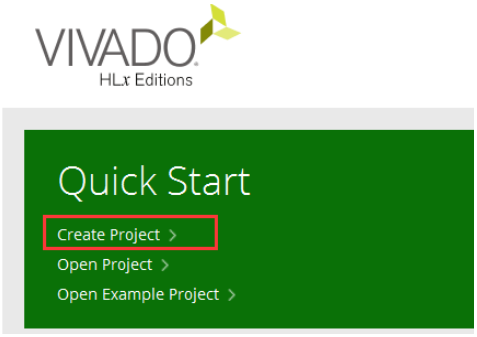
```
                end


endmodule
```

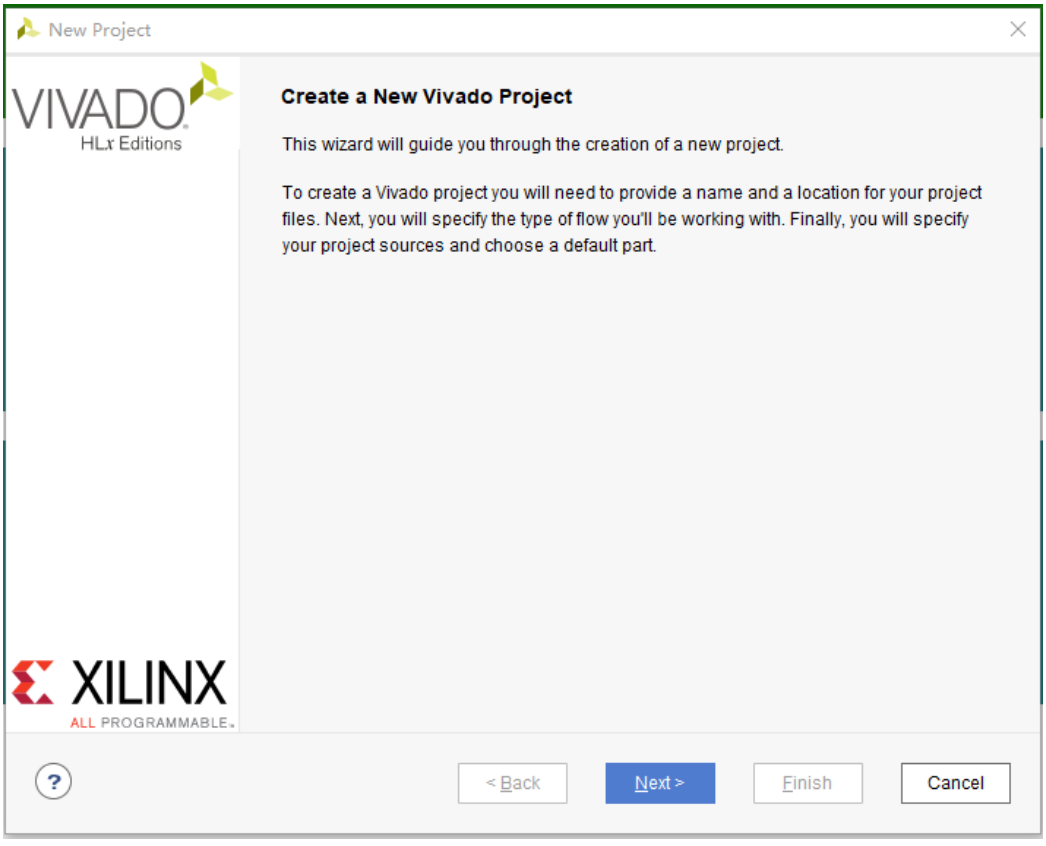
1.3 行为仿真

1.3.1 创建仿真工程

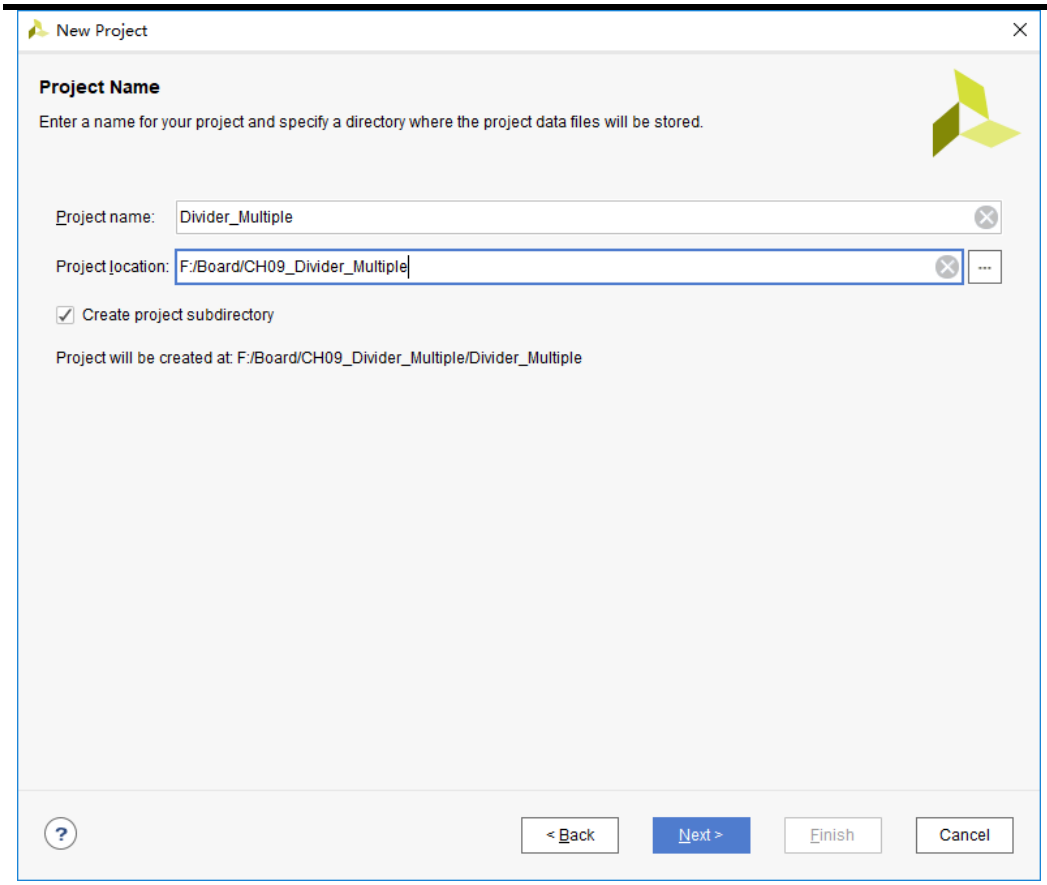
Step1:启动 VIVADO，单击 Create New Project



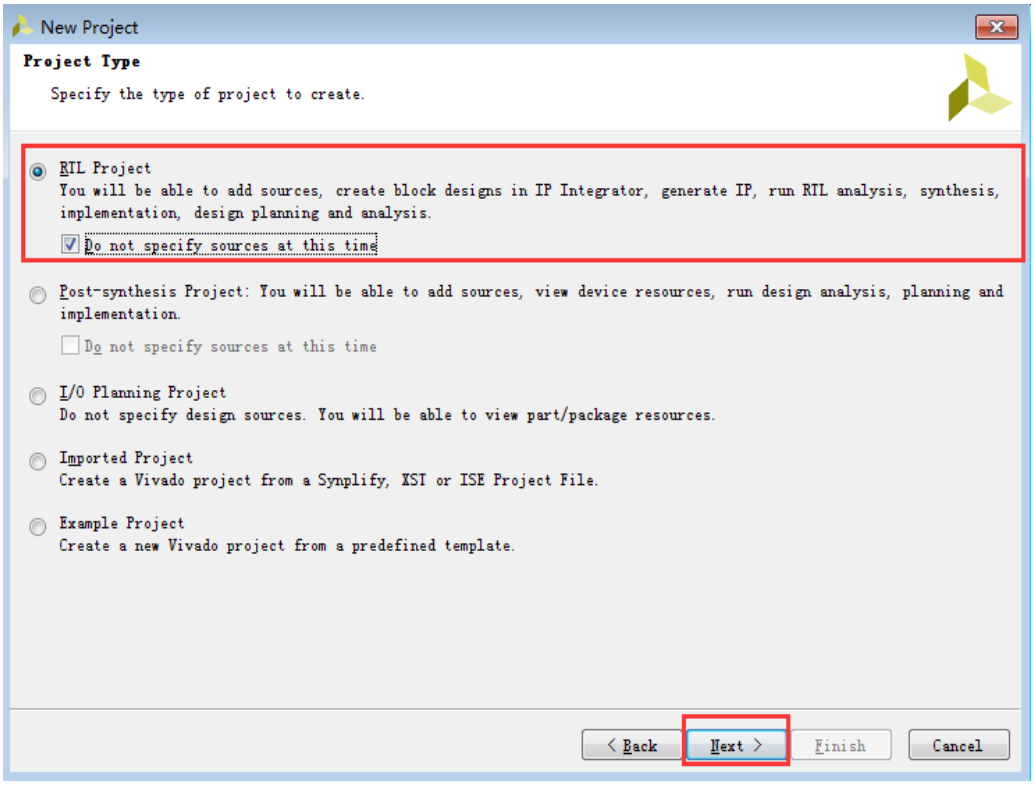
Step2:单击 NEXT



Step3:工程名字命名为 Divider_Multiple，并且设置保存的路径，单击 NEXT



Step4:新建一个 RTL 工程，并且勾选不要添加源文件，单击 NEXT

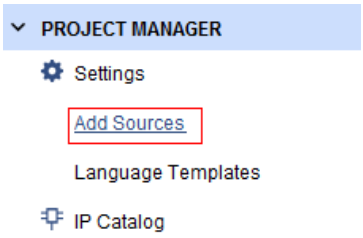


