
Vivado 基础实验

基于 Nexys4 DDR 板的跑马灯实验

东南大学计算机学院
软件学院
2015. 10

基于 Nexys4 DDR 板的跑马灯实验

1 引言

本实验将使用 Vivado 工具，利用一个自制的 8 进制计数器 IP 核、一个时钟分频器和一个简化的 38 译码器来实现一个每秒变换一次的跑马灯电路。实际上，做跑马灯完全不用这么复杂，但我们希望通过这个实验，您将学会 Vivado 的设计流程，IP 核的封装与使用。所有代码均用 Verilog HDL 语言编写。

2 自制一个 8 进制计数器的IP核

2.1 创建一个新项目

双击  打开 Vivado，然后点击  创建一个新项目（或者在菜单栏选择 **File** -> **New Project...**）。如图 1 所示。

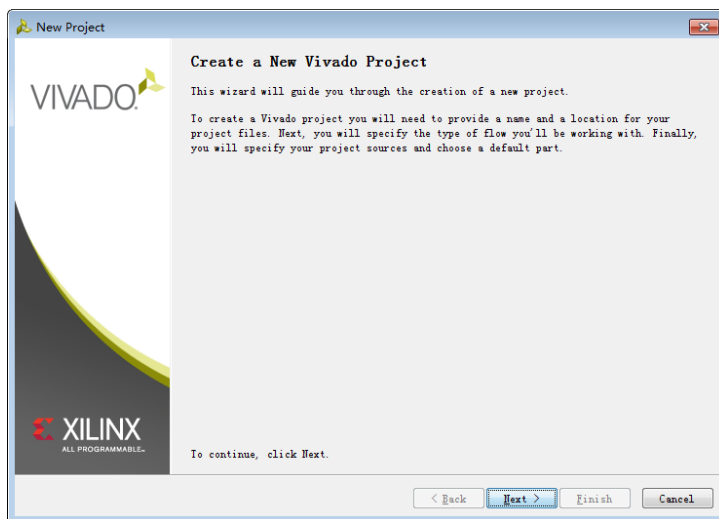


图 1 New Project

点击 **Next**。显示如图 2 所示的界面。按图 2 中所示命名项目名称和路径。这里项目名称为 ctc8，项目的位置是 D:/MIPSfpga_Fundamentals/Xilinx/VivadoProject，点击 **Next**。最后，整个项目将在 D:/MIPSfpga_Fundamentals/Xilinx/VivadoProject/ctc8 中

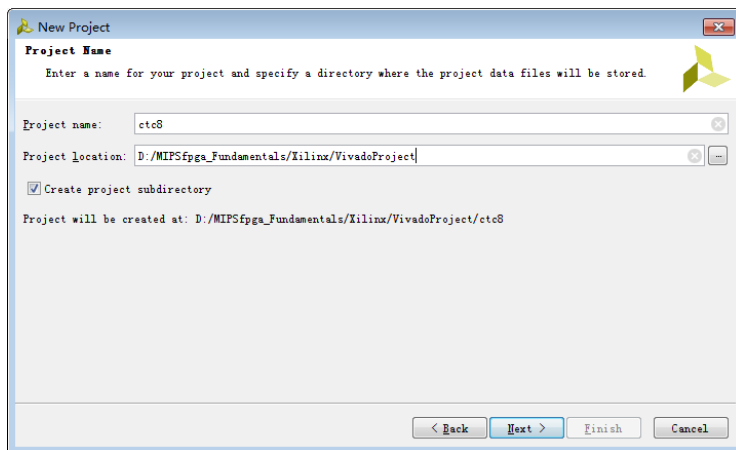


图 2 项目名称

如图 3 所示设置选择项目类型。点击 Next

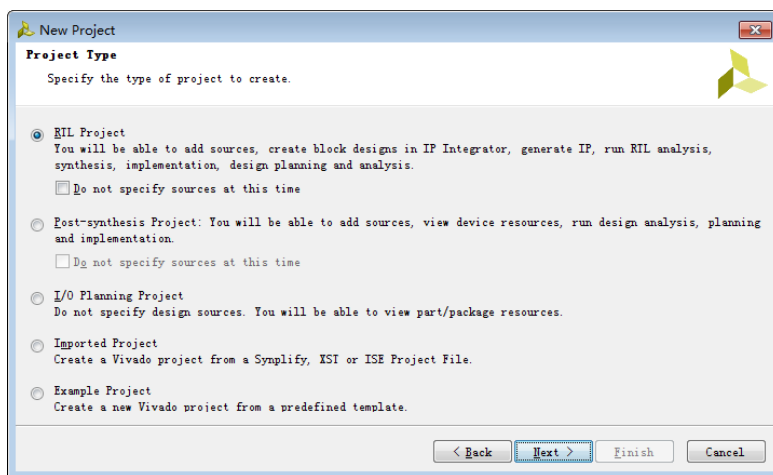


图 3 项目类型

因为不需要增加源文件，所以 Add source 窗口点击 Next(但 Target Language 选择 Verilog)。
不增加 IP 核，所以 Add Existing IP 窗口点击 Next。
不增加约束文件，所以 Add Constraints 窗口点击 Next。
按图 4 选择器件为 xc7a100tcsg324-1。点击 Next。

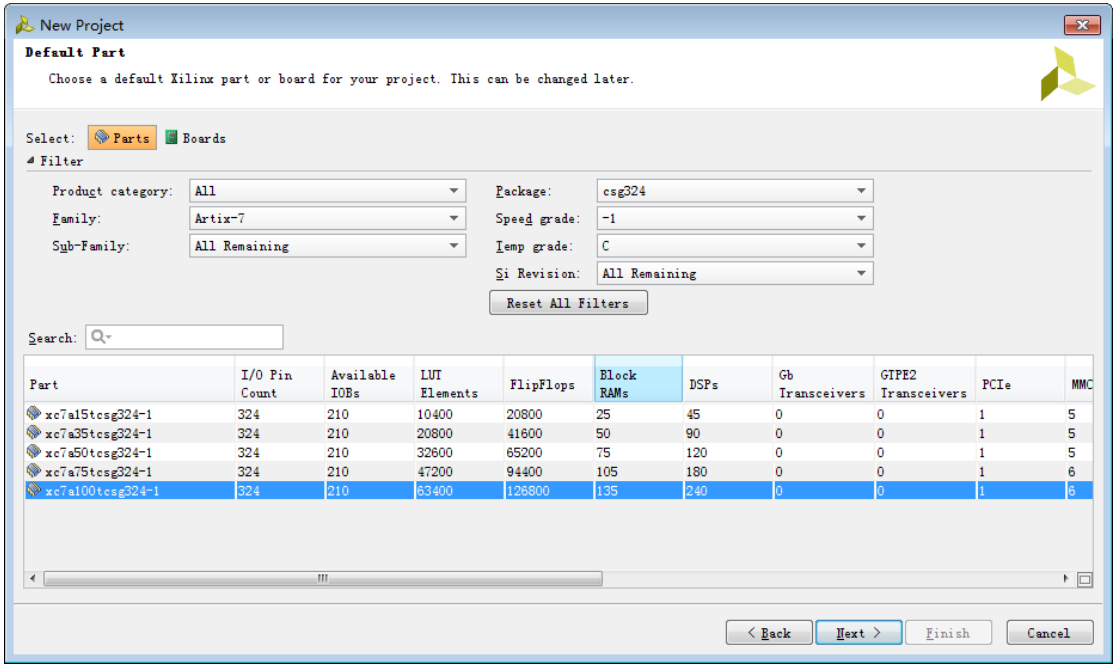


图 4 选择器件

可以看到如图 5 所示的新项目概览。点击 **Finish**。

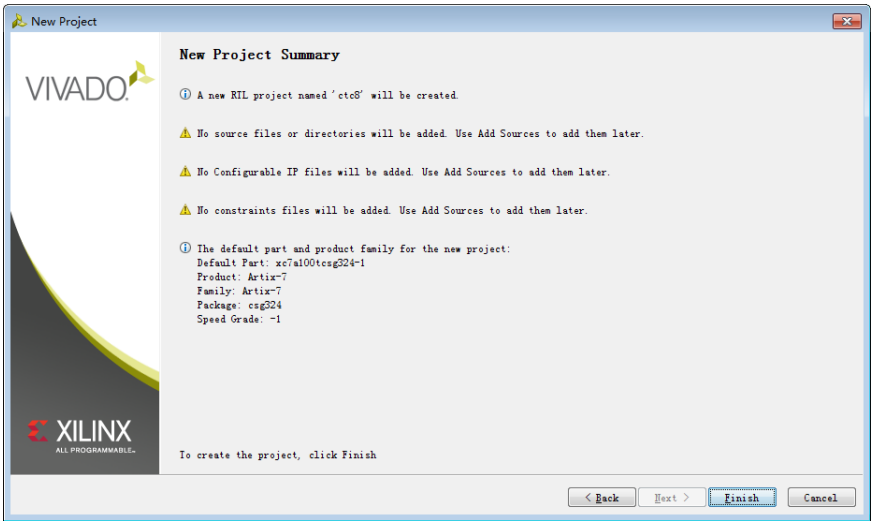


图 5 新项目概览

2.2 添加源代码

在图 6 所示的窗口中右键点击 **Design Sources**（图中高亮处），在弹出的菜单中选择 **Add Sources....**，出现图 7 所示的对话框。

