9 PDS 软件中时钟管理单元 PLL 配置和使用

工程源码	
相关视频课程	

章节导读

PLL（Phase Locked Loop，即锁相环）是最常用的 IP 核之一，其性能强大，可以对输入到 FPGA 的时钟信号进行任意分频、倍频、相位调整、占空比调整，从而输出期望时钟。

实验任务

本章中我们将调用配置 PDS 软件 提供的 PLL IP 核并对其进行仿真以了解其工作时序，然后介绍了参数化设计方法。本章的实验具体内容是：在 PGL22G 开发板上，使用 PLL 分别生成一个 25M、一个 75M、100M 的时钟，使用生成的三个时钟以及输入时钟分别驱动一个 LED 闪烁模块，控制 LED 的亮灭。通过观察四个 LED 灯在完全相同的驱动模块的驱动下，不同驱动时钟对其闪烁速度的影响，从而验证锁相环对时钟的倍频和分频处理的正确性。

9.1 工程建立

我们首先创建一个新工程，工程名为“PLL_LED”。接下来我们要为演示如何调用 PLL IP 核来产生不同的时钟。下面为程序设计的详细步骤。

依次点击菜单栏下的“Tools”下的“IP Compiler”,如图 9-1 所示。

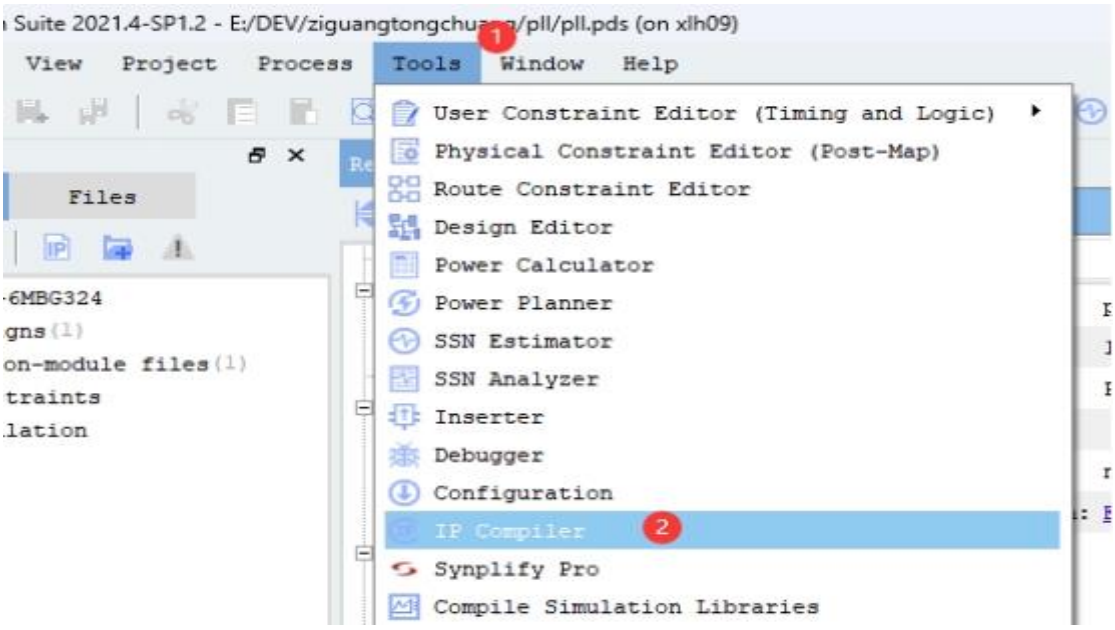


图 9-1“IP Compiler”页面

点击图 9-1 中的“IP Compiler”后会跳转图 9-2 页面。第一步，我们点击标签 1 中的“PLL”按钮，选择创建 PLL IP 核；第二步，将标签 2 中“Instance Name”选项命名为“PLL”；第三步，点击标签 3 中的“Customize”按钮进入 PLL IP 参数配置页面。