Step5: 选择芯片的型号和封装速度等级。

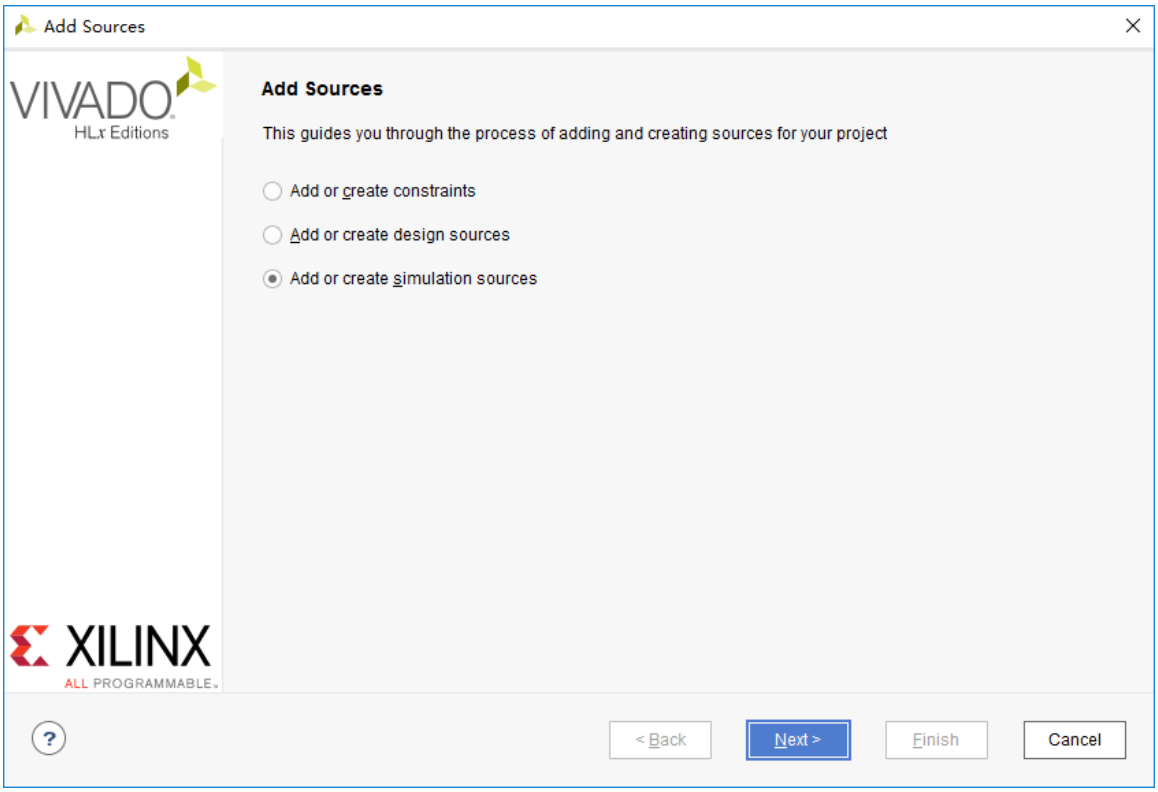
Step6:最后单击 Finish 完成工程的创建

1.3.2 添加仿真文件

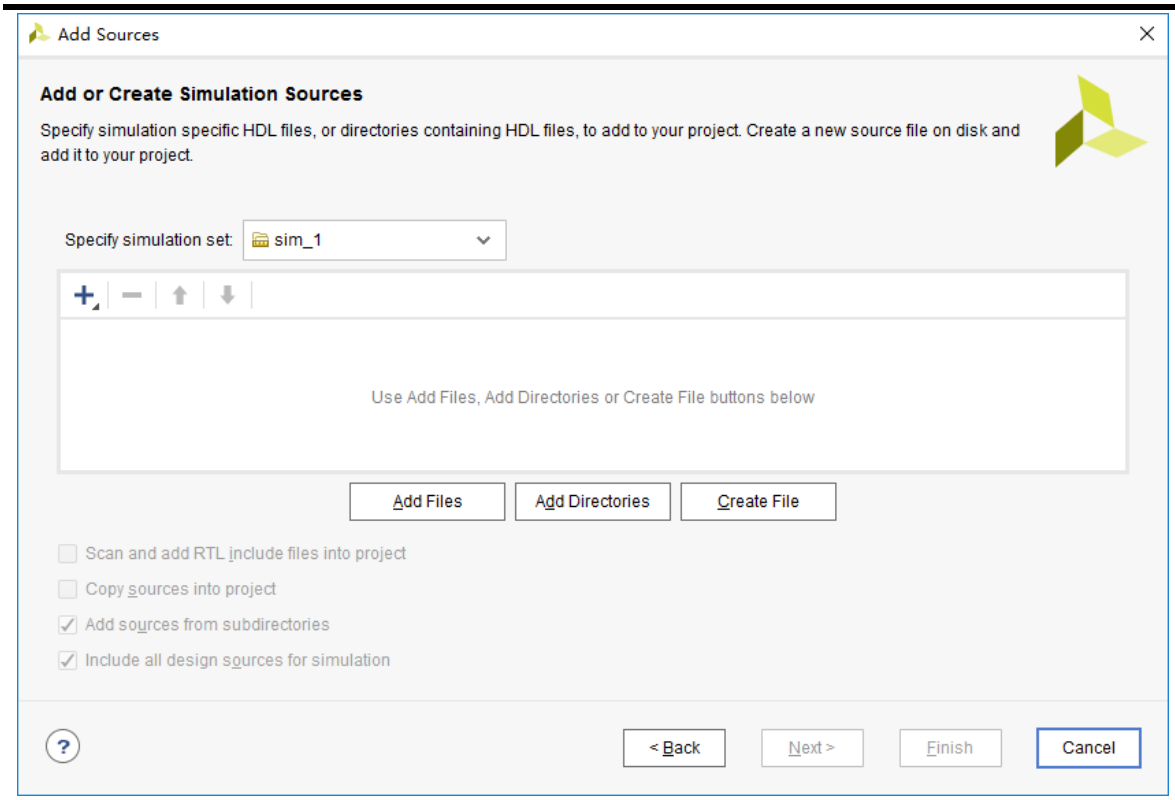
Step1:单击 Add Sources 添加仿真文件



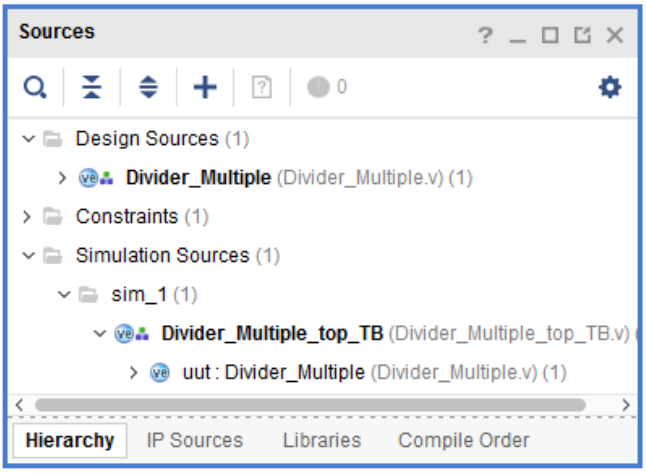
Step2:单击 Add Sources 添加仿真文件



Step3:单击 Add files 把仿真文件添加进来

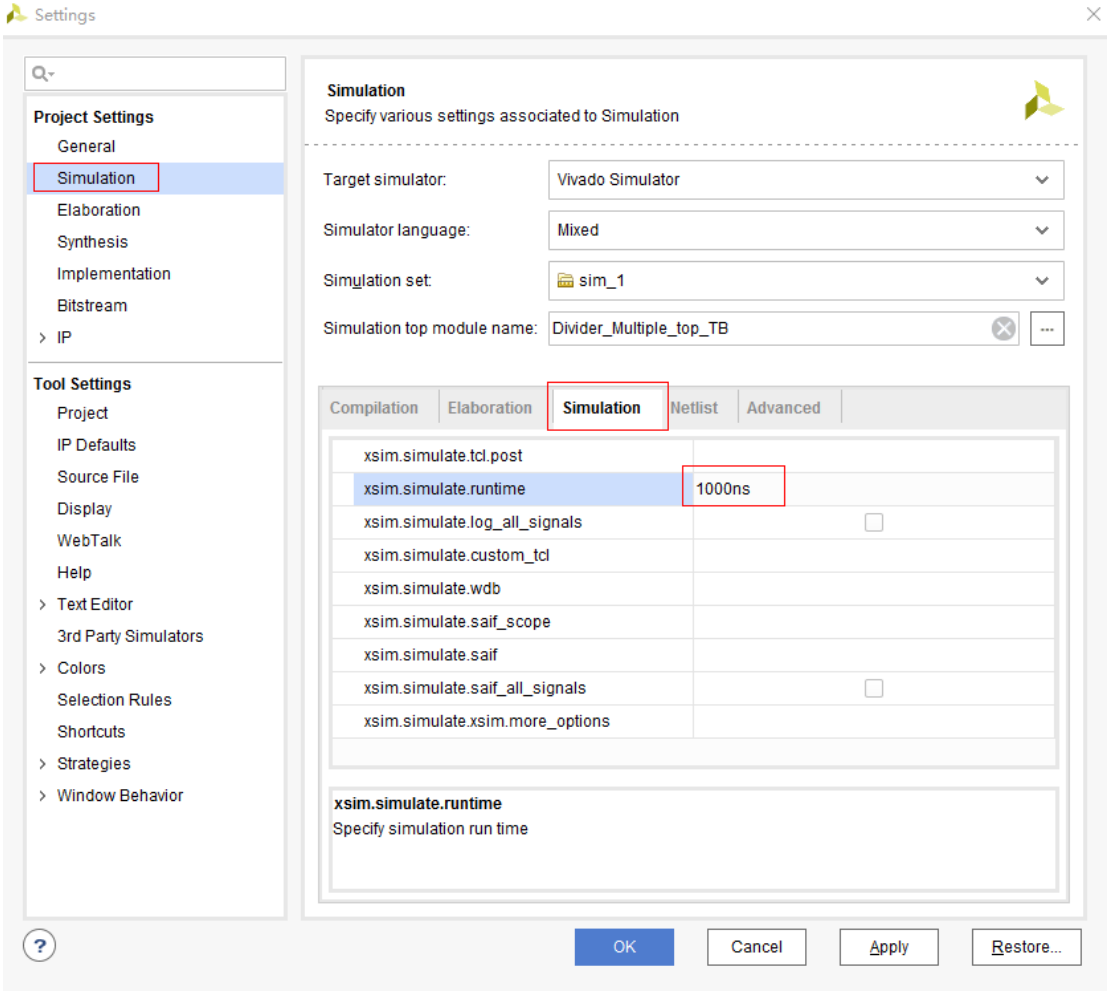


Step4:单击 Add files 把仿真文件添加进来



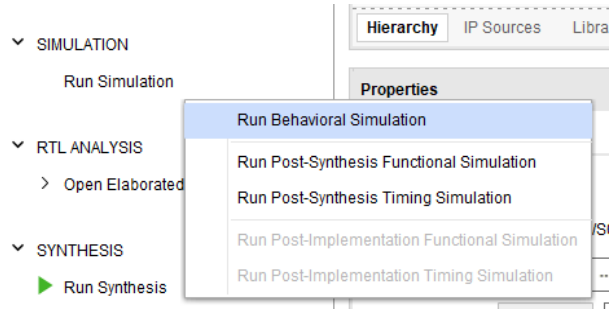