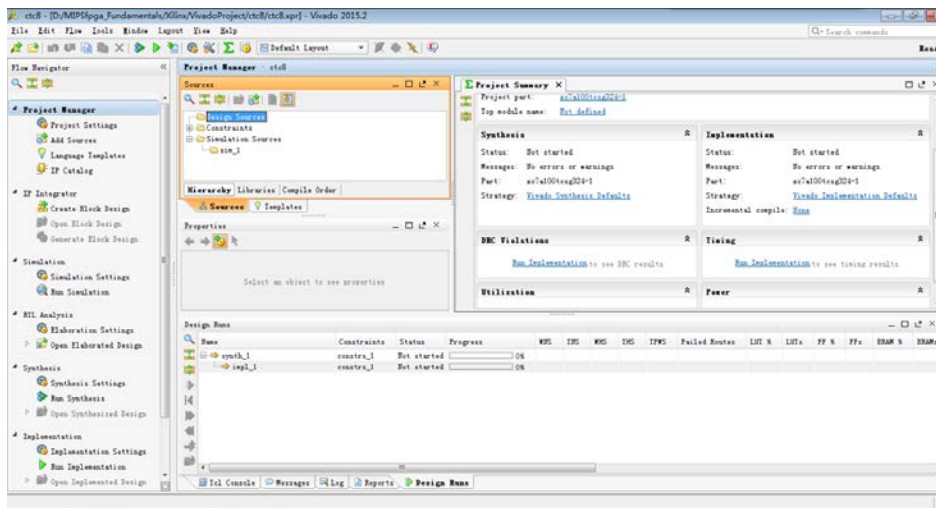



图 6 创建新幕后的界面



图 7 添加源程序对话框

按照图 7 所示选择后点击 **Next**。在接下来打开的对话框（如图 8 所示）中点击 ，并选择 **Create File...**。

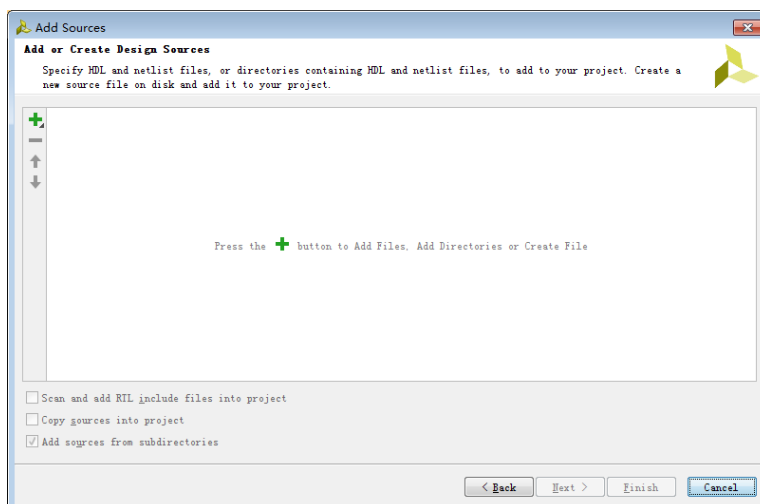


图 8 添加或创建设计文件对话框

此时会看到 Create Source File 对话框，按照图 9 所示填写后点击 **OK**。

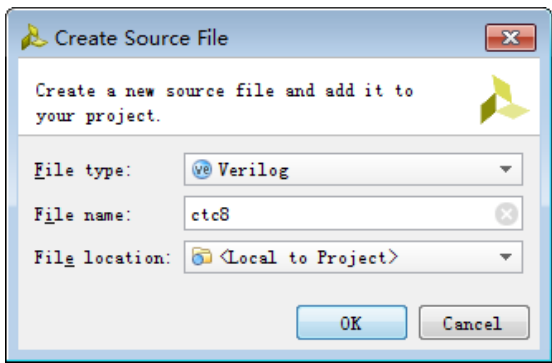


图 9 Create Source File 对话框

此时会看到图 10 所示的界面。

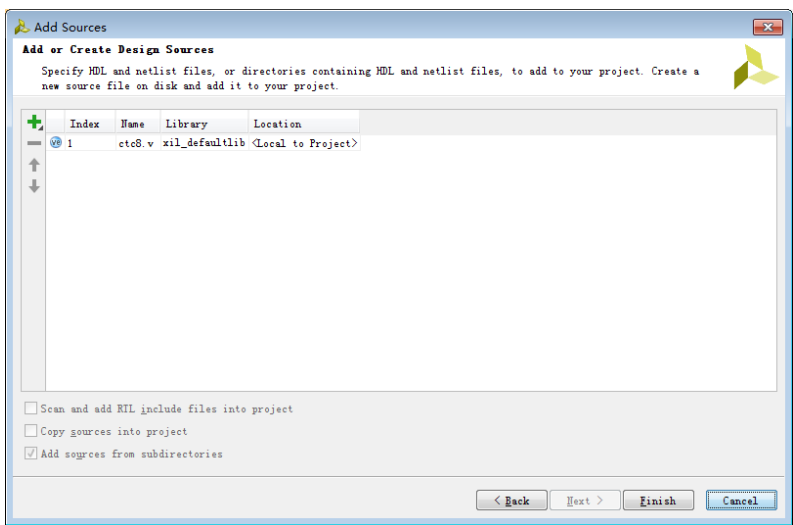


图 10 创建设计文件后的添加或创建设计文件对话框

点击 **Finish**。在接下来的如图 11 所示的 Define Module 对话框中选择 **OK**。在弹出的对话框(如图 12 所示)中点击 Yes。

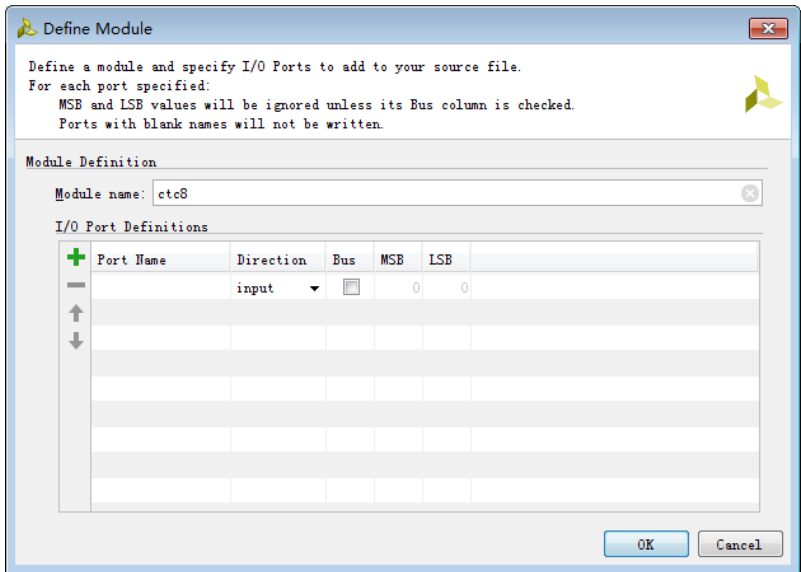


图 11 Define Module 对话框

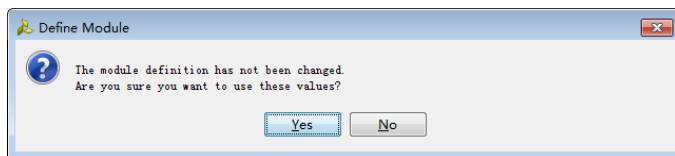


图 12 Define Module 确认框

接下来在图 13 所示界面上双击 `ctc8` 文件（图中高亮部分）就会在右边显示初始的 `ctc8` 文件内容。

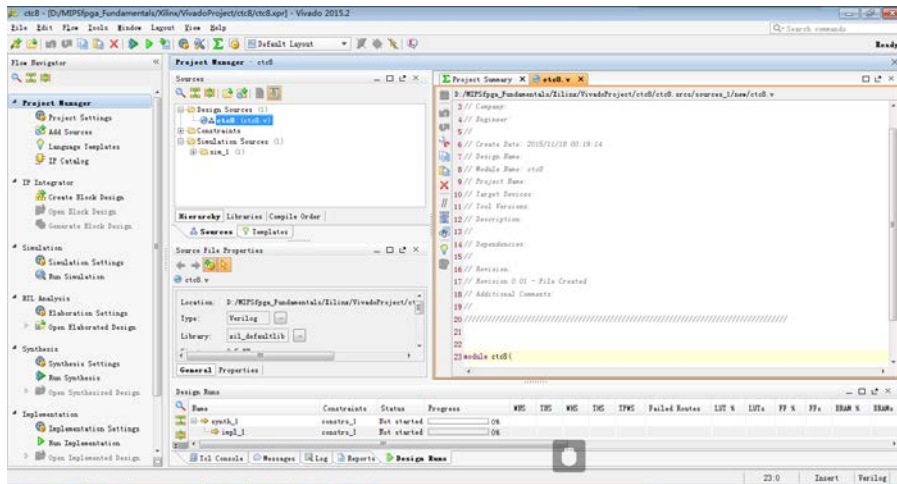


图 13 打开 `ctc8.v`

我们可以看到文件中 `ctc8` 的模块是空的。

```
module ctc8(

);

Endmodule
```

用下面的程序将这个空模块替代掉。



```
module ctc8(
    clk,
    resetn,
    count
);
    input clk, resetn;
    output reg[2:0] count;

    always @(posedge clk or negedge resetn) begin
        if(resetn == 0) begin
            count = 0;
        end else begin
            if(count == 0)
                count = 7;
            else begin
                count = count - 1;
            end
        end
    end
endmodule
```

```
end
end
endmodule
```

这段程序描述了一个 8 进制计数器，每次时钟上升沿到来，计数器减 1，减到 0 后重新开始计数。Reset 为低电平的时候复位。

2.3 综合

代码写好以后，点击  按钮或在 Project Manager 中点击  Run Synthesis 进行综合。如果出现图 14 所示对话框，点击 **Save**。