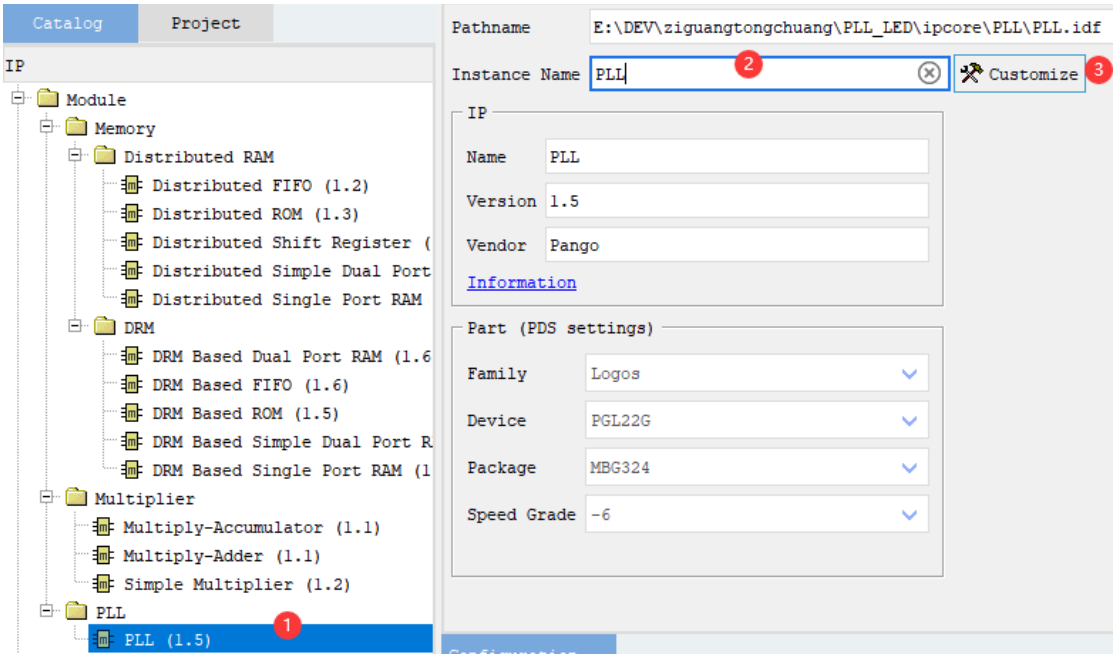


图 9-2IP 选择页面

点击图 9-2 中的“Customize”按钮进入图 9-3 中。

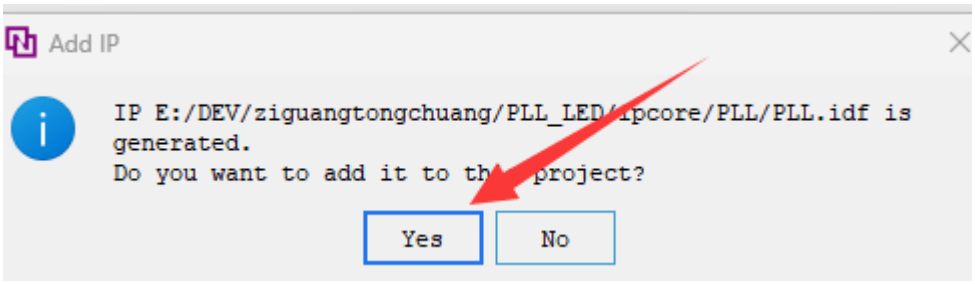


图 9-3 “Add IP 弹窗”

图 9-3 中表示的是 PLL.idf 文件已经生成，询问我们是否将该文件加载到我们的工程中。因为这个 IP 我们后续是要在工程中用到的，因此直接点击图 9-3 中箭头指向的“Yes”按钮进入下一页面。 在图 9-4 中标签 1“Mode Selection”（模式配置）有两个选项：一个是“Basic Configurations”（基础配置模式），一个是“Advanced Configuration”（高级配置模式）。由于本章实验仅仅是简单的 PLL IP 的使用，我们选择“Basic Configurations”（基础配置模式）即可。

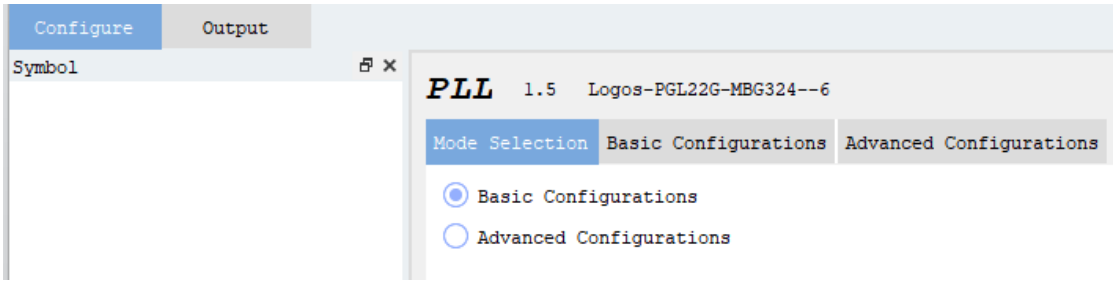


图 9-4 IP 模式配置窗口

然后点击“Basic Configurations”选项，进入 Basic 模式界面。Basic 模式界面主要包括输入配置（Input Configurations）、5 个输出时钟配置（Clkout0~4 Configurations）与显示 PLL 内部设置参数（Show Internal Settings of PLL），Basic Configurations 选项卡界面如下图所示:

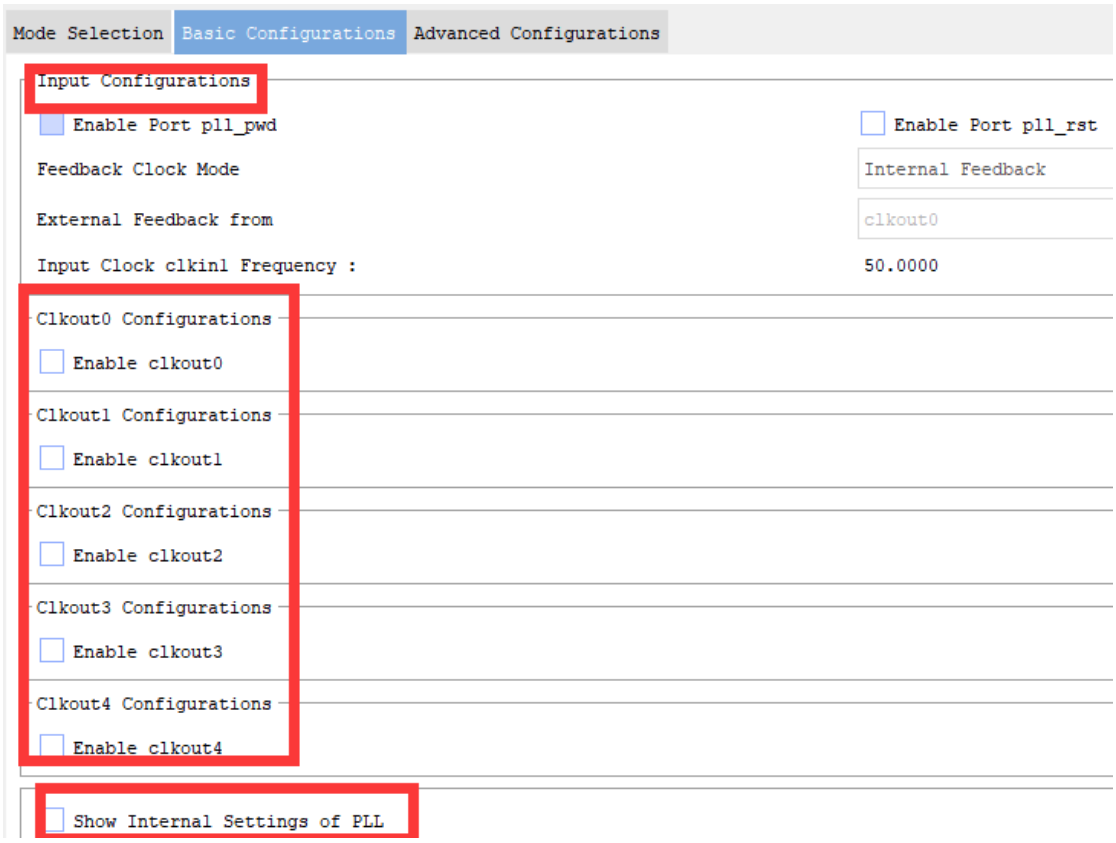


图 9-5 Basic 模式界面

把图 9-6 中的“Input Configurations”选项中的 “Input Clock clkin1 Frequency”（输入时钟频率）设置为我们开发板上的晶振频率 50MHz。勾选“Enable Port pll_rst”复位使能信号，时钟 IP 核默认是高电平有效复位的。输入配置框其他的设置保持默认即可。