Step5:单击 Simulation Settings 对仿真参数做一些设置

Step6:单击 Simulation 设置仿真时间为 1000ms

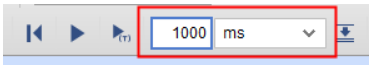


1.3.3 行为级仿真

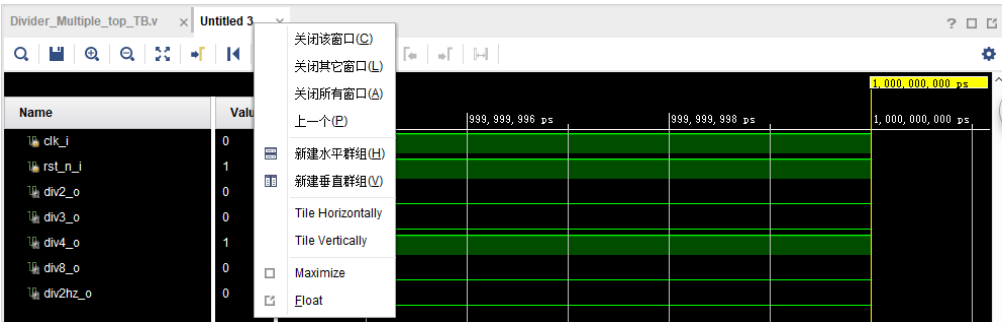
Step1: 进入仿真界面：SIMULATION->单击 Run Simulation ->单击 Run Behavioral Simulation。



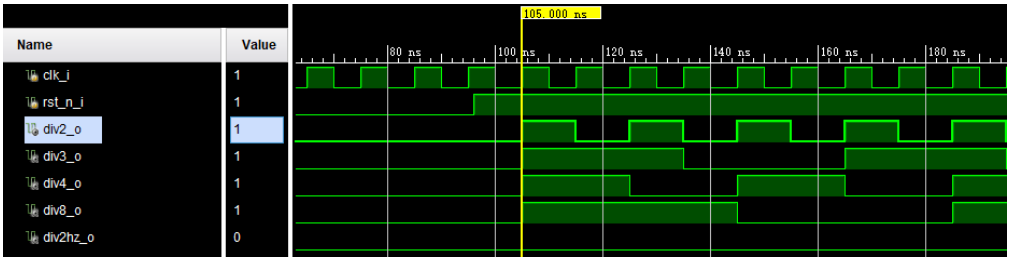
Step2:设置仿真时间，仿真时间为 1000ms。计算机 CPU 会模拟 FPGA 的运行，1000ms 运行来说通常需要几分钟时间。具体时间和 CPU 的配置有很大关系。