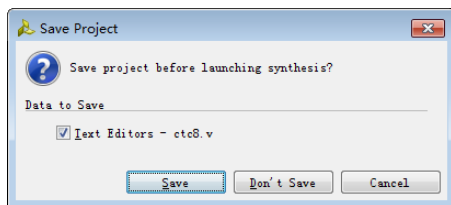


图 14 Save Project

综合完成后，如果出现图 15 所示对话框，选择 **Cancel**。

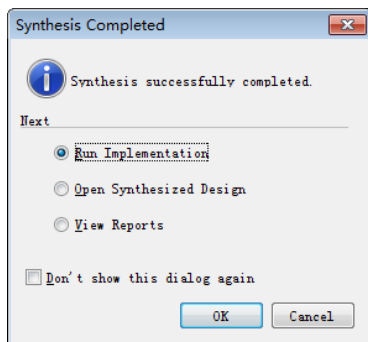


图 15 综合结束

2.4 封装IP核

在如图 16 所示的的界面中点击 **Project Settings**。

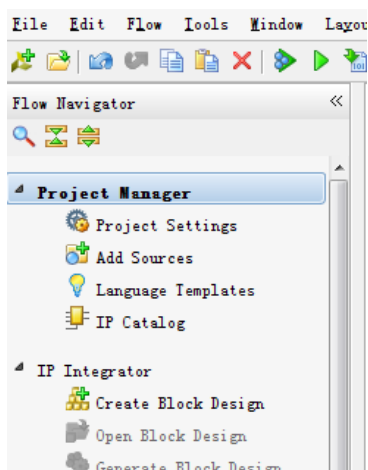


图 16 Project Manager

在 Project Settings 对话框中选择 IP，并进入 Packager 选项卡，如图 17 所示进行设置。设置好后，点击 Apply，然后点击 OK。记住这里设置的各个属性。

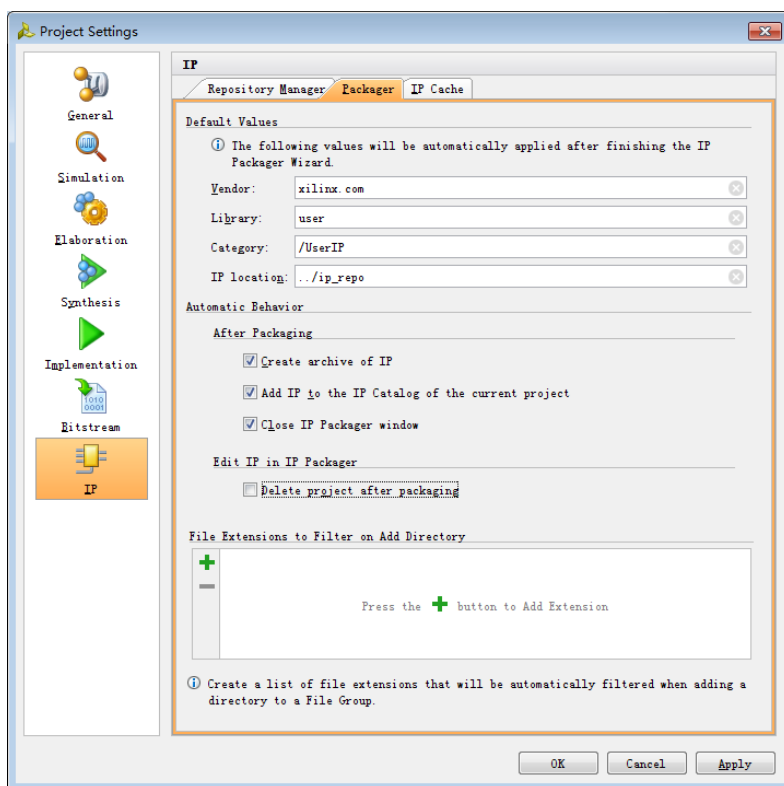


图 17 项目设置中设置 IP 核封装属性

在 Vivado 的菜单栏中选择 Tools->Create and Package IP...。在弹出的窗口中点击 Next。在之后弹出的窗口中如图 18 所示设置封装选项。点击 Next。

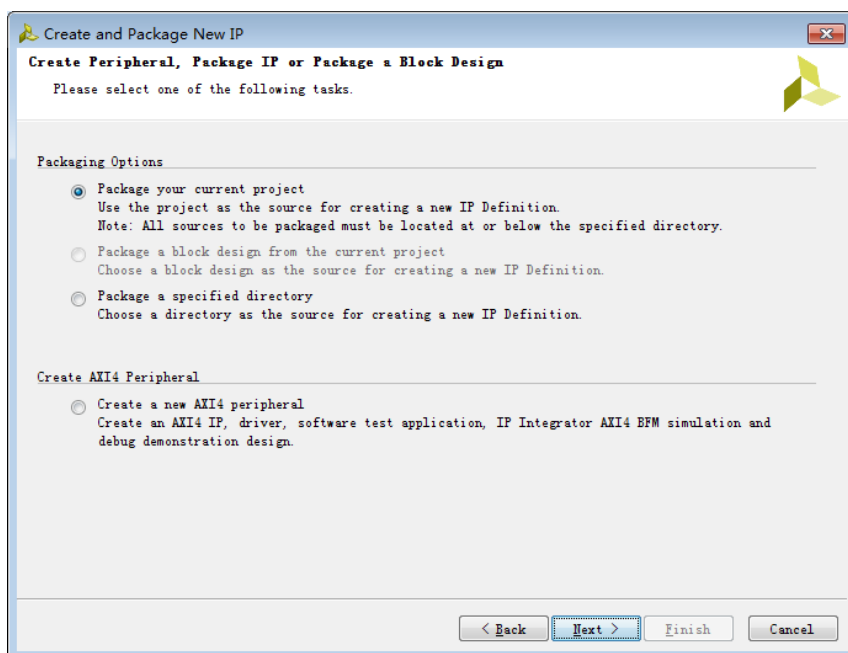


图 18 封装选项

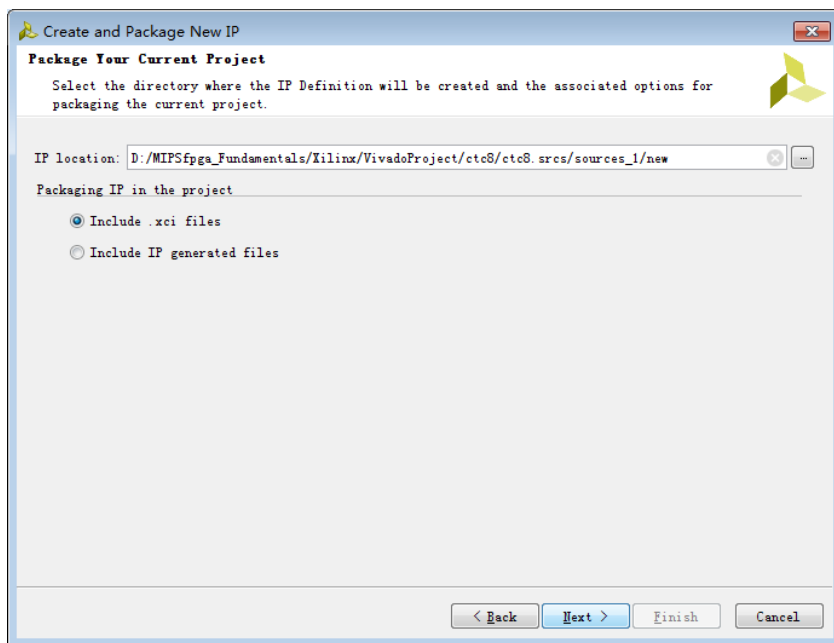


图 19 IP Location

在 IP Location（图 19）中我们不做修改，点击 **Next**。不过我们知道了封装后的 IP 放在了 D:/MIPSfpga_Fundamentals/Xilinx/VivadoProject/ctc8/ctc8.srcs/sources_1/new 这个文件夹中。

在图 20 所示的对话框中点击 **Finish**。

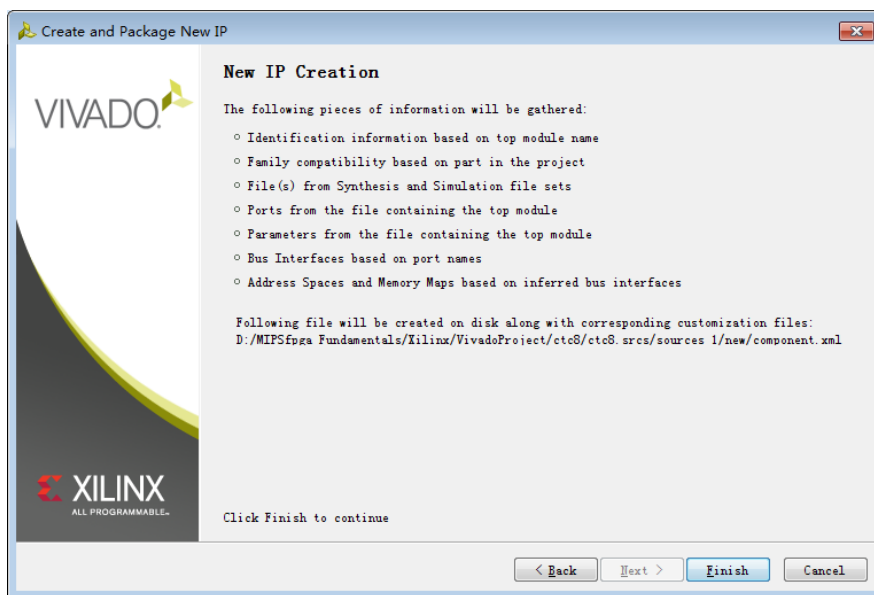


图 20 创建 IP 核结束

这是可以看到如图 21 所示的 IP 封装设置。按图 21 那样设置好各个项目。

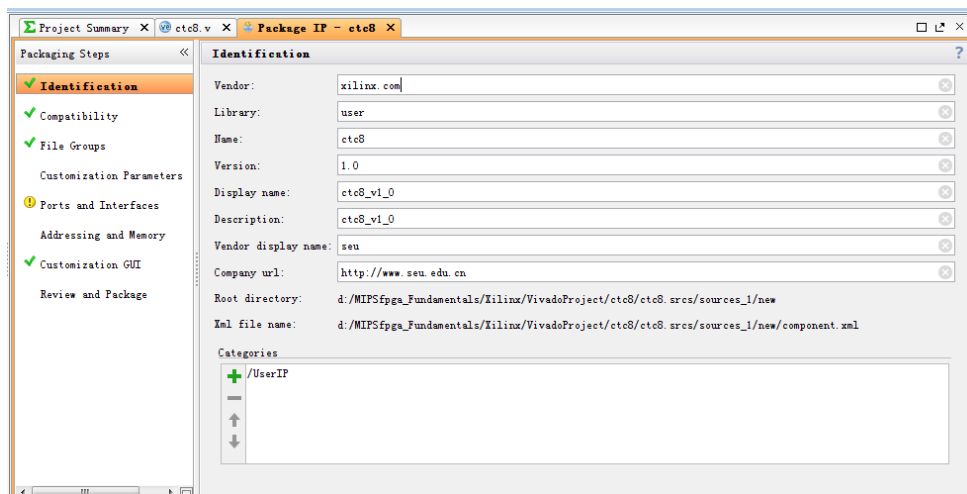



图 21 设置 Identification

在图 22 中我们可以添加 IP 核所支持的芯片家族。点击  并选择 Add Family Explicitly，在如图 23 所示的 Add Family 对话框。