图 9-6“Input Configurations”选项

在图 9-7 中的“Clkout0 Configurations”选项卡中勾选箭头指向的“Enable clkout0”选项。其中，标签 1 中的“Desired Frequency”表示的是希望得到的输出频率，我们这里在图 1.7 中填写数据 25，即希望输出频率为 25Mhz。标签 2 中的“Desired Phase Shift”表示的是输出时钟的相位偏移度，我们这里默认设置，即希望输出的相位偏移为 0 度。标签 3 中的“Desired Duty Cycle”表示的是输出时钟的占空比，我们这里填写的是 50，即希望输出时钟的占空比为 50%，其它

设置保持默认即可。

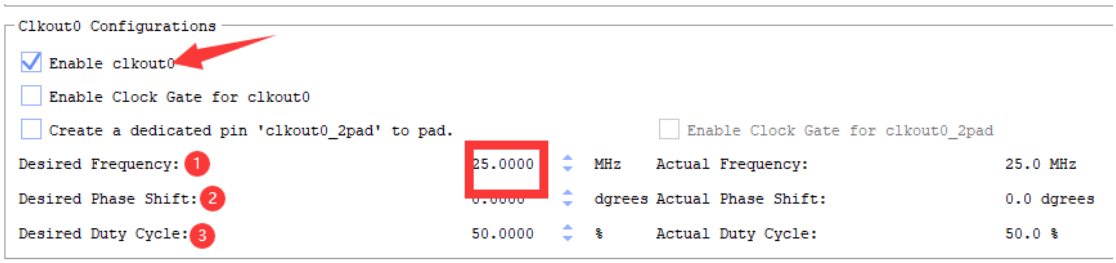


图 9-7“Clkout0 Configurations”选项卡配置

我们接着在图 9-8 中的“Clkout1 Configurations”进行如下配置。图 9-8 表示的是得到一个输出时钟信号，它的输出频率为 75Mhz，相位偏移为 0 度，输出时钟的占空比为 50%。

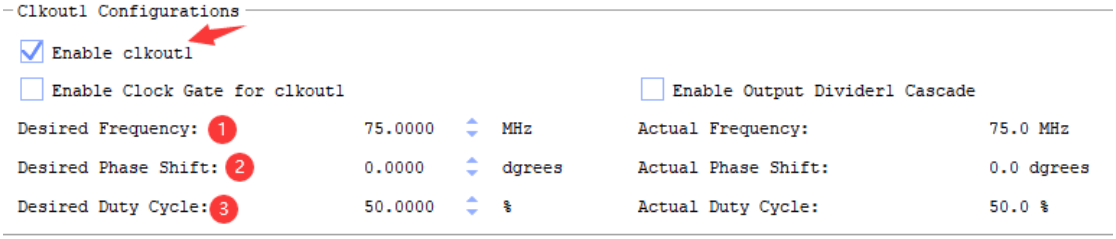


图 9-8“Clkout1 Configurations”选项卡配置

按照上述方式配置方式再配置一个输出频率为 100Mhz，相位偏移为 0 度，输出时钟的占空比为 50%的时钟信号。

最后，我们要勾选如图 9-9 中箭头指向的“Show Internal Settings of PLL”(PLL 内部设置参数)，而“Internal Setting of PLL”框里面的信息是 PLL 的配置参数。