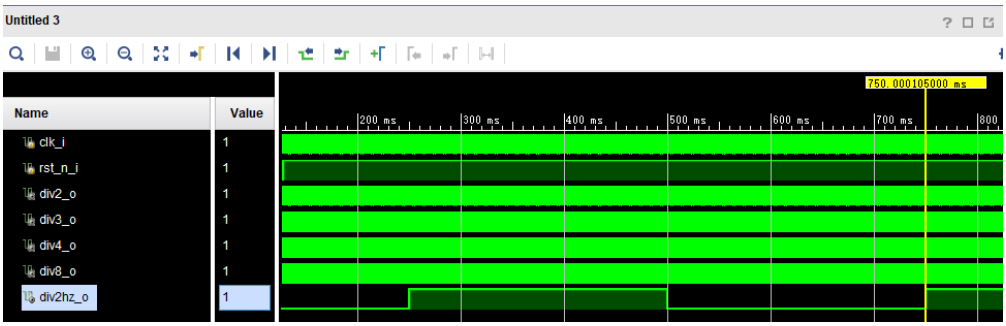
Step3:仿真结束后查看波形，为了观察方便，右击窗口选择 float



Step4:使用放大工具放大后观察



Step5:使用放大工具放大后观察

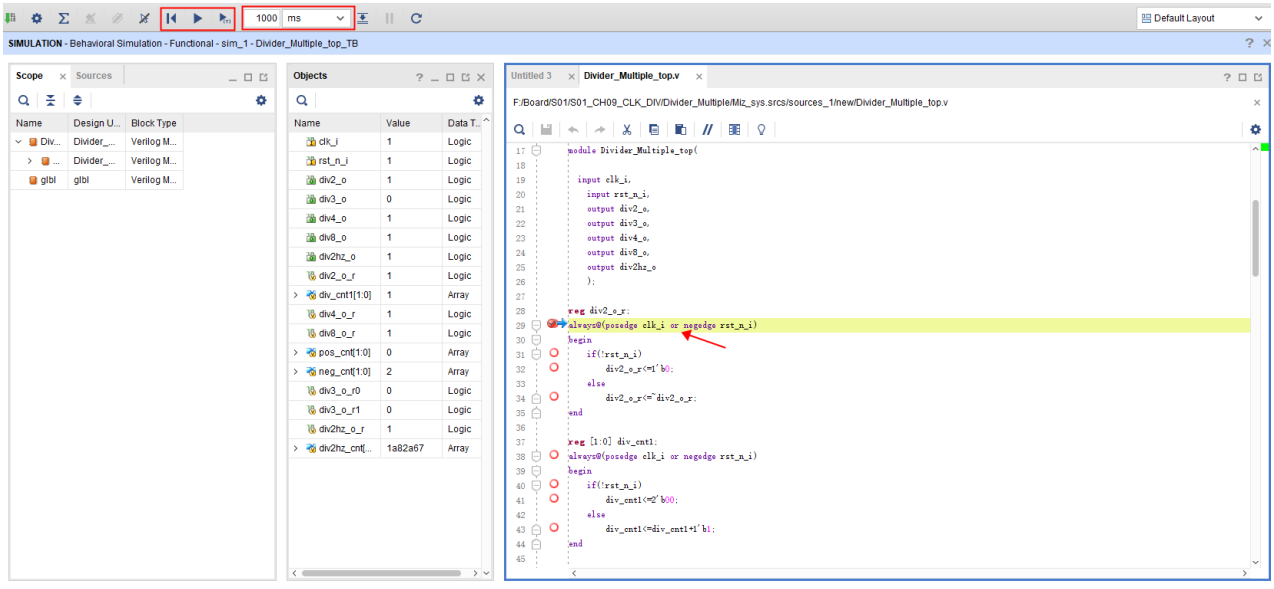


Step6:断点观察更多信号

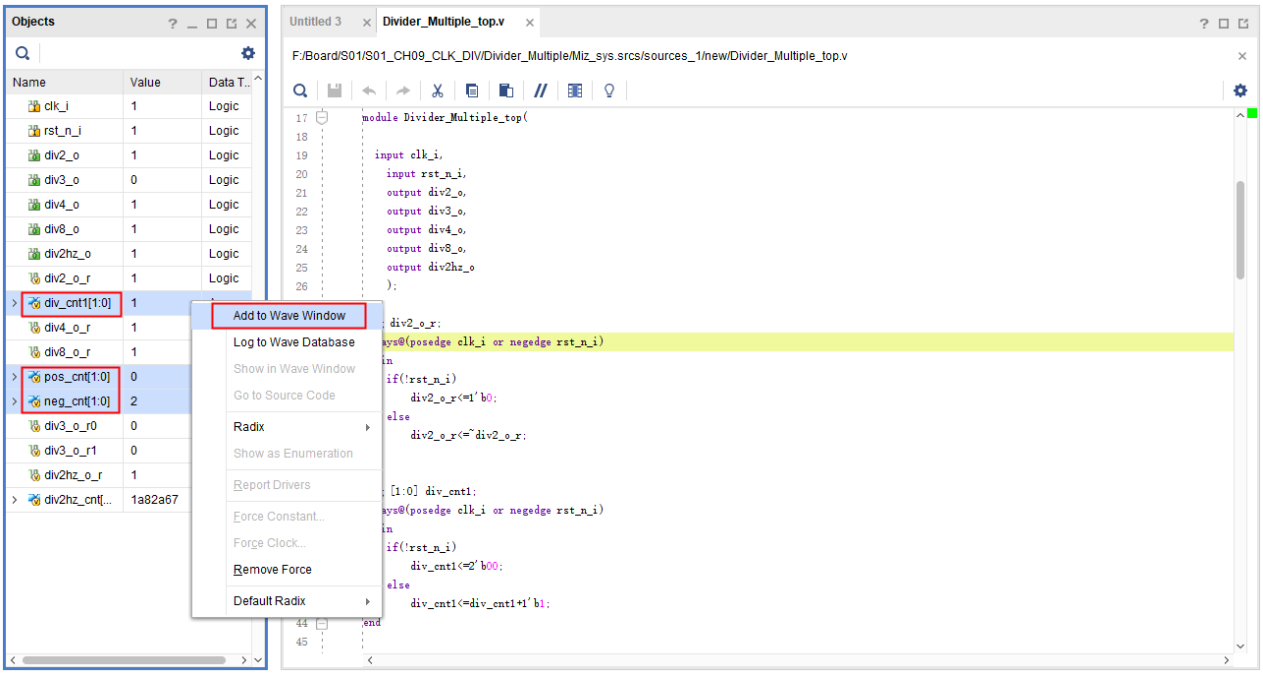
1、打开 divider_multiple_top.v 文件只要是显示红色圆圈的位置就是可以设置断点，单击红色圆圈。

2、单击运行

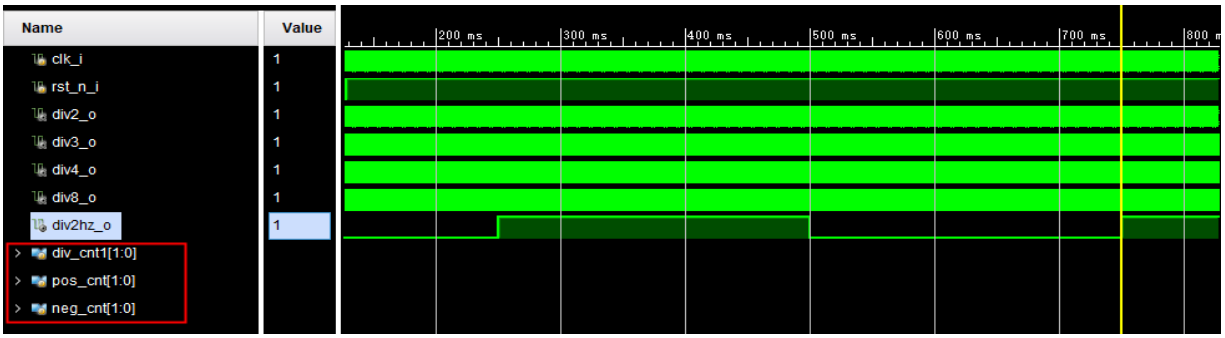
3、可以看到红色线框内有很多信号了，这些就是内部的运行信号，可以让我们观察程序更加仔细。



Step7:用鼠标单击这个几个信号，然后右击后单击 Add To Wave Window



Step8:可以看到我们添加进来的三个寄存器变量



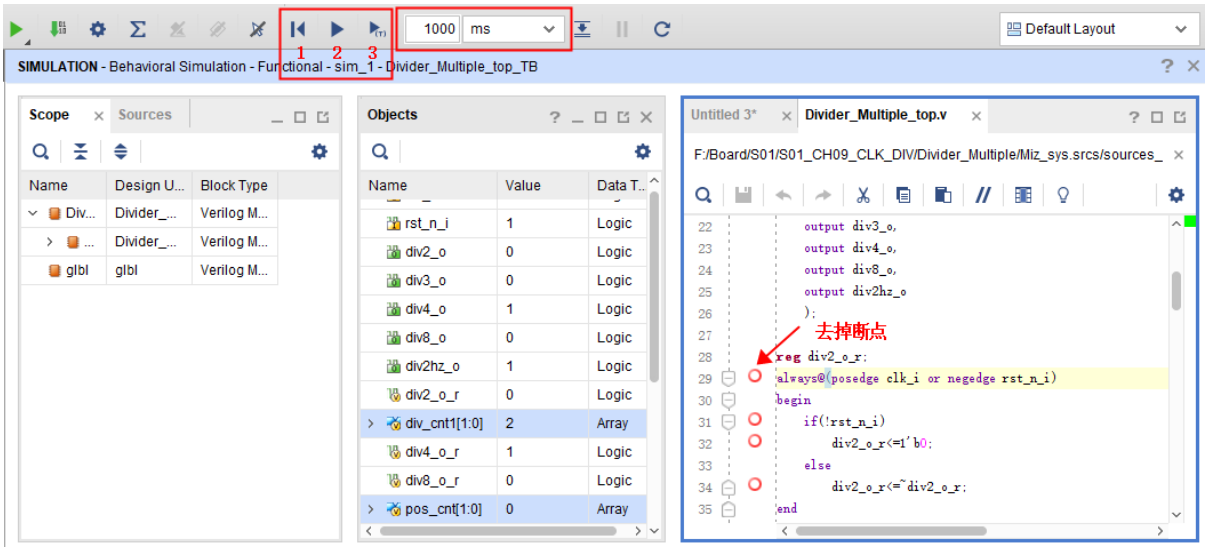
Step9:重新仿真 按钮 1 是初始化仿真 按钮 2 是仿真开始 按钮 3 是仿真到设置时间