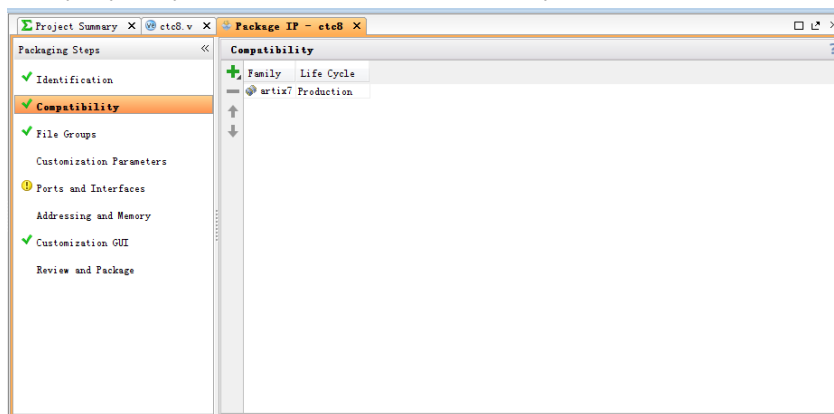


图 22 设置 compatibility

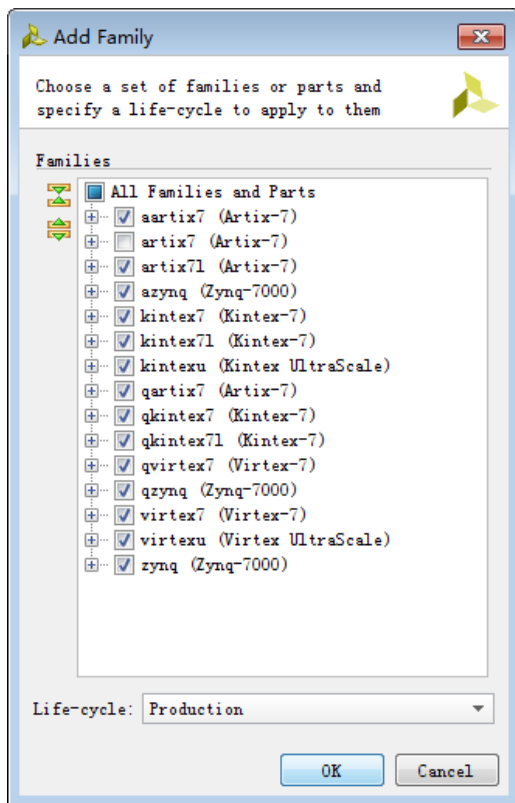


图 23 Add Family 对话框

在 Add Family 对话框中选中除 artix7 之外的所有芯片家族(因为 artix7 系列已经有了), 然后点击 **OK**。这样就设置完了 compatibility。
接下来我们到 Review and Packaging (如图 24 所示)。

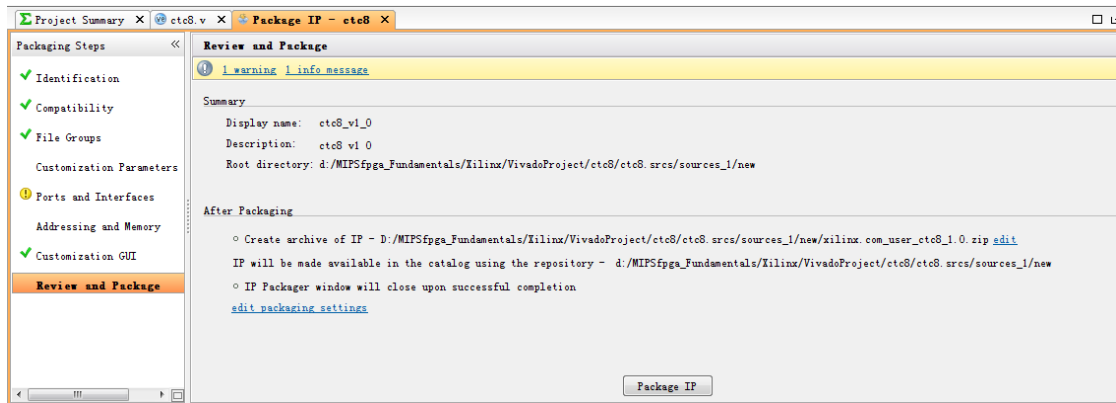


图 24 Review and Packaging

我们可以看到 IP 生成到一个成为 xilinx.com_user_ctc8_1.0.zip 文件中, 路径是 D:/MIPSFpga_Fundamentals/Xilinx/VivadoProject/ctc8/ctc8.srcs/sources_1/new。
点击 Package IP。这样, ctc8 的 IP 核就生成了。

3 创建跑马灯项目

3.1 创建一个新的项目

参考 2.1 的步骤，创建一个新的项目，注意，新项目名称为 `ledLights`。项目的位置依然是 `D:/MIPSfpga_Fundamentals/Xilinx/VivadoProject`，创建完后，最后，整个项目在 `D:/MIPSfpga_Fundamentals/Xilinx/VivadoProject/ledLights` 中。创建完后的界面如图 25 所示。

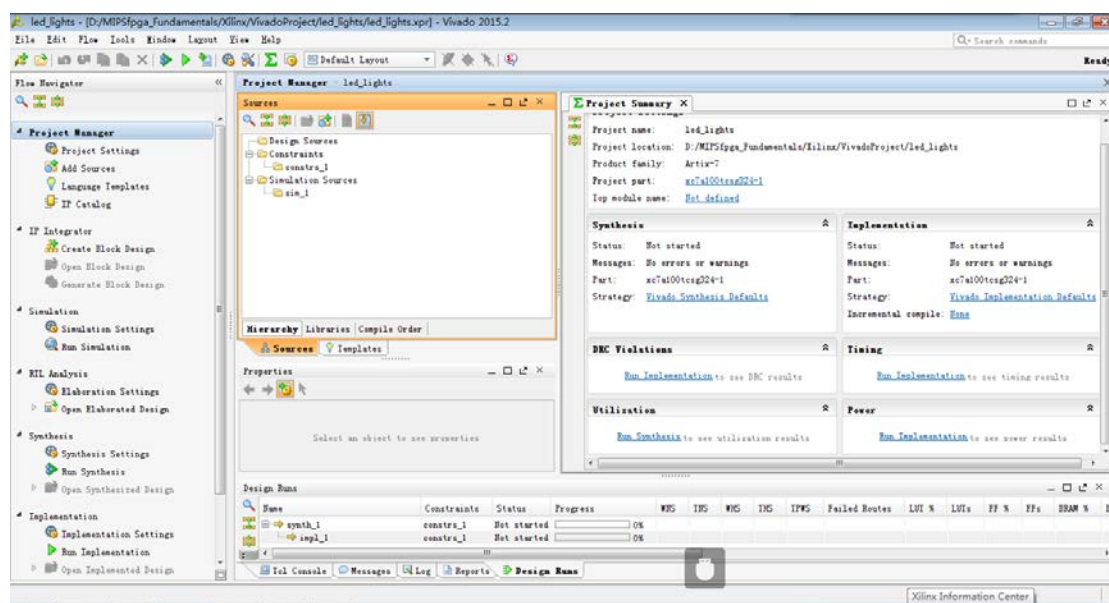


图 25 ledLights 项目创建后的情况

3.2 使用ctc8 IP核

3.2.1 导入ctc8 IP核到IP catalog

到 `D:/MIPSfpga_Fundamentals/Xilinx/VivadoProject/ctc8/ctc8.srcs/sources_1/new` 路径下将 `xilinx.com_user_ctc8_1.0.zip` 文件拷贝到下面的路径中。