图 9-9 PLL 内部设置参数

接下来我们点击图 9-10 中箭头指向的“Generate”按钮。

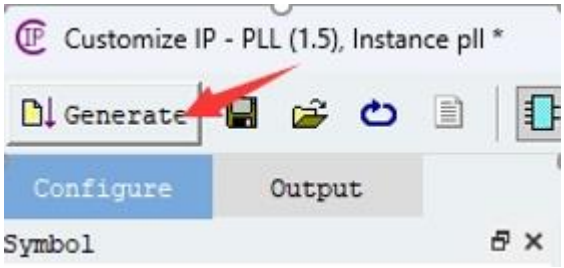


图 9-10“Generate”页面

之后会弹出图 9-10 界面，图 9-11 表示的是：IP 核修改的数据将会保存，是否继续操作。我们点击图 9-11 界面中箭头指向的“OK”按钮。

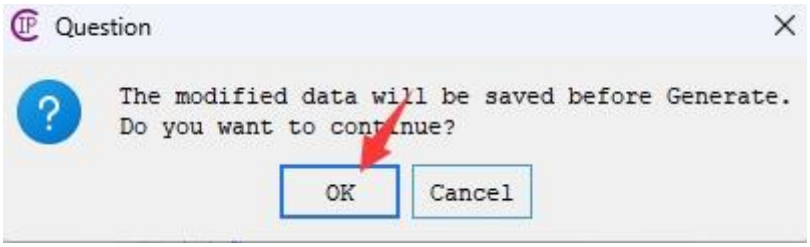


图 9-11“Question”页面

最终会弹出图 9-12 页面，至此 PLL IP 核配置成功。

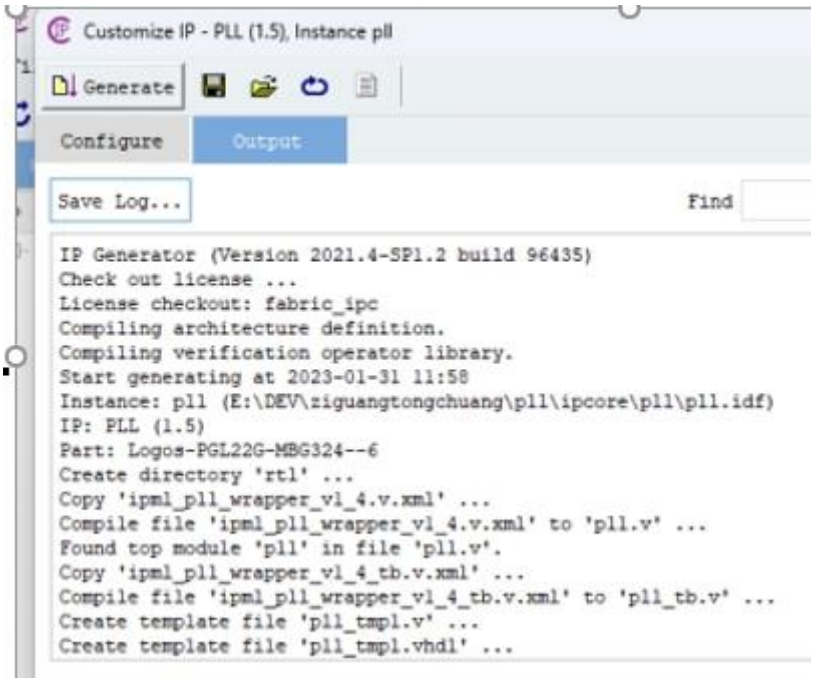


图 9-12 PLLIP 配置成功

IP 核配置成功后会自动弹出图 9-13 即这个 IP 的例化模板文件。

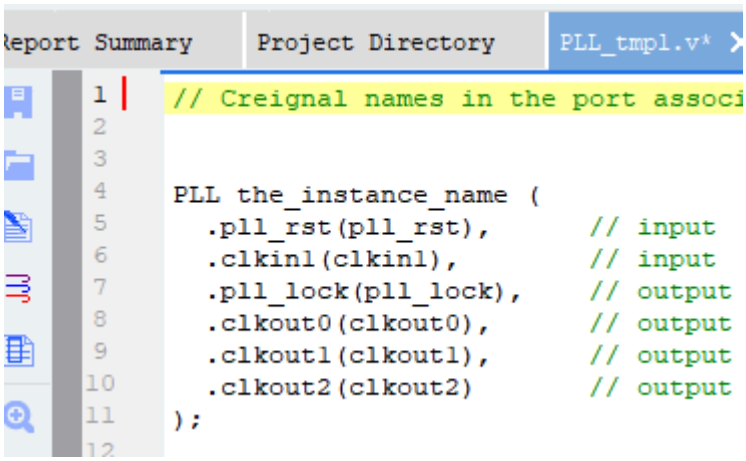


图 9-13 例化的模板文件

将 IP 核配置过程中弹出的页面关闭，返回 Source 面板，Source 面板中内容如图 9-14 所示。

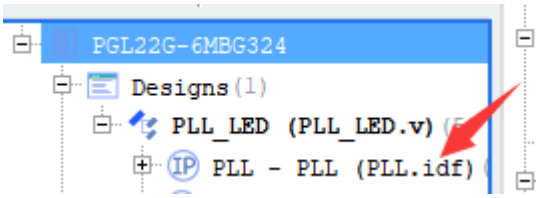


图 9-14 Source 面板

点击图 9-14 中 PLL IP 左侧加号位置，可以看到 PLL IP 核下有个 PLL.v 文件，点击图 9-15 箭头所指的位置，打开 PLL.v 文件。

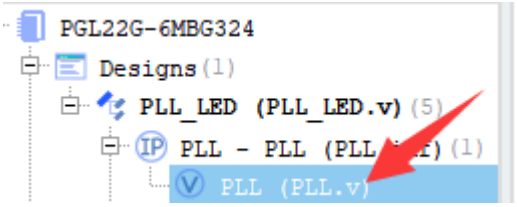


图 9-15 IP 核包含的文件

部分 PLL.v 文件内容如图 9-16 所示。

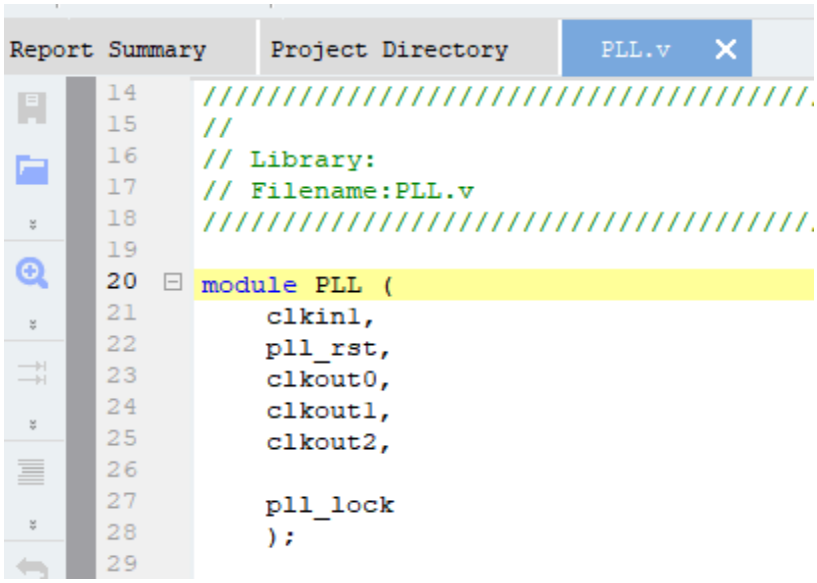


图 9-16 PLL.v 文件内容

若要调用 PLLIP，我们可以选择将图 9-13 中的端口例化到需要调用 PLLIP 的模块中，或者是直接例化图 9-16 中内容。

9.1.1 代码设计

接下来我们创建一个 verilog 源文件，其名称为 PLL_LED，代码如下：

```
module PLL_LED(
  clk,
  reset_n,
  LED
);

  input clk;
  input reset_n;
  output [3:0]LED;

  wire c0;  //25M
  wire c1;  //75M
  wire c2;  //100M
  wire locked;

pll pll (
  .pll_rst(~reset__n),      // input
  .clkin1(clk),        // input
  .pll_lock(locked),    // output
  .clkout0(c0),      // output
  .clkout1(c1),      // output
  .clkout2(c2)       // output
);
```

```
 counter
  #(
    .CNT_MAX(25'd24_999_999)
  )
  counter0(
    .clk(c0),
    .reset_n(reset_n),
    .led(LED[0])
  );

  counter
  #(
    .CNT_MAX(25'd24_999_999)
  )
  counter1(
    .clk(c1),
    .reset_n(reset_n),
    .led(LED[1])
  );

  counter
  #(
    .CNT_MAX(25'd24_999_999)
  )
  counter2(
    .clk(c2),
    .reset_n(reset_sn),
    .led(LED[2])
  );

  counter
  #(
    .CNT_MAX(25'd24_999_999)
  )
  counter3(
    .clk(clk),
    .reset_n(reset_n),
    .led(LED[3])
  );

endmodule
```