1、去掉刚才设置的断点

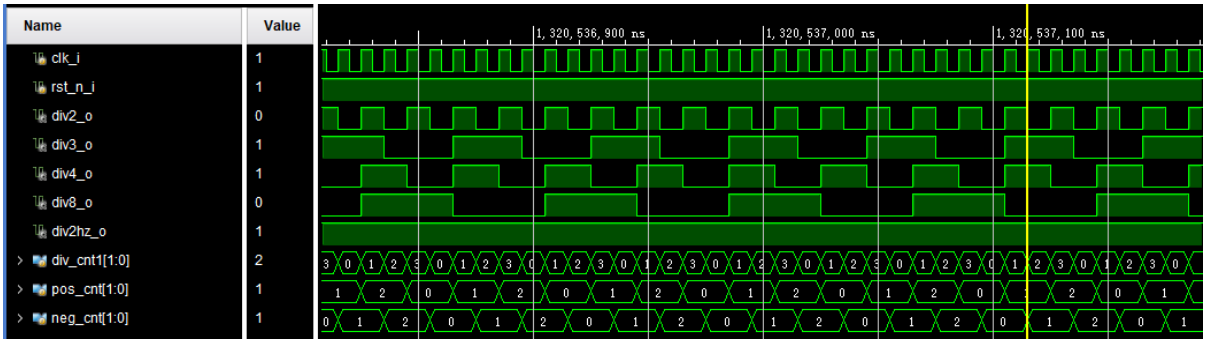
2、单击 1 处按钮重新加载初始化仿真

3、设置仿真时间为 1000ms

4、单击 3 处按钮



Step10:重新仿真后结果

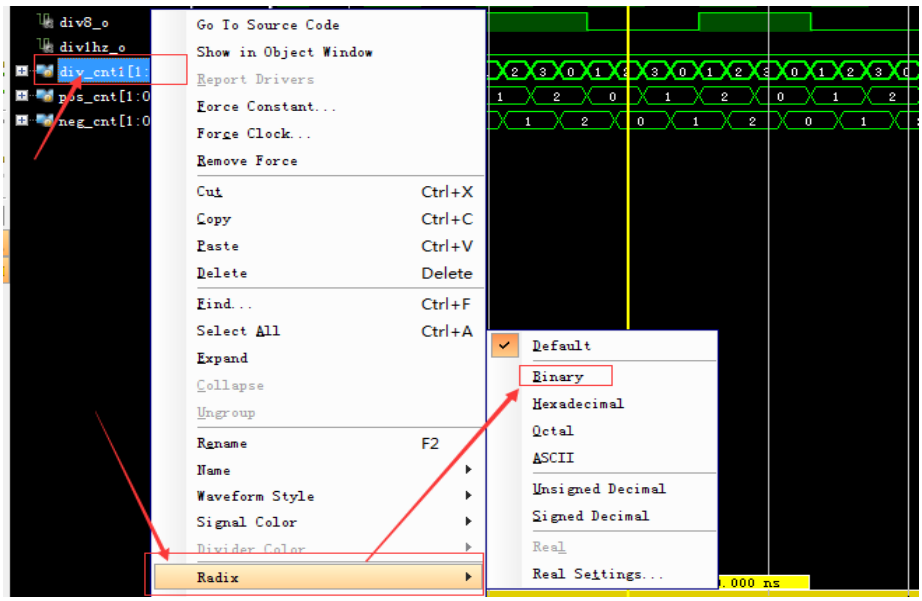


Step11:设置观察的数据类型

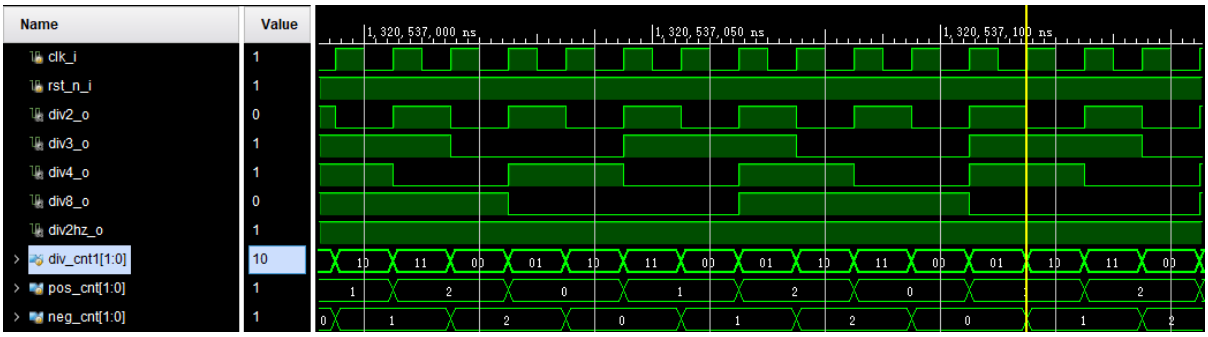
1、首先选择一个要观察的变量

2、右击选择 Radix

3、假设选择 Binary 以二进制形式观察



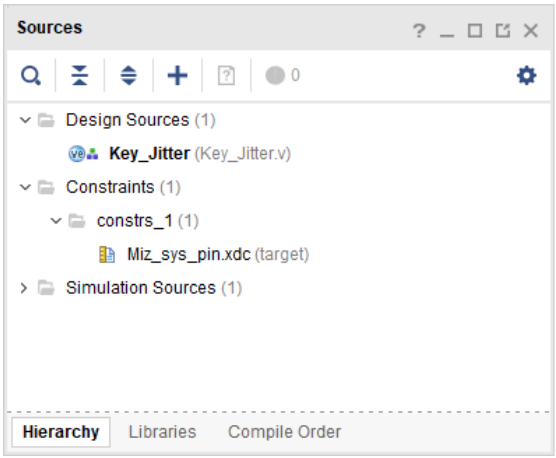
Step12:设置好后的效果



1.4 综合 Synthesis

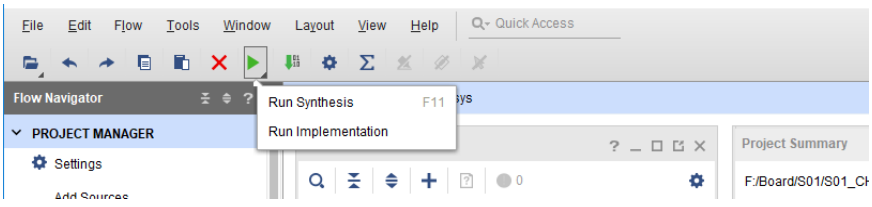
1.4.1 添加文件

Step1:把 Divider_multiple_top.v 添加进来，并且添加 xdc 管脚约束文件

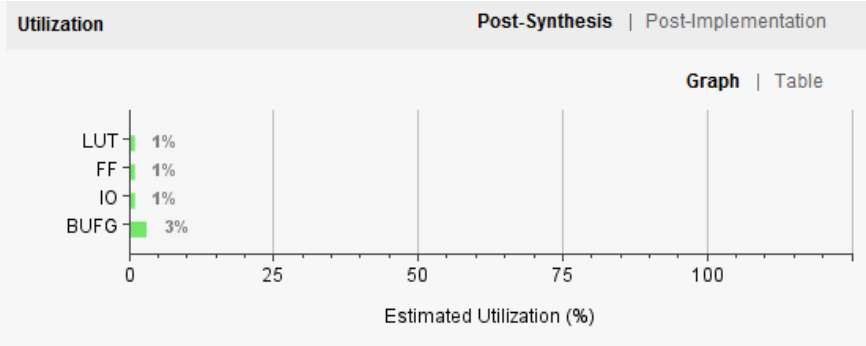


1.4.2 综合并查看报告

Step1:点击综合按钮



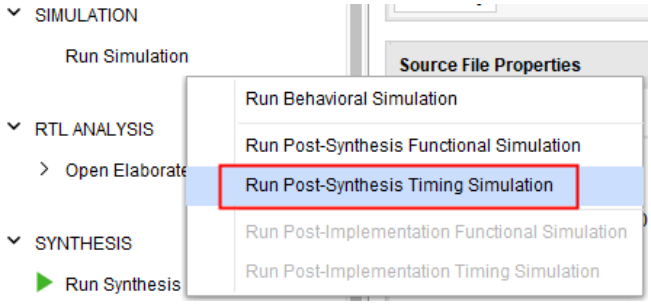
Step2:综合完成后通过查看报告看资源的利用情况



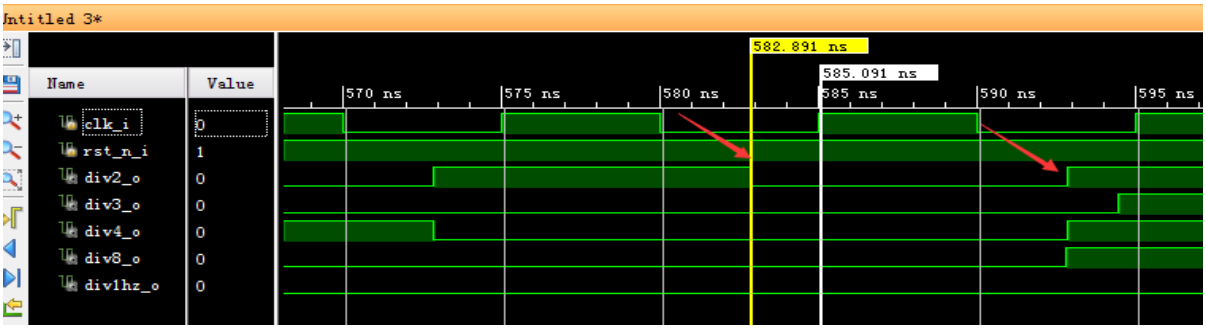
可以看到这个工程只是利用到了很少一部分资源

1.4.3 综合时序仿真

Step1:单击 Run Simulation 选择 Run Post-synthesis Timing Simulation



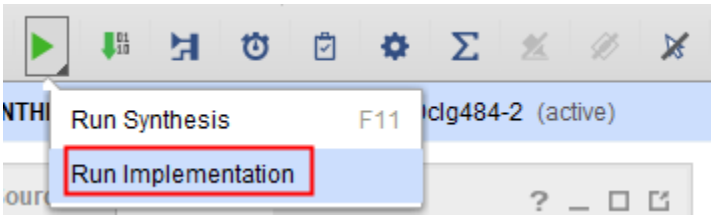
Step2:观察波形可以清晰看到综合后仿真加入了延迟更加接近实际芯片的运行情况



1.5 执行 Implementation

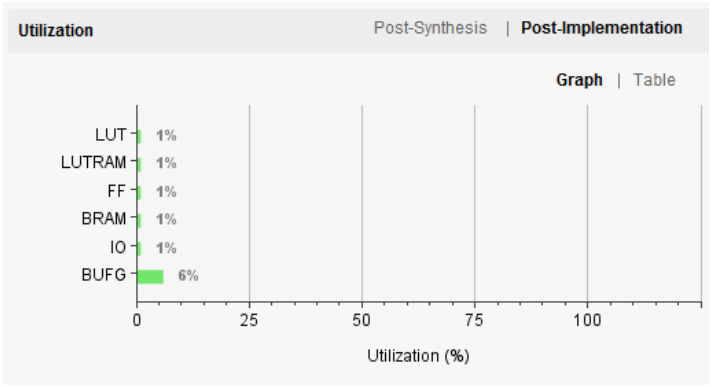
1.5.1 执行并查看报告

Step1:点击执行按钮