`D:\MIPSfpga_Fundamentals\Xilinx\VivadoProject\ledLights`。

然后将其解压到 `xilinx.com_user_ctc8_1.0` 路径中，如图 26 所示。

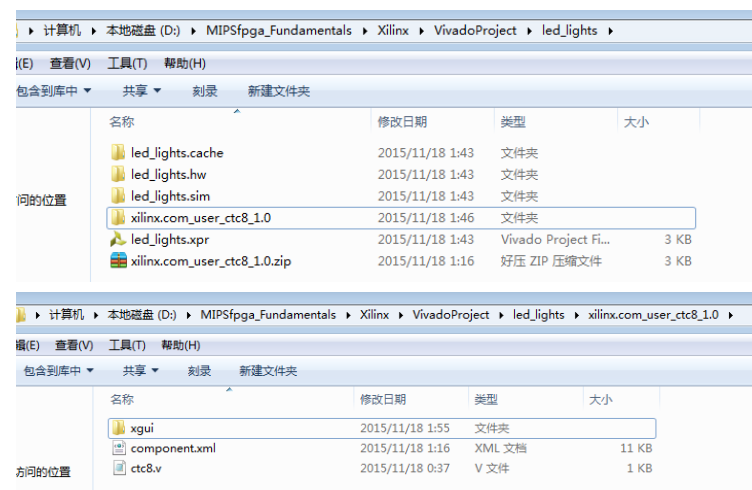


图 26 解压后的 IP 核

回到图 25 的界面，点击 Project Settings，打开 Project Settings 对话框，并转到 IP 项上。如图 26 所示。

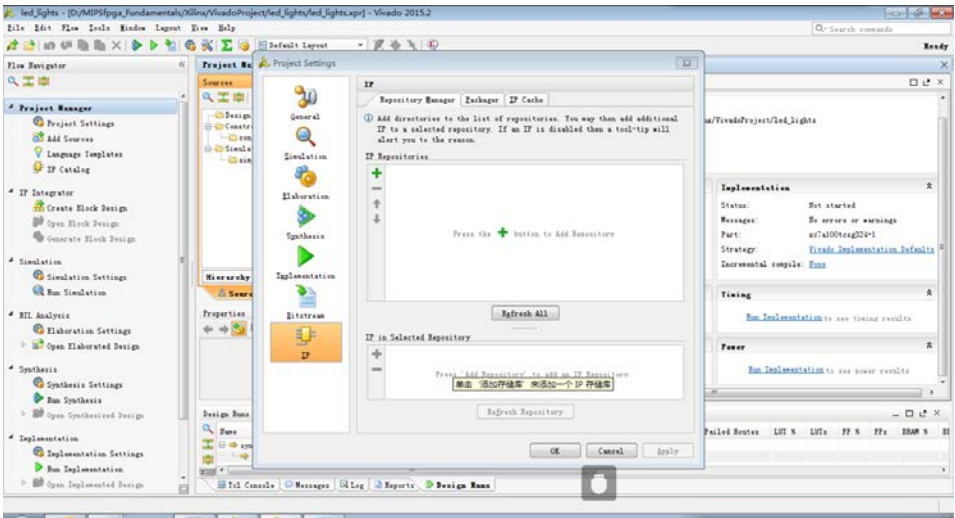


图 26 项目设置中添加 IP 核

点击 **+**，找到下面的目录：

D:/MIPSfpga_Fundamentals/Xilinx/VivadoProject/ledLights/xilinx.com_user_ctc8_1.0

如图 27 所示，点击 Select。

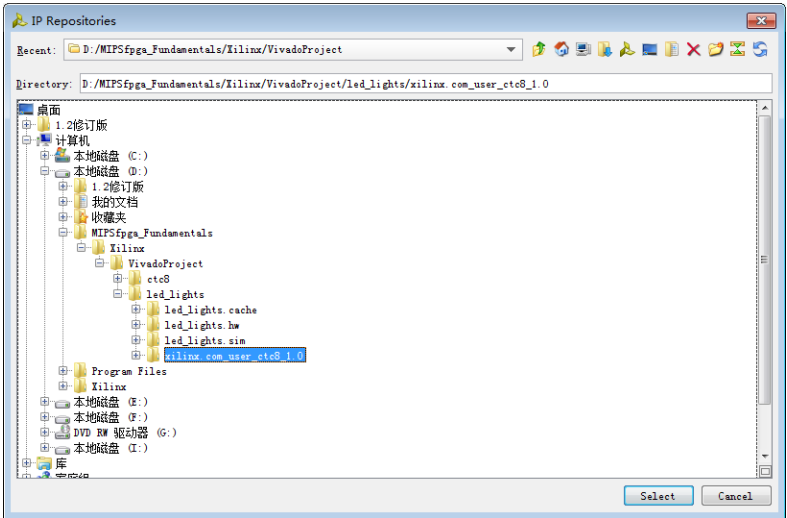


图 27 选择 ctc8 IP 核的位置

现在回到了如图 28 所示的 Project Settings 对话框。点击 Apply，然后点击 OK。

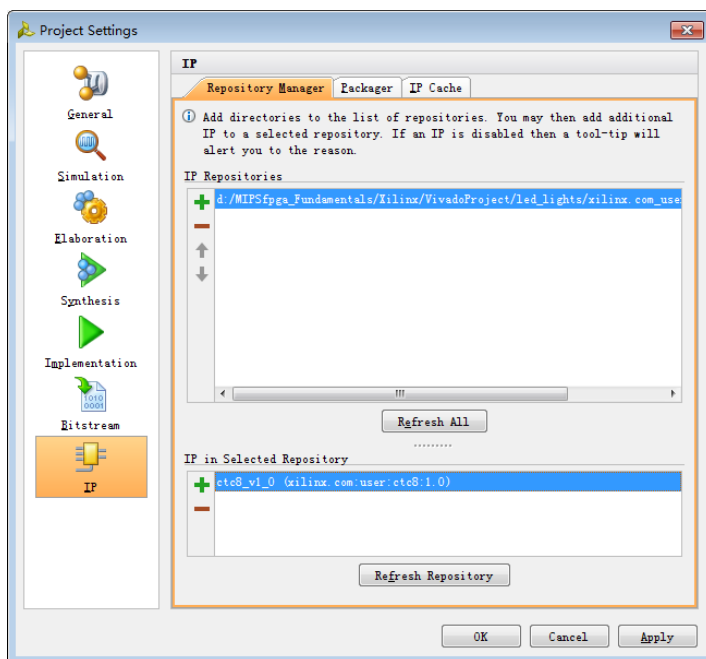


图 28 导入 ctc8 后的 Project Settings 对话框

现在点击 Project Manager 下的 IP Catalog， 我们就会看到如图 29 所示的界面中， IP Catalog 中已经有了我们的 ctc8。