在 PLL_LED.v 文件中例化了 PLL IP 核，把 FPGA 的系统时钟 50Mhz 连接到 PLL IP 核的 clkin1，系统复位信号连接到 PLL IP 核的 pll_rst，由于时钟 IP 核默认是高电平复位，而输入的系统复位信号 reset_n 是低电平复位，因此要对系统复位信号进行取反。在 PLL_LED.v 文件中还例化了四次 counter 模块，

counter 模块代码如下。

```
module counter(
      clk,
      reset_n,
      led
    );

  input clk;  //系统时钟
  input reset_n;  //全局复位，低电平复位

  output reg led; //led 输出

  reg [24:0]cnt;  //定义计数器寄存器

  parameter CNT_MAX = 25'd24_999_999;

//计数器计数进程
  always@(posedge clk or negedge reset_n)
  if(reset_n == 1'b0)
    cnt <= 25'd0;
  else if(cnt == CNT_MAX)
    cnt <= 25'd0;
  else
    cnt <= cnt + 1'b1;

//led 输出控制进程
  always@(posedge clk or negedge reset_n)
  if(reset_n == 1'b0)
    led <= 1'b1;
  else if(cnt == CNT_MAX)
    led <= ~led;
  else
    led <= led;

endmodule
```

源代码设计完成后，PLL_LED.v 文件由于包含了 counter.v 文件，所以 PDS 软件将 PLL_LED.v 自动识别成顶层文件。

计数器模块使用参数化定义的方式先声明一个参数化常量“CNT_MAX”，这样顶层文件调用 counter 模块时就可以通过以下方式。

```
counter
  #(
    .CNT_MAX(25'd24_999_999)
  )
  counter0(
    .clk(c0),
```

```
    .reset_n(reset_n),
    .led(LED[0])
  );
```

使用 PDS 软件对工程进行分析和综合，注意观察综合报告，查看是否有语法或逻辑报错，如果有报错，根据报错内容修正代码并重新进行分析和综合，直到设计分析和综合通过。

9.2 激励创建及仿真测试

当顶层文件创建好之后，我们可以通过仿真方式来对该顶层文件进行测试，进而分析和验证顶层文件在逻辑设计时是否符合实验要求。这里针对该顶层文件，编写一个简单的 testbench 来进行仿真测试，testbench 代码如下所示。

```
`timescale 1ns/1ps
`define clk_period 20

module PLL_LED_tb;
    reg Clk;
    reg Rst_n;
    wire [3:0]LED;

    PLL_LED PLL_LED(
        .Clk(Clk),
        .Rst_n(Rst_n),
        .LED(LED)
    );

    defparam PLL_LED.counter0.CNT_MAX = 24;
    defparam PLL_LED.counter1.CNT_MAX = 24;
    defparam PLL_LED.counter2.CNT_MAX = 24;
    defparam PLL_LED.counter3.CNT_MAX = 24;

    initial Clk = 1;
    always #(`clk_period/2)Clk = ~Clk;

    initial begin
        Rst_n = 1'b0;
        #(`clk_period * 20 + 1);
        Rst_n = 1'b1;
        #(`clk_period * 2000);
        $stop;
    end
endmodule
```

为了测试仿真编写测试激励文件，同时为了减少仿真时间，这里将计数器的计数值从 24_999_999 修改为 24。

如何对模块仿真这里我们不再赘述，仿真波形如图 9-17 所示

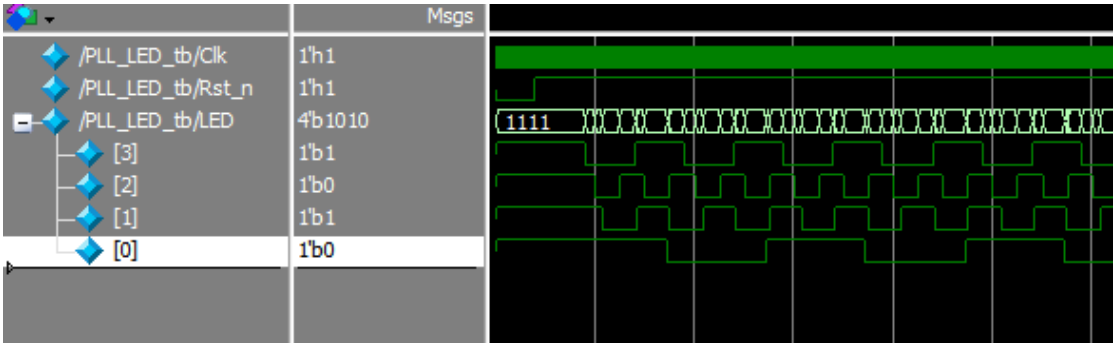


图 9-17 仿真波形

由于代码设计中 LED[0]由 PLL IP 核输出的 clkout0 通过连线赋值给 c0（25MHz 的时钟）驱动；LED[1]由 PLL IP 核输出的 clkout1 通过连线赋值给 c1（75MHz 的时钟）驱动 ；LED[2]由 PLL IP 核输出的 clkout2 通过连线赋值给 c2(100MHz 的时钟）驱动;LED[3]是由系统时钟（50MHz 的时钟）驱动。从图 9-18 中可以看出 clkout0 的周期是输入系统时钟周期的 2 倍；是 clkout1 的周期的 3 倍；是 clkout2 的 4 倍。由于 LED 是由不同频率的时钟控制，所以各 LED 灯亮灭的周期之间的倍数关系应该与驱动各 LED 灯的时钟周期倍数保持一致。