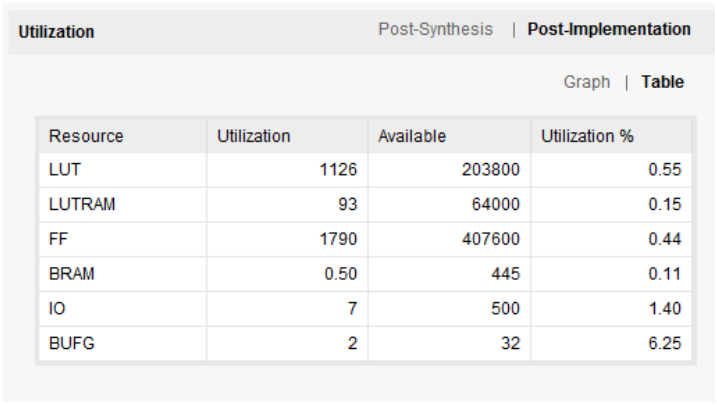


Step2:执行运行完毕后再次单击

Step3:查看执行运行完毕后的报告,执行完成后的报告比综合后的报告相比，是精确的分析和评估

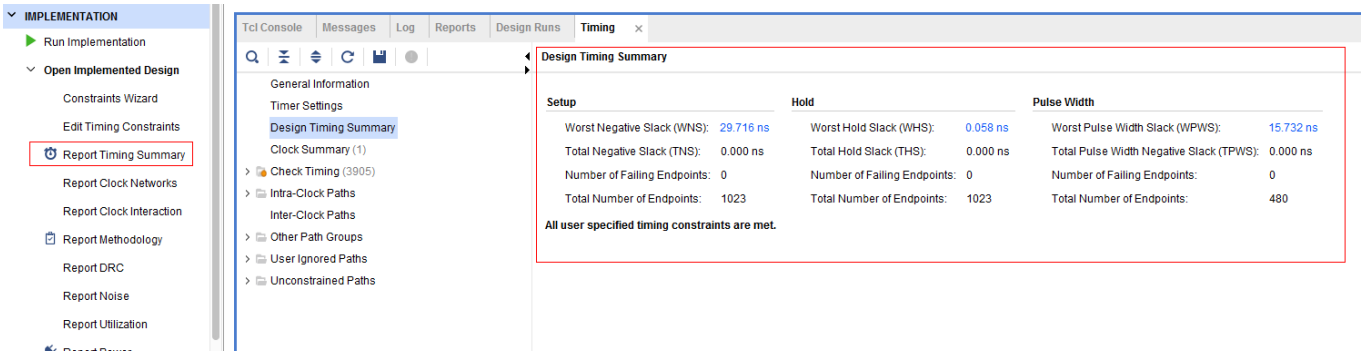


Step4:点开 Table 可以看到使用的资料的具体参数



Resource	Utilization	Available	Utilization %
LUT	1126	203800	0.55
LUTRAM	93	64000	0.15
FF	1790	407600	0.44
BRAM	0.50	445	0.11
IO	7	500	1.40
BUFG	2	32	6.25

Step5:查看执行完后的时序约束报告

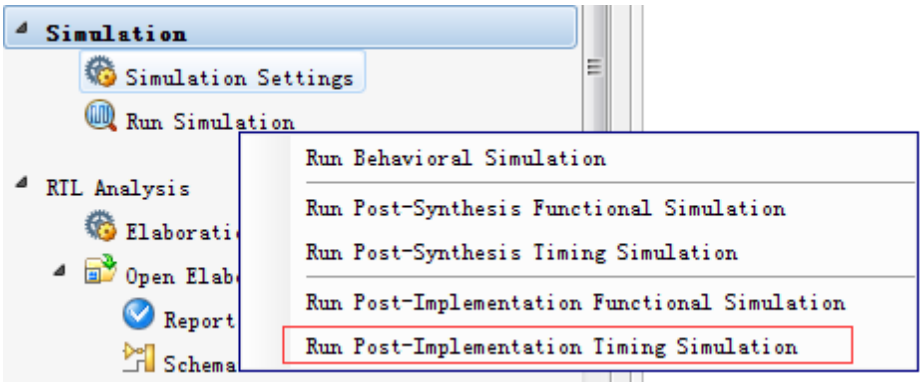


时序约束报告是 FPGA 开发中很重要的一项参数，所以必须看一下是否有违反时序约束的情况。可以看到有一些黄色的 waring。在这里不会影响我们的输出结果，因为我们这输出并没有做时序约束。但是如果要输出很严格的

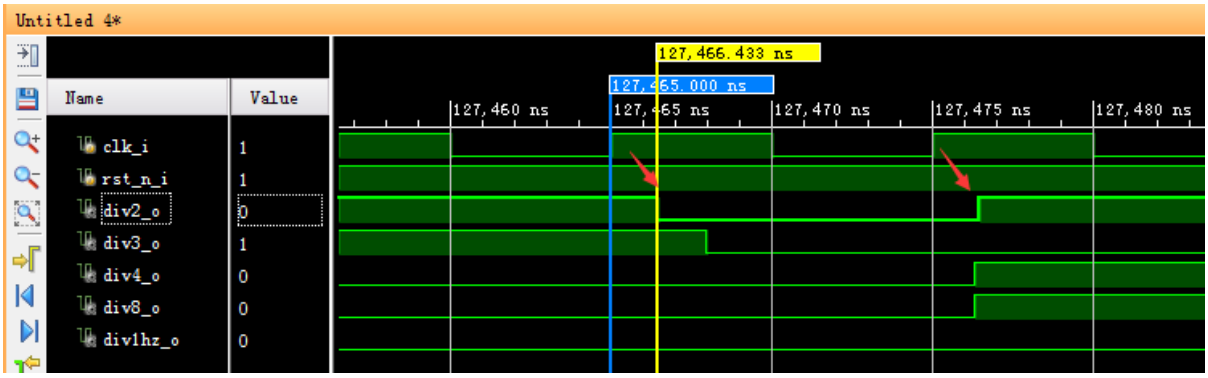
时序就需要加上时序约束。

1.5.2 布局布线后时序仿真

Step1:单击 Run Simulation 选择 Run Post-Implementation Timing Simulation



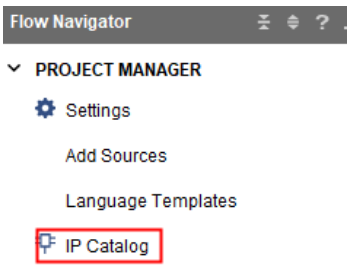
Step2:观察波形可以清晰看到布局布线后仿真加入了延迟这要比综合后的时序更加接近真实的情况



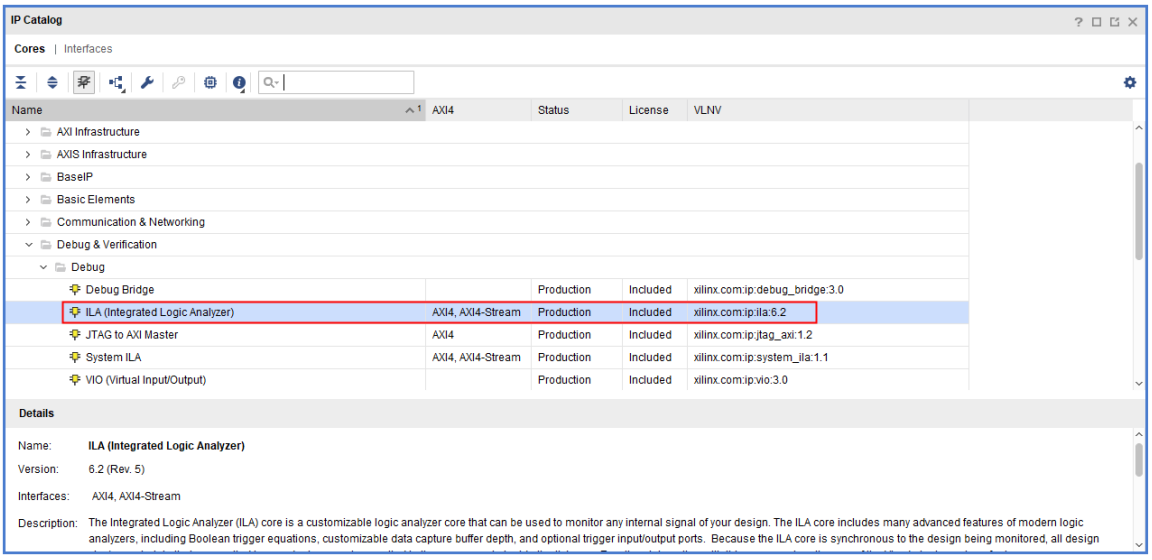
1.6VIVADO 在线逻辑分析仪使用

1.6.1 IP Catalog 添加 IA ip core

Step1: 单击 IP Catalog



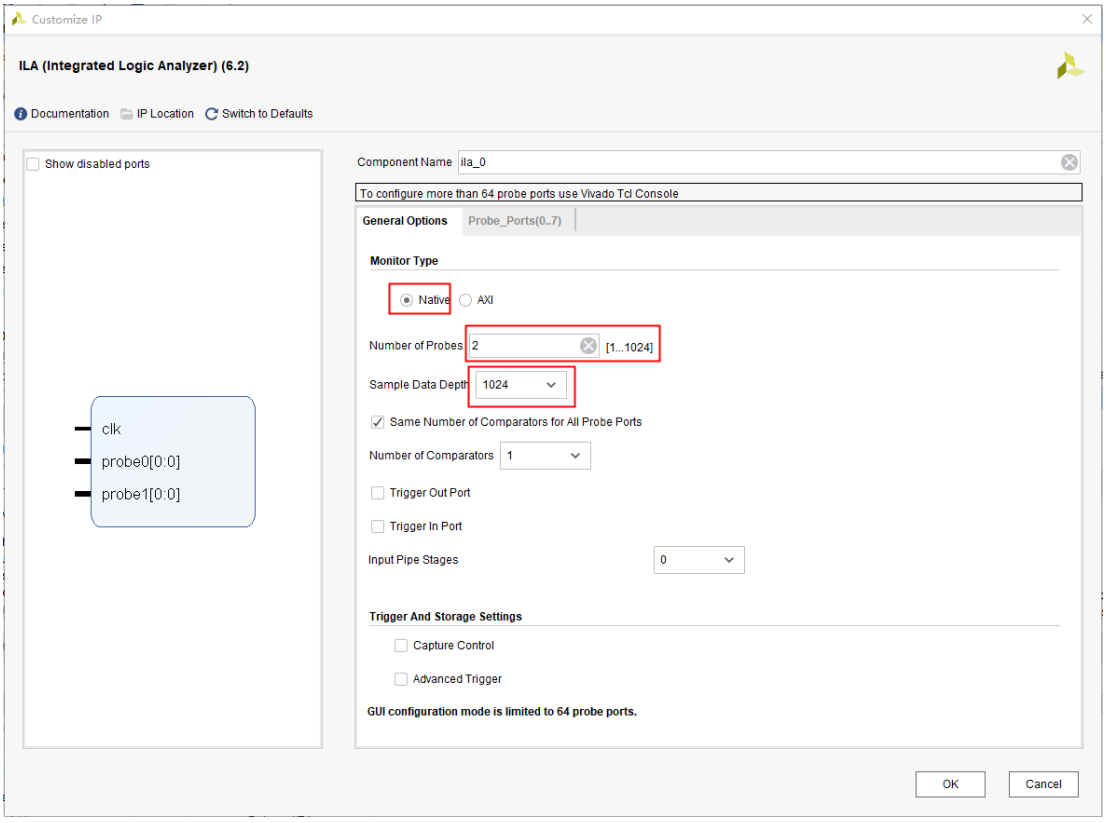
Step2: 打开 Debug & Verification > Debug ->双击 ILA