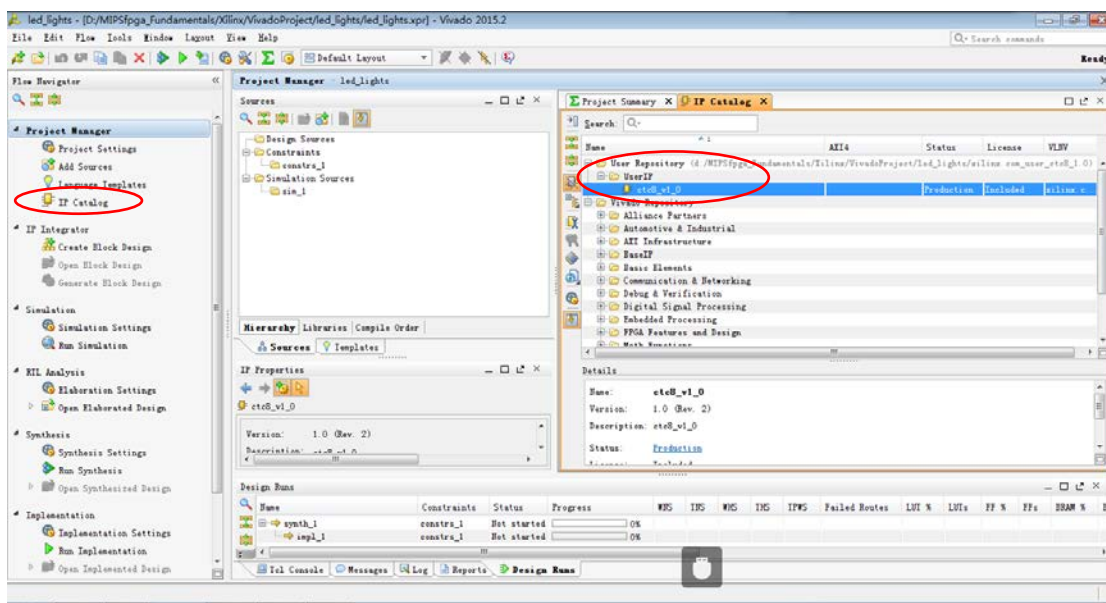


图 29 IP Catalog 中的 ctc8

3.2.2 将ctc8 IP核导入到项目中

在 3.2 中，我们已经将 ctc8 IP 核导入到了 IP Catalog 中，我们现在需要将它导入到我们的项目中。

在图 29 中，双击 ctc8_v1_0，得到图 30 所示的 ctc8_v1_0 的定制窗口。

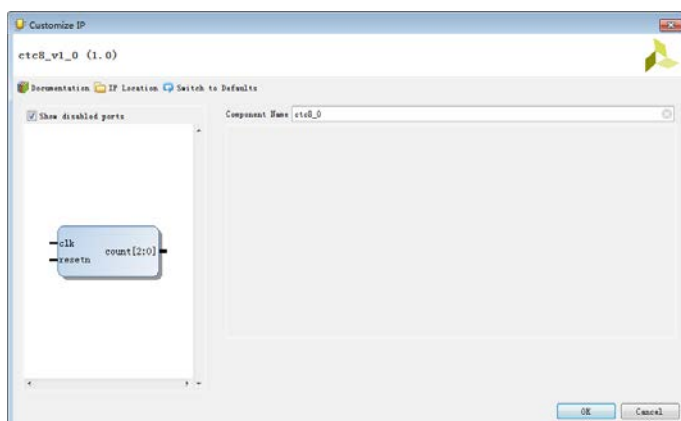


图 30 ctc8_v1_0 的定制窗口

点击 **OK**。弹出如图 31 所示的 Generate Output Products 窗口。点击 **Generate**。

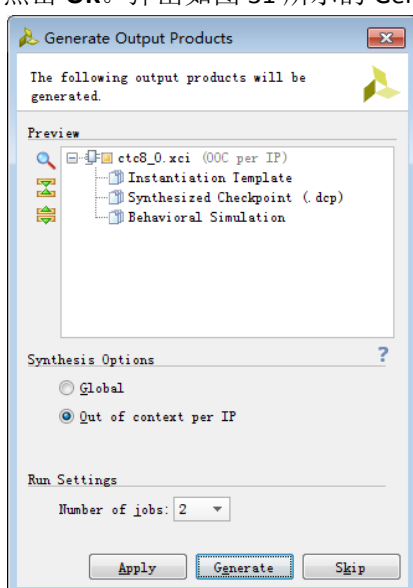


图 31 Generate Output Products 窗口

在随后弹出的对话框中点击 **OK**。此时我们在 Project Manager 的 Sources 中可以看到刚加进去的 ctc8_0，如图 32 所示。

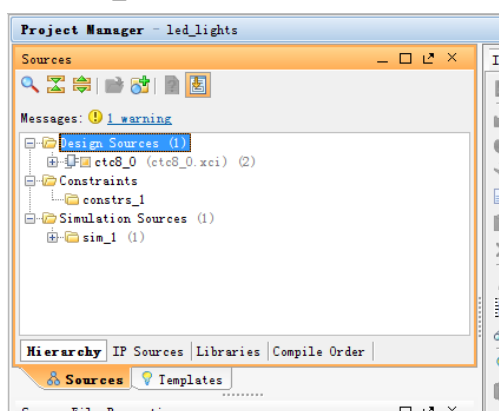


图 32 加紧项目中的 ctc8 IP 核。

双击 ctc8_0，会有图 33 所示的窗口弹出，点击 **OK**。就可以看到该 IP 核中的各种文件了。

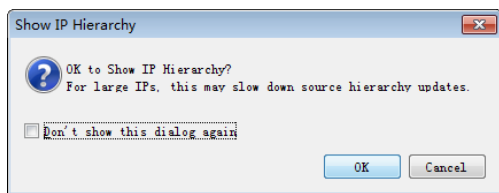



图 33 展开 IP 核的提问

3.3 添加分频器的源代码

Nexys 4 DDR 板上提供了一个 100MHz 的时钟，但我们的跑马灯是 1 秒一变换，因此需要一个分频器，将 100MHz 降到 1Hz。

右键点击 Design Sources，在弹出的菜单中选择 Add Sources....，在出现的对话框中按照图 7 所示选择后点击 Next。在接下来打开的 Add Sources 对话框（如图 8 所示）中点击 , 并选择 Create File...。此时会看到 Create Source File 对话框，按照图 34 所示填写后点击 OK。

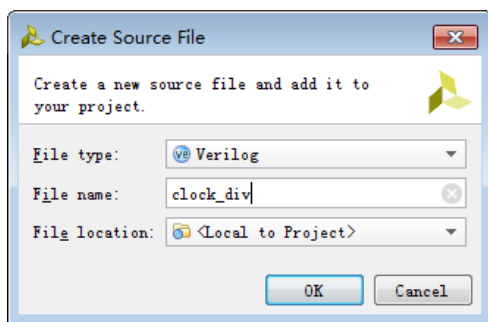


图 34 文件名为 clock_div

现在回到了 Add Sources 对话框，点击 Finish。在弹出的 Define Module 对话框中选择 OK。这时会弹出图 12 所示的对话框，点击 Yes。

在 sources 窗口中双击 clock_div，打开该文件。该文件的模块如下：

```
module clock_div(

    );
endmodule
用下面的代码替换它。
module clock_div(
    clk,                //100MHz
    clk_sys              //1Hz
    );
    input clk;
    output clk_sys;

    reg clk_sys = 0;
    reg [25:0] div_counter = 0;

    always @(posedge clk) begin
        if(div_counter>=50000000) begin
```

```
        clk_sys<=~clk_sys;
        div_counter<=0;
    end else begin
        div_counter <= div_counter + 1;
    end
end
endmodule
```

这里实现了一个加 1 计数器，每当 100MHz 的时钟上升沿，计数器加 1，每次计数器到 50000 的时候输出电平翻转，这样输出的波形的周期刚好 1 秒，频率为 1Hz。

3.4 添加简易 38 译码器的源代码

如同 2.2 和 3.3 中的步骤那样，添加一个新的 Verilog 文件，命名为 s_38。

双击打开后，模块为：

```
module s_38(
```

```
    );
```

```
endmodule
```

用下列代码替换该模块。

```
module s_38(
```

```
    Y0,
```

```
    Y1,
```

```
    Y2,
```

```
    Y3,
```

```
    Y4,
```

```
    Y5,
```

```
    Y6,
```

```
    Y7,
```

```
    A,
```

```
    B,
```

```
    C,
```

```
    Enable
```

```
);
```

```
input  A,B,C,Enable;
```

```
output reg Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7;
```

```
always @(A or B or C or Enable) begin
```

```
    if(!Enable)
```

```
        {Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0} <= 8'b0000_0000;
```

```
    else begin
```

```
        case ({C,B,A})
```

```
            3'b000 : {Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0} <= 8'b0000_0001;
```

```
            3'b001 : {Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0} <= 8'b0000_0010;
```

```
            3'b010 : {Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0} <= 8'b0000_0100;
```

```

        3'b011 : {Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0} <= 8'b0000_1000;
        3'b100 : {Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0} <= 8'b0001_0000;
        3'b101 : {Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0} <= 8'b0010_0000;
        3'b110 : {Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0} <= 8'b0100_0000;
        3'b111 : {Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0} <= 8'b1000_0000;
        default: {Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0} <= 8'b0000_0000;
    endcase
end
end
endmodule

```

这是一个简单的 3-8 译码器，把三个使能端简化为一个使能端 **Enable**。

3.5 创建顶层文件

如同 2.2 和 3.3 中的步骤那样，添加一个新的 Verilog 文件，命名为 **led_lights**。

双击打开后，模块为：

```
module led_lights(
```

```
    );
```

```
endmodule
```

用下列代码替换该模块。

```
module led_lights(
```

```
    clock,
```

```
    resetn,
```

```
    y0,
```

```
    y1,
```

```
    y2,
```

```
    y3,
```

```
    y4,
```

```
    y5,
```

```
    y6,
```

```
    y7,
```

```
);
```

```
input clock,resetn;
```

```
output y0,y1,y2,y3,y4,y5,y6,y7;
```

```
wire clk_sys;
```

```
wire [2:0] count;
```

```
// clock
```

```
clock_div U0(
```

```
    .clk(clock),
```

```
    .clk_sys(clk_sys)
```

```
);
```

```
// 38decoder
s_38  U1(
    .A(count[0]),
    .B(count[1]),
    .C(count[2]),
    .Enable(resetn),
    .Y0(y0),
    .Y1(y1),
    .Y2(y2),
    .Y3(y3),
    .Y4(y4),
    .Y5(y5),
    .Y6(y6),
    .Y7(y7)
);

// ctc
ctc8_0  U2(
    .clk(clk_sys),
    .resetn(resetn),
    .count(count)
);
endmodule
```

可以看到，顶层文件就是对三个模块进行了例化。并用 Verilog 语言将三个模块用线连接了起来。时钟和 **resetn** 控制计数器工作，**resetn** 还同时作为 3-8 密码器的使能端。计数器的三个输出分别接到 3-8 译码器的 CBA 输入上，随着计数值的变化，3-8 译码器输出发生变化。顶层文件还定义了对外的接口，两个输入 **clock**，**resetn** 分别接板上时钟信号和复位信号，Y0-Y7 的 8 个输出分别连接板上的 LED0-LED7。

到此，所有的模块都建好了，把所有的源程序存盘后，在 Project Manager 的 Sources 中我们可以看到整个项目的层次结构如图 35 所示。所有三个模块都在顶层文件之下。

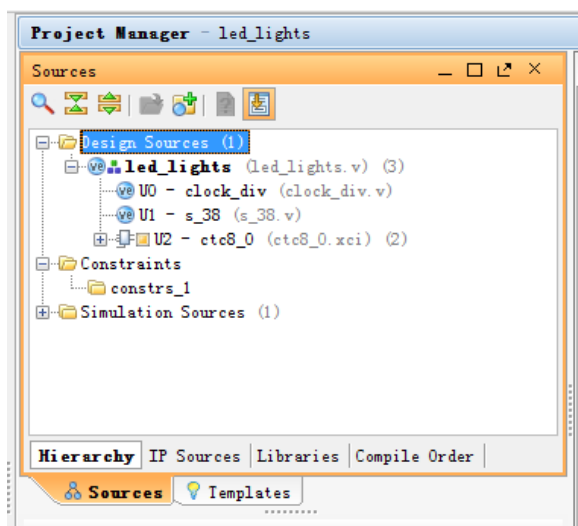



图 35 项目的文件层次结构

4 仿真

在正式综合和下载之前，要先对设计进行仿真，以检查电路设计是否正确。右键点击 Project Manager 下面 Sources 中的 Simulation Sources，在弹出的菜单中选择 Add Source…。按照图 36 所示的设置点击 Next。