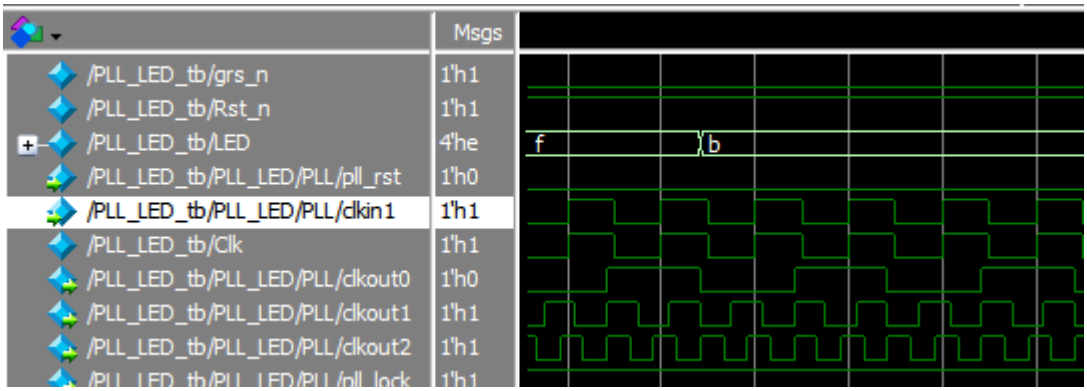


图 9-18 仿真波形

对比图 9-18 与图 9-17 可以看出代码设计的正确性。如果需要更为细致的验证，可以外接示波器观察波形，这里我们不做过多介绍。

9.3 板级调试与验证

实验的板级验证环节，主要验证以下几个目标：

1. 能否正确将生成的 bit 文件下载到 PGL22G 开发板；
2. 下载完成后能否正确显示 PGL22G 开发板 LED 的点亮实验现象；

系统所需硬件：

1. PGL22G 开发板；
2. 电源电缆一根；
3. 硬件条件符合实验要求，具有完全开发功能的 PC 机一台；

9.3.1 添加 I/O 约束

通常，一个设计中的 FPGA 不会是独立使用的，FPGA 一定会与其他外设、接口相连接，比如时钟，按键等。因此，FPGA 设计需要指定对应的 IO 引脚位置信息。

添加 IO 约束的方法非常简单：首先点击图 9-19 上方工具栏“Tools”栏中“User Constranit Editor(Timing and Logic)”然后点击 “Pre Synthesize USE”选项。

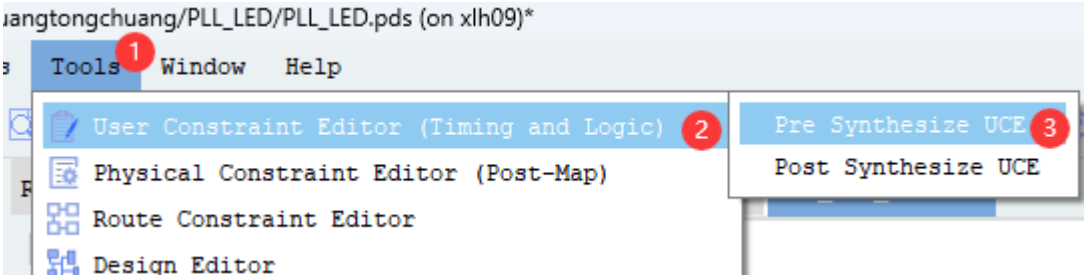


图 9-19 “Tools”工具栏

点击图 9-19 中的“Pre Synthesize USE”选项后，弹出图 9-20 界面，首先点击“Pre Synthesize USE” 选项，然后点击“Device”选项，最后点击“I/O”选项，进入管脚分配页面。

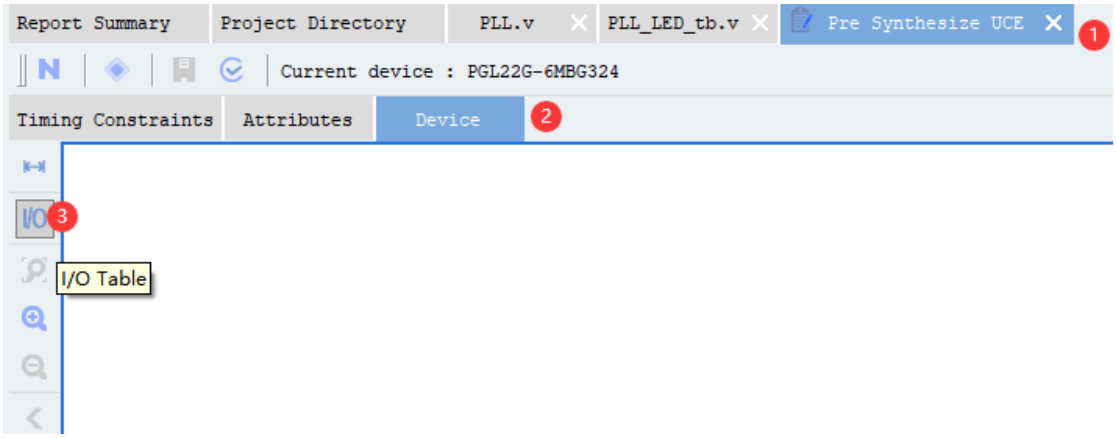
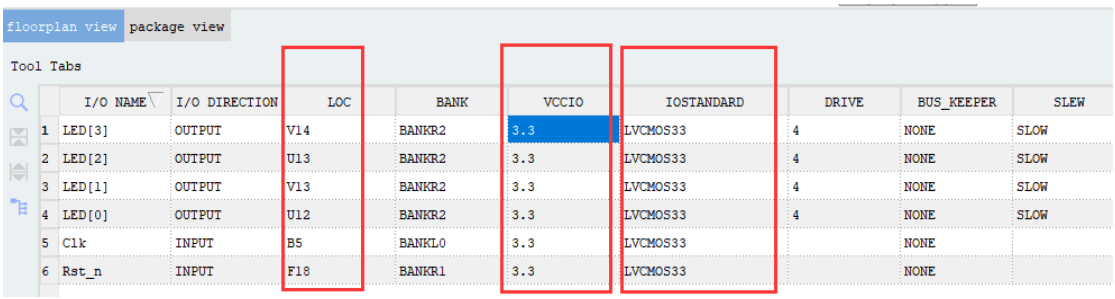


图 9-20 管脚分配入口

打开管脚约束窗口如图 9-21，其中 LOC ,VCCIO,IOSTANDARD 是我们要约束的内容。这里，参考我们提供的管脚约束表即可完成管脚绑定。



	I/O NAME	I/O DIRECTION	LOC	BANK	VCCIO	IOSTANDARD	DRIVE	BUS_KEEPER	SLEW
1	LED[3]	OUTPUT	V14	BANKR2	3.3	LVCMOS33	4	NONE	SLOW
2	LED[2]	OUTPUT	U13	BANKR2	3.3	LVCMOS33	4	NONE	SLOW
3	LED[1]	OUTPUT	V13	BANKR2	3.3	LVCMOS33	4	NONE	SLOW
4	LED[0]	OUTPUT	U12	BANKR2	3.3	LVCMOS33	4	NONE	SLOW
5	Clk	INPUT	B5	BANKL0	3.3	LVCMOS33		NONE	
6	Rst_n	INPUT	F18	BANKR1	3.3	LVCMOS33		NONE	