Step3:游标 General Options 设置如下

1、Number of probes 2 为设置需要观察信号的组为 2 组，因为我们准备 1 组放触发信号，1 组放普通观察的信号

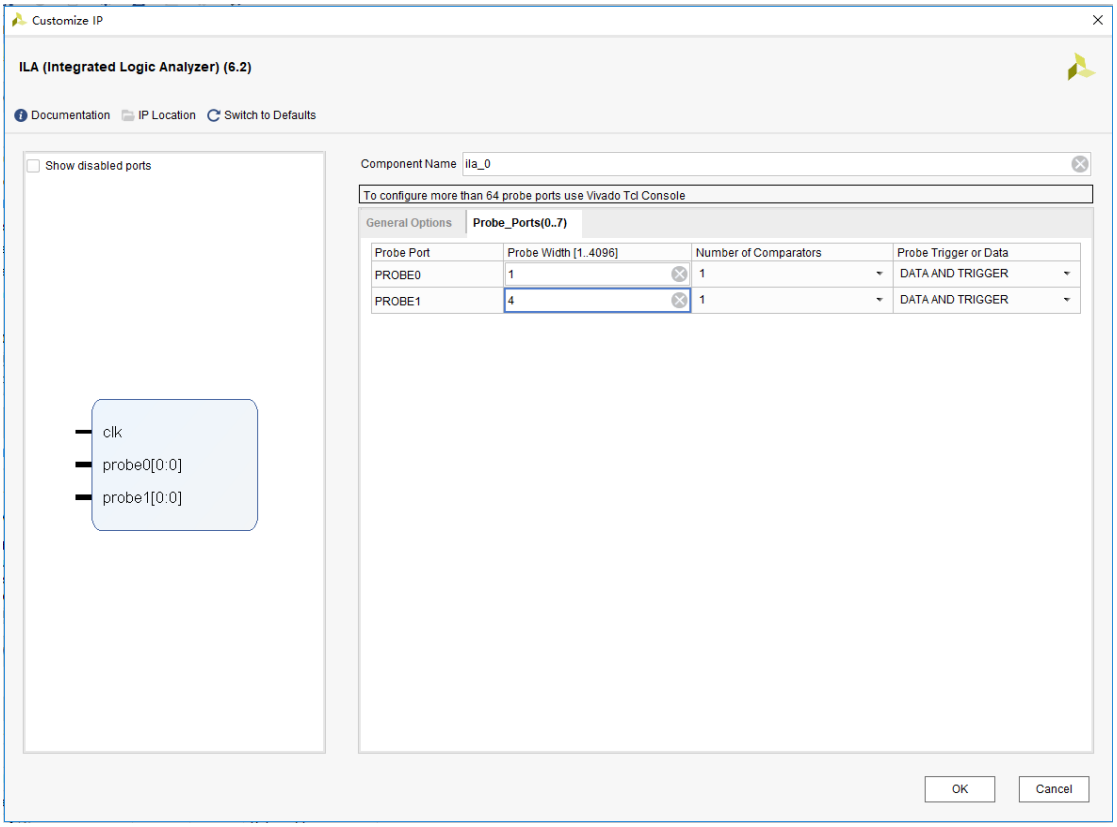
2、Sample Data Depth 1024 设置采样的深度，这是需要消耗 FPGA 的 BRAM 的 BRAM 越大可以设置的采样深度就越大，当然编译速度会降低。



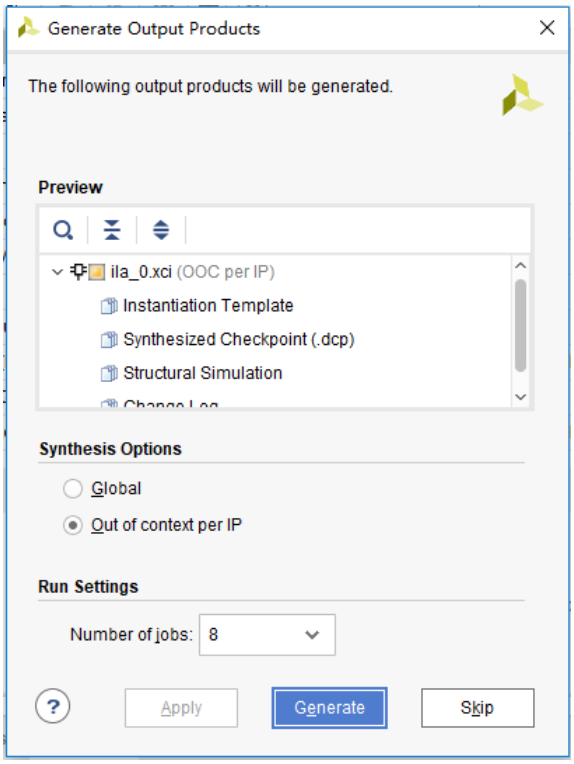
Step4:游标 Probe_Ports 设置如下

Probe Port 探针类似示波器的表笔，只是这里是在 FPGA 内部，我们设置了 Probe0 用来检查 2HZ 的信号，Probe1 用来检测另外 4 个分频信号。

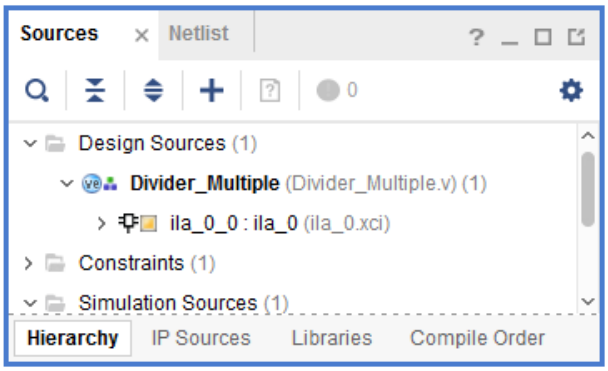
设置好后单击 OK 关闭窗口



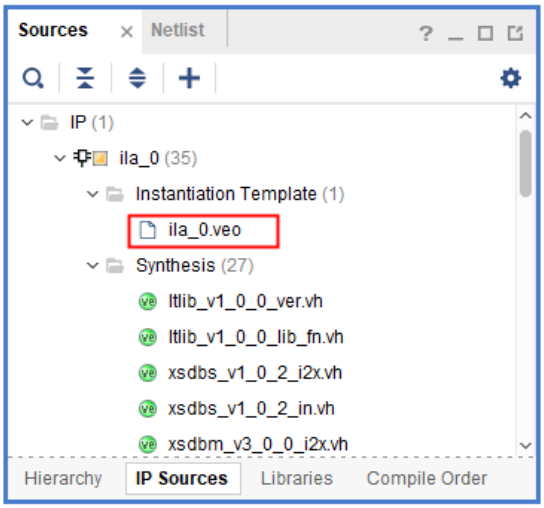
Step5: 直接单击 Generate



Step6: 可以看到 ila 这个逻辑分析仪的 IP 添加进来了



Step7:切换 IP Sources 游标下，然后双击 ila_0.veo 打开调用的接口模版



Step8:IP 接口调用模版打开后，可以看到这是一个 IP 接口，显然我们只要把需要被检测的信号根据前面的设置填进去就可以了。 clk 就是采样时钟，probe0 就是 2HZ 信号，proble1 就是其他需要被观察的信号。

```
56  ila_0 your_instance_name (
57      .clk(clk), // input wire clk
58
59
60      .probe0(probe0), // input wire [0:0]  probe0
61      .probe1(probe1) // input wire [3:0]  probe1
62  );
```

修改，并且嵌入到顶层文件中

```
ila_0 ila_0_0 (
    .clk(clk_i), // input wire clk
    .probe0(div2hz_o), // input wire [0:0]   probe0
    .probe1({div2_o,div3_o,div4_o,div8_o}) // input wire [3:0]   probe1
);
```