图 36 添加仿真源程序

在弹出的如图 37 的窗口中点击 ，并选择并选择 Create File…。

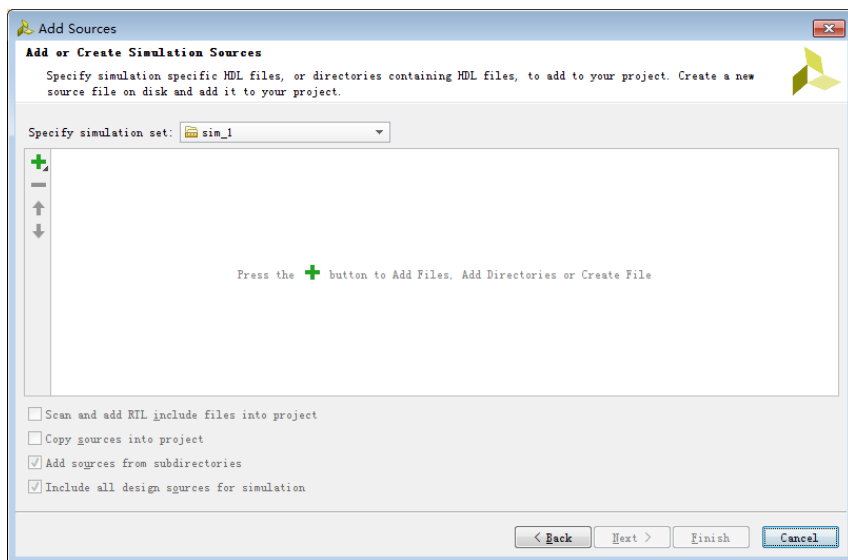


图 37 增加仿真源程序第二步-、Add Sources 对话框

在创建文件的窗口如图 38 那样设置仿真源文件的文件名为 led_sim，并点击 OK。

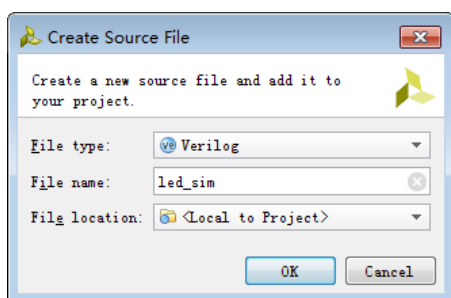


图 38 设置仿真源文件文件名

现在回到了 Add Sources 对话框，点击 OK。在接下来的 Define Module 窗口中点击 OK，接下来弹出的窗口（如图 12）点击 Yes。

如图 39 所示，现在我们在项目中看到了这个仿真源文件（图中高亮部分）。

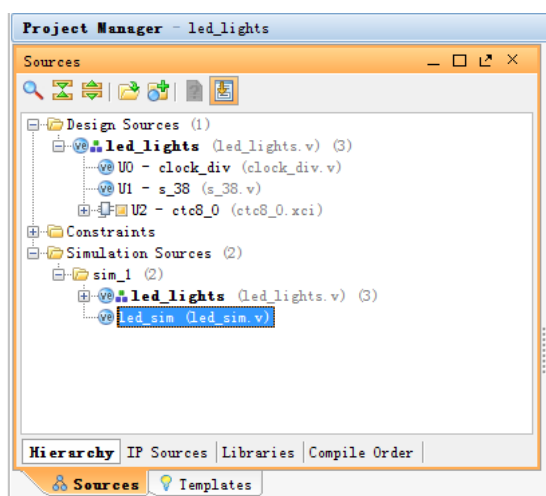


图 39 项目中的仿真源文件

双击图 39 中的高亮部分，打开该文件。可以看到目前的模块定义为：

```
module led_sim(
```

```
    );  
Endmodule  
用下面的代码替代。  
module led_sim(    );  
    //input  
    reg clock = 0;  
    reg resetn = 0;  
  
    //output  
    wire y0,y1,y2,y3,y4,y5,y6,y7;  
  
    //instantiate the Unit under test  
    led_lights uut(  
        .clock(clock),  
        .resetn(resetn),  
        .y0(y0),  
        .y1(y1),  
        .y2(y2),  
        .y3(y3),  
        .y4(y4),  
        .y5(y5),  
        .y6(y6),  
        .y7(y7)  
    );  
  
    initial begin  
        #500 resetn = 1;  
    end  
  
    always #5 clock = ~clock;  
endmodule
```

上面的代码我们首先例化了 `led_light` 模块，对 `clock` 和 `resetn` 均初始化为 0。之后，将 `resetn` 设置为 1。`always #5 clock = ~clock;` 这句形成 100MHz 的时钟信号。保存好 `led_sim.v` 文件后，我们看到的项目文件层次如图 40 所示。

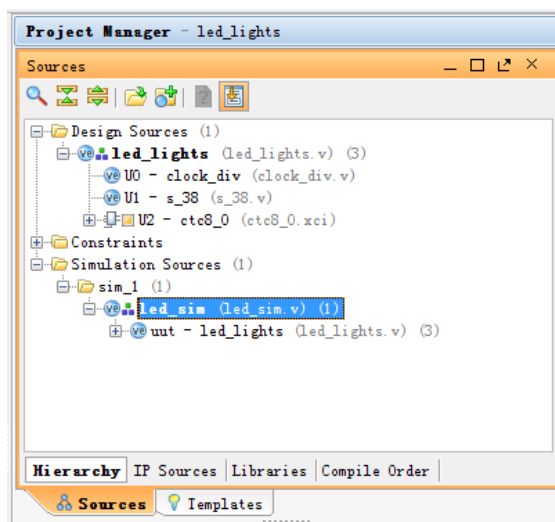


图 40 编辑完 led_sim.v 后的项目文件层次

为了仿真能快一些，我们需要修改一下 `clock.v` 的代码，将下面一句

```
if(div_counter>=50000000) begin
```

改为

```
if(div_counter>=50) begin
```

这样，我们用 1us 来仿真 1 秒。

在如图 41 所示的 Project Manager 中点击 Run Simulation，在弹出的菜单中选择 Run behavior Simulation。

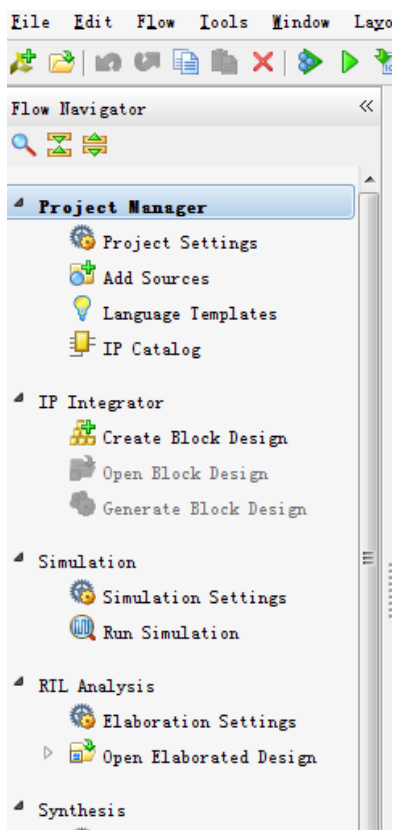



图 41 点击 Run Simulation

将工具条中的仿真时间调整为

, 点击, 然后点击

开始仿真。仿真完后，点击得到如图 42 所示的波形图。

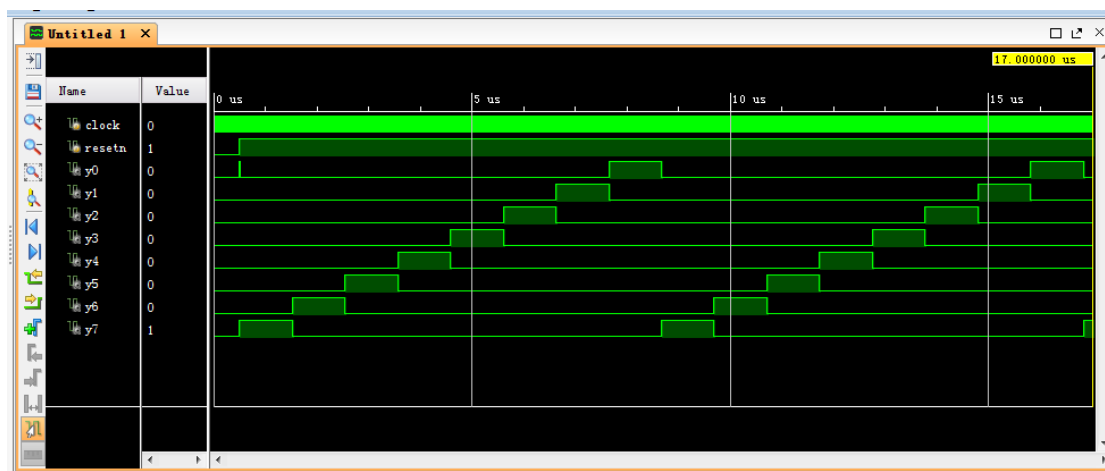


图 42 仿真波形图

从图上我们可以明显地看到，0.5us 后 resetn 信号无效，计数器和 3-8 译码器开始工作，每隔 1us，3-8 译码器在计数器的推动下，换一个输出，Y7-Y0 依次输出，周而复始。


5 综合与实现

首先，我们要把 clock.v 的代码修改回来，将下面一句

```
if(div_counter>=50) begin
```

改为

```
if(div_counter>=50000000) begin
```

接下来，在 Project Manager 中点击 Run Synthesis 或者工具条上的按钮。综合如果没有问题，会弹出如图 43 所示的窗口。

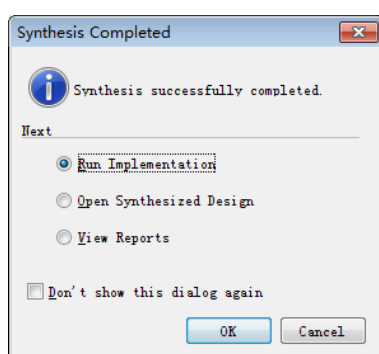



图 43 综合以后的窗口

按照图 43 的选择，点击 OK，对设计进行实现。也可以点击工具条中的按钮或者在 Project Manager 中点击 Run Implementation 来对设计进行实现。