图 9-21 管脚分配页面

这样管脚约束添加完成了。但是此时约束内容保存在内存中，还没有写入文件，点击工具栏保存按钮(图 9-22 中箭头指向的位置)，或者直接 Ctrl+S。

图 9-22 保存约束文件

然后会自动弹出图 9-23 中所示界面，添加约束文件名PLL_LED，点击“SAVE”按钮。

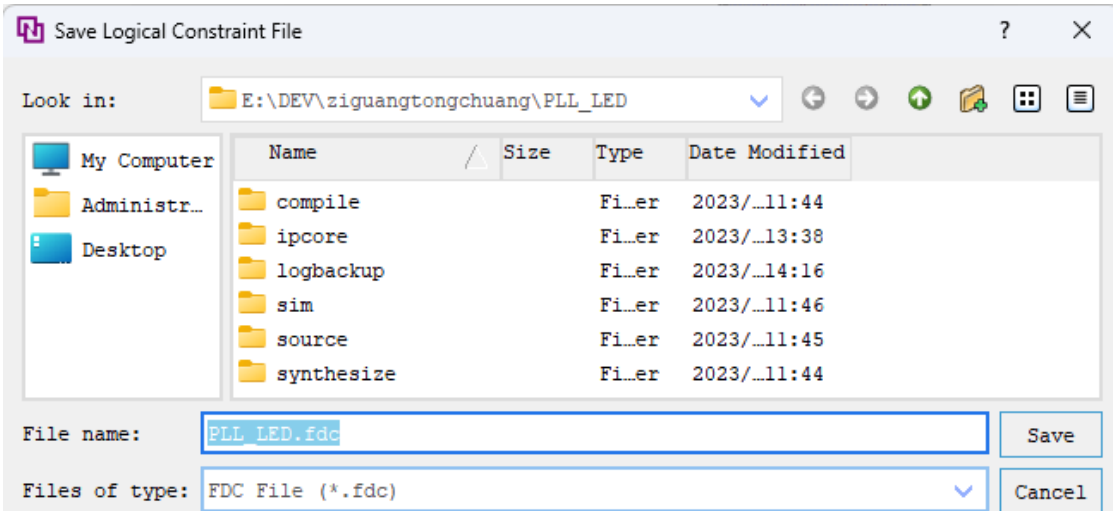


图 9-23 约束文件命名

这样就将 I/O 约束写入到约束文件中。如图 9-24 所示，在 Source 窗口的 Constraints 下可找到刚保存的PLL_LED.fdc 文件。双击可以打开约束文件。