Step9: Run Synthesis-> Run Implementation->Generate Bitstream 生产 Bit 流文件。

1.6.2 逻辑分析仪抓取的信号

设置好逻辑分析仪，需要抓取的信号为

div2_o_r,

div3_o_r, div3_o_r0, div3_o_r1,

div4_o_r,

div8_o_r。

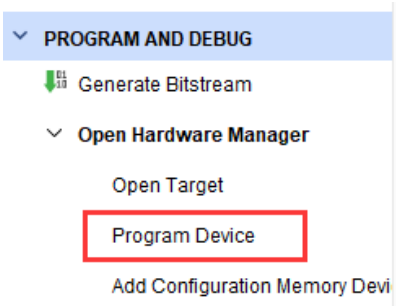
逻辑分析仪抓取的信号如下图所示。

1.6.3 逻辑分析仪使用

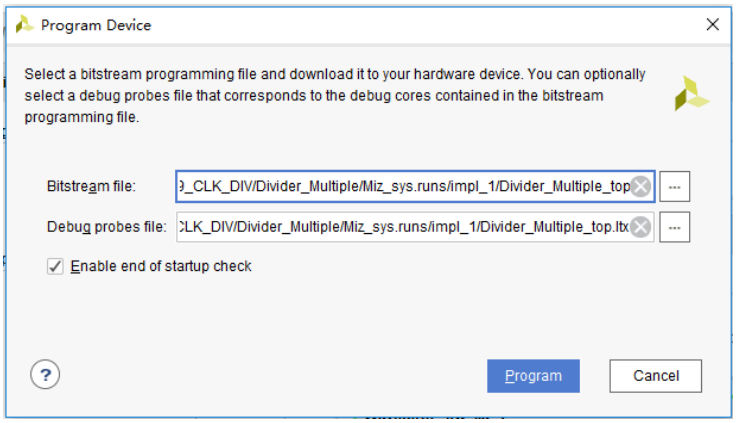
Step1:给开发板通电，并且连接下载器

Step2:单击 OpenTarget 然后单击 Auto Connect

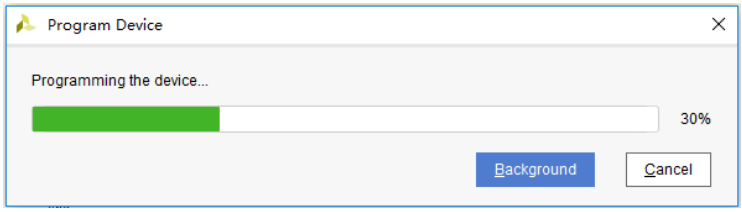
Step3:连接成功后，单击 Program Device



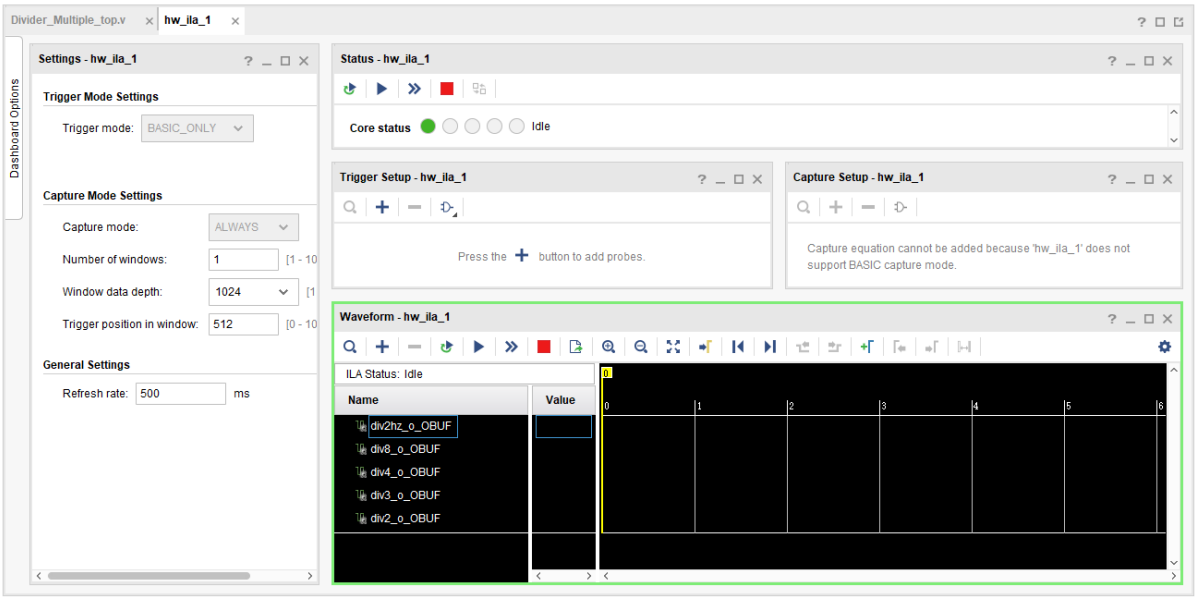
Step4:单击 Program Device ，弹出的对话框中有我们要下载的 Bit 文件



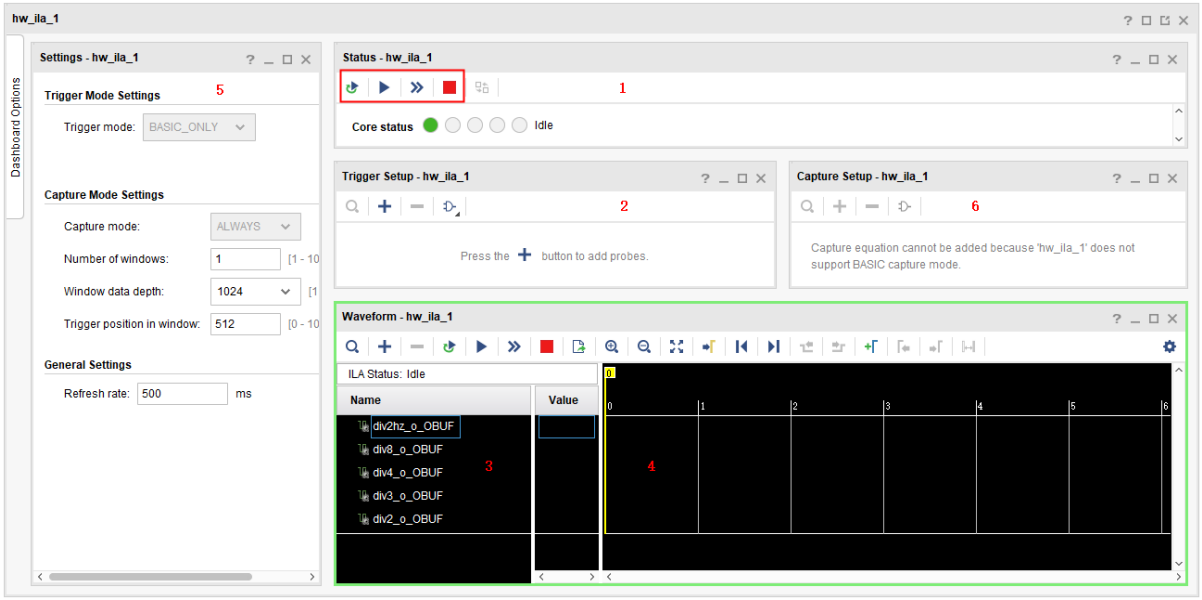
Step5:下载过程



Step6:下载后出现 Chipscope 界面



Step7: Chipscope 界面介绍



区域 1：设置采样的启动停止，和采样的方式

区域 2：设置触发信号

区域 3：被观察的信号名字

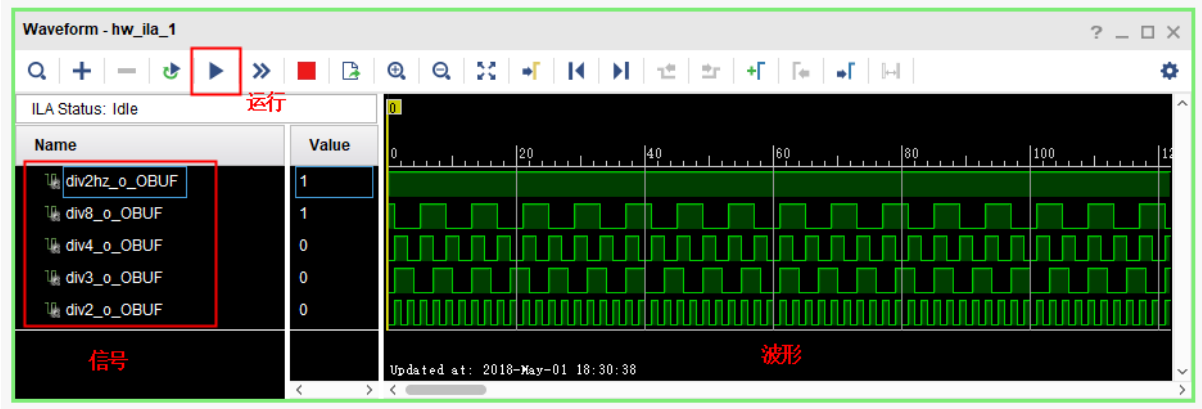
区域 4：被观察的信号波形

区域 5：触发模式设置

区域 6：触发设置

那么我们主要使用的有 1、2、3、4 这个几个区域。

Step8: Chipscope 运行



1.7 小结

本章全面介绍了 VIVADO 的 FPGA 开发流程规范。包括了程序设计、行为仿真、综合过程、综合后时序仿真、执行过程、执行后仿真、FPGA 资源的利用情况分析、利用 VIVADO 自带的逻辑分析仪抓取信号波形，进行分析、IAL 逻辑分析仪 IP 的使用和设置。