实现完后出现图 44 的窗口。点击 Cancel。

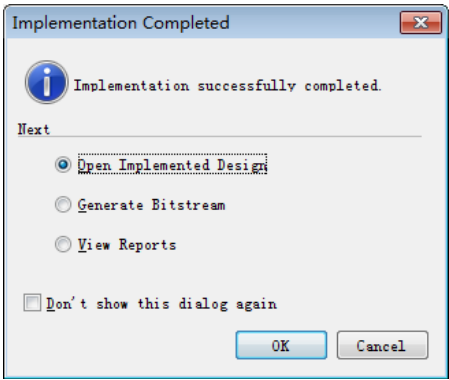


图 44 实现后的窗口

这样我们就完成了综合与实现。

6 管脚分配

在 Project Manager 中点击 Open Synthesized Design。有一个滚动条说明进度。滚动条结束后，选择菜单 Layout->I/O Planning。出现如图 45 所示的管脚分配表。

Name	Direction	Neg Diff Pair	Site	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	SL
All ports (10)										
resetsn_34238 (1)	IN					default (LVCMOS18)	1.800			
Scalar ports (1)										
resetsn	IN					default (LVCMOS18)	1.800			
Scalar ports (9)										
clock	IN					default (LVCMOS18)	1.800			
y0	OUT					default (LVCMOS18)	1.800	12		SL
y1	OUT					default (LVCMOS18)	1.800	12		SL
y2	OUT					default (LVCMOS18)	1.800	12		SL
y3	OUT					default (LVCMOS18)	1.800	12		SL
y4	OUT					default (LVCMOS18)	1.800	12		SL

图 45 管脚分配表

现将 I/O Std 这一列全部改成 LVCMOS33。然后根据下表，点击 site 对每个管脚进行分配。如 resetsn 对应的是 C12，clock 对应的是 E3 等。

表 1 管脚分配表

信号	管脚
clock	E3
resetsn	C12
y0	H17
y1	K15
y2	J13
y3	N14
y4	R18
y5	V17
y6	U17
y7	U16

管脚分配好后，如图 46 所示。

Name	Direction	Mag Diff Pair	Site	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination
reseta_34238 (1)	IN			<input checked="" type="checkbox"/>		15 LVCMOS33*		3.300			NONE	NONE
Scalar ports (1)												
reseta	IN		C12	<input checked="" type="checkbox"/>		15 LVCMOS33*		3.300			NONE	NONE
Scalar ports (9)												
clock	IN		E3	<input checked="" type="checkbox"/>		35 LVCMOS33*		3.300			NONE	NONE
y0	OUT		H17	<input checked="" type="checkbox"/>		15 LVCMOS33*		3.300	12	SLOW	NONE	FP_VII_50
y1	OUT		K15	<input checked="" type="checkbox"/>		15 LVCMOS33*		3.300	12	SLOW	NONE	FP_VII_50
y2	OUT		J13	<input checked="" type="checkbox"/>		15 LVCMOS33*		3.300	12	SLOW	NONE	FP_VII_50
y3	OUT		H14	<input checked="" type="checkbox"/>		14 LVCMOS33*		3.300	12	SLOW	NONE	FP_VII_50
y4	OUT		R18	<input checked="" type="checkbox"/>		14 LVCMOS33*		3.300	12	SLOW	NONE	FP_VII_50
y5	OUT		V17	<input checked="" type="checkbox"/>		14 LVCMOS33*		3.300	12	SLOW	NONE	FP_VII_50
y6	OUT		U17	<input checked="" type="checkbox"/>		14 LVCMOS33*		3.300	12	SLOW	NONE	FP_VII_50
y7	OUT		U16	<input checked="" type="checkbox"/>		14 LVCMOS33*		3.300	12	SLOW	NONE	FP_VII_50

图 46 管脚分配后

按 Ctrl+S，出现如图 46 所示的窗口，点击 OK。

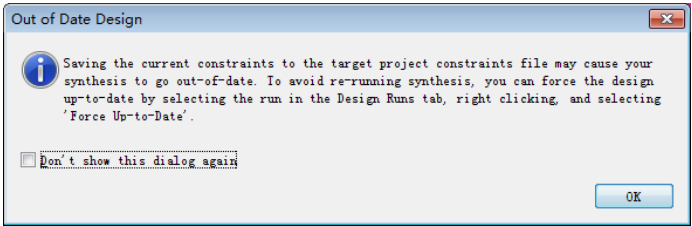


图 46 保存管教分配设置

在出现的如图 47 所示的窗口中填写约束文件名为 led_lights。

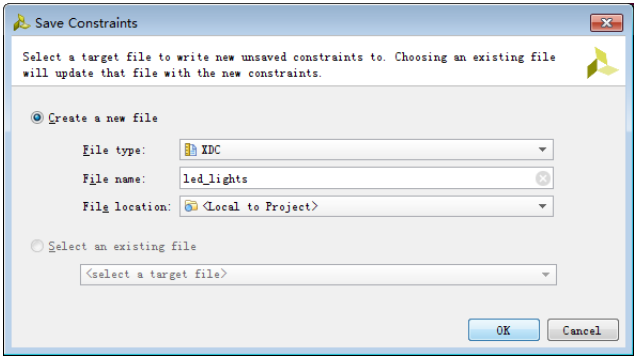


图 47 命名约束文件

点击 OK。

7 产生比特流文件并下载

在 Project Manager 中点击 Generate Bitstream 或者工具条上的 按钮。。比特流生成后会出现图 48 所示的对话框

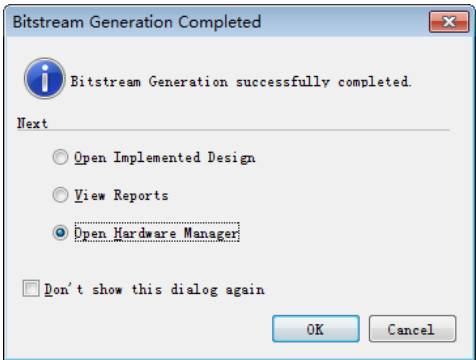


图 48 比特流生成完成

用 USB 下载线将 Nexys4 DDR 板的 PROG UART 与 PC 机的 USB 相连，打开 Nexys4 DDR 板的电源。按照图 48 所示选中 Open Hardware Manager。点击 **OK**。（在 Project Manager 中点击 Hardware Manager）。出现图 49 所示的界面。

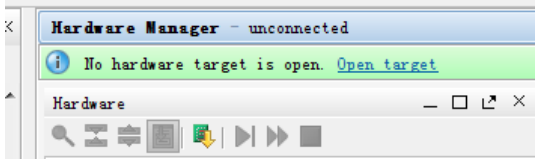


图 49 Hardware Manager

点击 Open target->Open new target。在弹出的窗口中点击 Next，再次点击 Next。此时出现图 50 所示的滚动条。

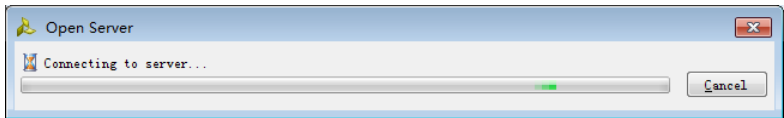


图 50 寻找硬件

硬件连接上后，会出现图 51 所示界面。

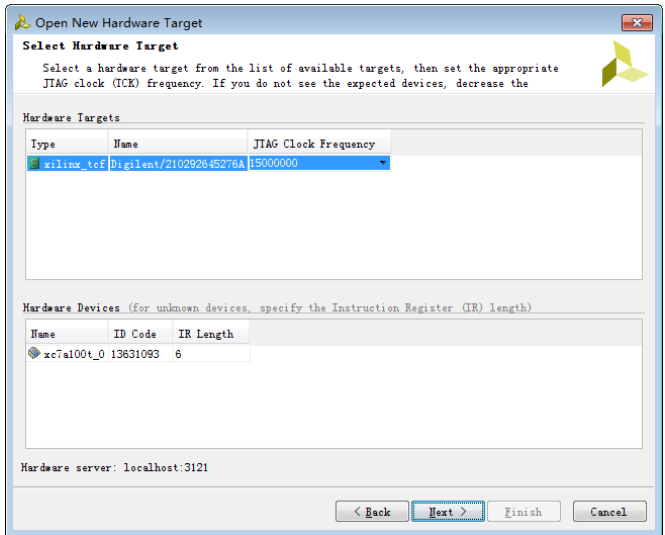


图 51 找到硬件

所列的正是我们接到 PC 上的板子的主芯片。点击 Next，然后点击 Finish。出现如图 52 所示界面。

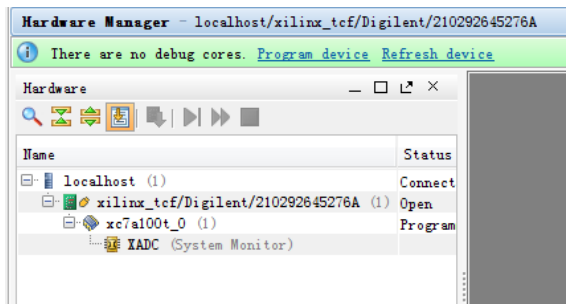


图 52 连接上硬件后

点击 Program devices->xc7a100t_0。出现的 Program Device 窗口(如图 53)中点击 **Program**。

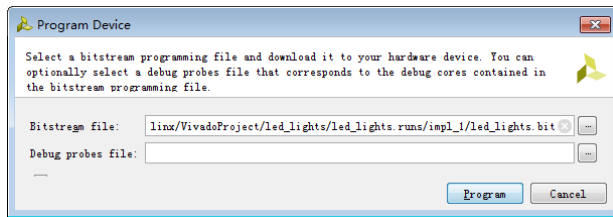
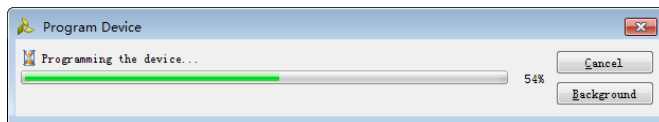


图 53 Program Device 窗口

出现如图 54 所示的正在下载的进度条。



下载结束后，会看到 Nexys4 DDR 板上的 LED 灯每秒变换一次，从左向右依次点亮。