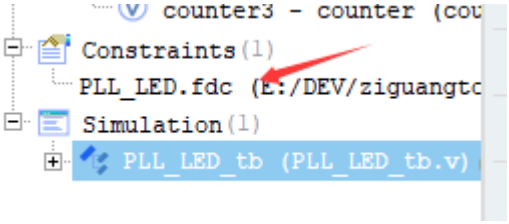


图 9-24 Source 面板

9.3.2 下载与验证

将下载器一端连接电脑，另一端与开发板上的 JTAG 下载口连接，连接电源线，并打开开发板的电源开关，硬件连接如下。

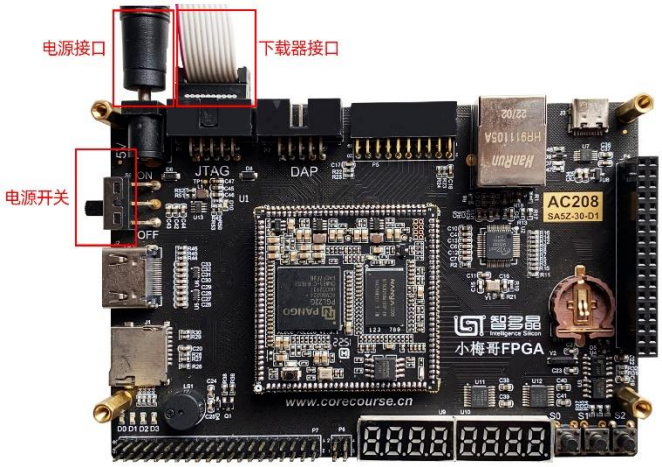


图 9-25 硬件连接

接下来我们直接点击图 9-26 中箭头指向的“Generate Bitstream”,PDS 软件会自动执行综合，布局布线，和 Bit 流生成。

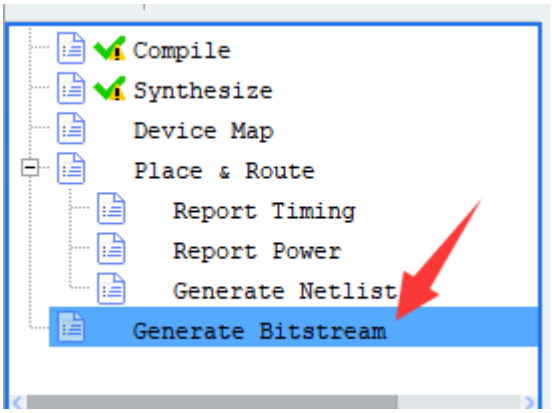


图 9-26 比特流生成界面

点击图 9-27 箭头指向的 PDS 工具栏下载按钮。

图 9-27 下载按钮

然后在弹出的图 9-28 中的 Fabric Configuration 界面中双击“Boundary Scan”。

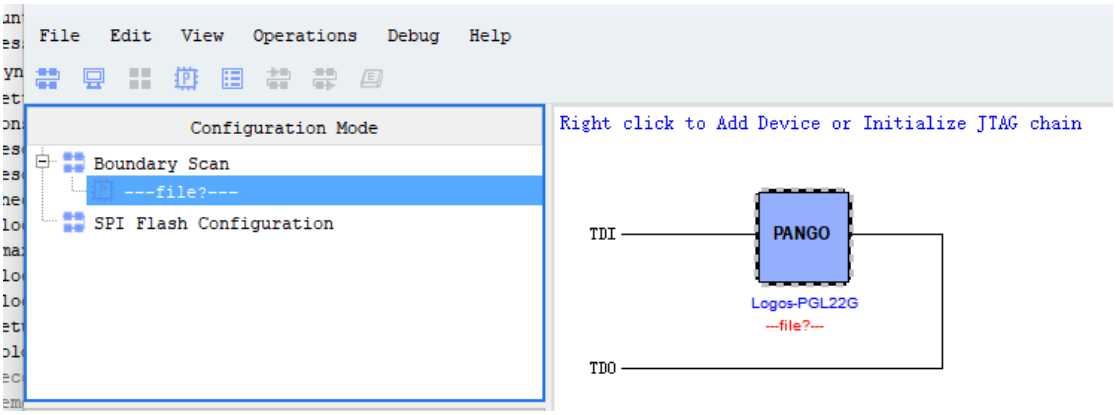


图 9-28 Fabric Configuration 界面

双击图 9-28 图 9-28 中的“Boundary Scan”后会自动弹出图 9-29，我们选中图 9-29 中 PLL_LED.sbit 文件，然后点击图中的 Open。

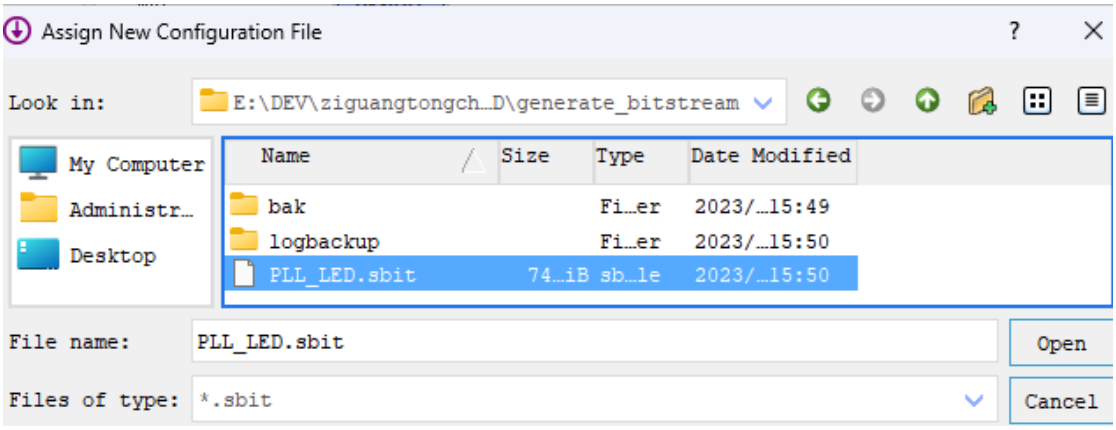


图 9-29 加载 PLL_LED.sbit 文件

右击图 9-30 中标签 1 所在的位置，然后点击标签 2 指向的“Program”将生成好的 PLL_LED. sbit 流文件下载到开发板中去。

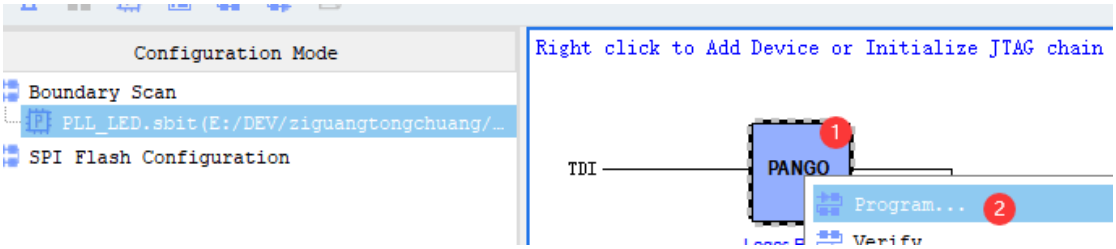


图 9-30 下载 PLL_LED.sbit 文件

实验现象如图 9-31 所示。

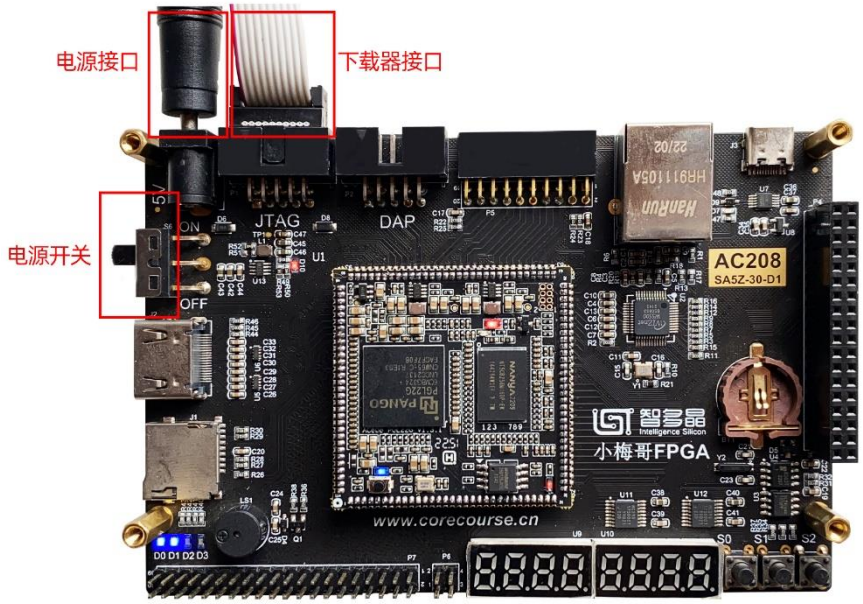


图 9-31 实验现象

将下载 PLL_LED.sbit 流文件下载到开发板后可以观察到对应的现象，即各个 led 灯的闪烁情况。

9.4 常见问题

1. 明确 LED 的一端连接的是高电平，所以在 FPGA 驱动端赋值逻辑 0 可以得到高电平，以此可以点亮 LED 灯。
2. 例化的目的是在上一级模块中调用已例化的模块完成代码功能。在 FPGA 中，模块名必须与要例化的模块名一致。

9.5 总结

本章介绍了锁相环 PLL 的组成及各部分的功能，使用 IP 核实现一个含有倍频和分频的 PLL，并在设计过程中学习了参数化定义的使用方法。同时，使读者对本开发板有一个初步的上手体验。建议读者能够跟随本实验内容，完整的进行整个实验。