

Vivado下 PLL 实验

黑金动力社区 2018-02-27

1 文档简介

很多初学者看到板上只有一个特定时钟输入的时候都产生疑惑,比如时钟输入 50MHz 或其它频率,那么我要产生一个想要的频率,如 100Mhz、150Mhz 等怎么办? 其实在很多 FPGA 芯片内部都集成了 PLL,其他厂商可能不叫 PLL,但是也有类似的功能模块,通过 PLL 可以倍频分频,产生其他很多时钟。本实验通过调用 PLL IP core 来学习 PLL 的使用、vivado 的 IP core 使用方法。

2 实验环境

- Windows 10 64 位
- vivado(vivado2019.1)
- 黑金 FPGA 开发板(AX7035 开发板、AX7050 开发板)
- 示波器

3 实验原理

PLL(phase-locked loop),即锁相环。是 FPGA 中的重要资源。由于一个复杂的 FPGA 系统往往需要多个不同频率,相位的时钟信号。所以,一个 FPGA 芯片中 PLL 的数量是衡量 FPGA 芯片能力的重要指标。FPGA 的设计中,时钟系统的 FPGA 高速的设计极其重要,一个低抖动,低延迟的系统时钟会增加 FPGA 设计的成功率。

本实验将通过使用 PLL, 输出一个方波到开发板上的扩展口(AX7035 开发板 J10 的 PIN3 脚, AX7050 开发板 J8 的 PIN3 脚), 来给大家演示在 Vivado 软件里使用 PLL 的方法。

7 系列的 FPGA 使用了专用的全局(Global)和区域(Regional)IO 和时钟资源来管理设计中各种的时钟需求。Clock Management Tiles(CMT)提供了时钟合成(Clock frequency synthesis),倾斜矫正 (deskew),过滤抖动(jitter filtering)功能。

每个 CMTs 包含一个 MMCM(mixed-mode clock manager)和一个 PLL。如下图所示, CMT 的输入可以是 BUFR, IBUFG, BUFG, GT, BUFH, 本地布线(不推荐使用), 输出需要接到 BUFG 或者 BUFH 后再使用



- 混合模式时钟管理器(MMCM)
 MMCM 用于在与给定输入时钟有设定的相位和频率关系的情况下,生成不同的时钟信
 - 号。 MMCM 提供了广泛而强大的时钟管理功能,

MMCM 内部的功能框图如下图所示:



PLL 内部的功能框图如下图所示:

 \geq

数字锁相环(PLL) 锁相环 (PLL)主要用于频率综合。使用一个 PLL 可以从一个输入时钟信号生成多个时钟信号。



想了解更多的时钟资源,建议大家看看 Xilinx 提供的文档"7 Series FPGAs Clocking Resources User Guide"。

4 建立工程

本实验中为大家演示如果调用 Xilinx 提供的 PLL IP 核来产生不同频率的时钟,并把其中的一个时钟输出到 FPGA 外部 IO 上,也就是 AX7035 开发板 J10 的 PIN3 脚, AX7050 开发板 J8 的 PIN3 脚。

下面为程序设计的详细步骤。

1. 新建一个 pll_test 的工程,点击 Project Manager 界面下的 IP Catalog。



2. 再在 IP Catalog 界面里选择 FPGA Features and Design\Clocking 下面的 Clocking Wizard,双击打开 配置界面。

Project Summary 🗙 IP Catalog 🗙		
Cores Interfaces		
Q ¥ ♦ ₽ +t, ♪ ₽	• •	
Search: Q-		
Name	^ 1 AXI4	Status Licen
> 🗀 Utility		
FPGA Features and Design		
🗸 📄 Clocking		
👎 Clocking Wizard	AXI4	Production Inclue
> 📄 IO Interfaces		
> 📄 Soft Error Mitigation		
> 🚍 XADC		
> 🚍 Kernels		
> 📄 Math Functions		
> 📄 Memories & Storage Elements		

3. 默认这个 Clocking Wizard 的名字为 clk_wiz_0, 这里我们不做修改。在第一个界面 Clocking Options 里 ,我们选择 PLL 资源,输入的时钟频率为 50Mhz。

	Component Name clk_wiz_0		
	Clocking Options Output Clocks Port Renaming	PLLE2 Settings Summary	
	Clock Monitor		
	Enable Clock Monitoring		
	Primitive		
	◯ MMCM ④ PLL		
	Clocking Features	Jitter Optimization	
out1	✓ Frequency Synthesis 🗌 Minimize Power	 Balanced 	
	✓ Phase Alignment	O Minimize Output Jitter	
	Dynamic Reconfig	O Maximize Input Jitter filtering	J
	Safe Clock Startup		
1 —	Dynamic Reconfig Interface Options		
	AXI4Lite DRP	Ity Cycle Config 📄 Write DRP registers	
	Input Clock Information		
	Input Cleok Port Name Input Freeu	enev(MHz)	r Options
	Primary clk_in1 50.000	🛞 11.000 - 800.000 UI	

4. 在 Output Clocks 界面里选择 clk_out1~clk_out4 四个时钟的输出,频率分别为 200Mhz, 100Mhz, 50Mhz, 25Mhz。这里还可以设置时钟输出的相位,我们不做设置,保留默认相位, 点击 OK 完成,

locking Wizard (5.4)	h to Defaulte					
IP Symbol Resource	Component Name	clk_wiz_0				
Show disabled ports	Clocking Options	Output Clock	s Port Renamir	g PLLE2 Settings	Summary	
	The phase is calc	ulated relative to	the active input clo	ck.		
	Output Clock	Port Name	Output Freq (MHz)		Phase (degrees)	
	o apar oroon	- or maine	Requested	Actual	Requested	Actu
	Clk_out1	clk_out1	200.000	200.000	0.000	0.00
	✓ clk_out2	clk_out2	100.000	100.000	0.000	0.00
clk_out1 - clk_out2 - reset clk_out3 -	✓ clk_out3	clk_out3	50.000	50.000	0.000	0.00
	✓ clk_out4	clk_out4	25.000	25.000	0.000	0.00
cik_out4 -	clk_out5	clk_out5	100.000	N/A	0.000	N/A
	clk_out6	clk_out6	100.000	N/A	0.000	N/A
	USE CLOCK	SEQUENCING		Clocking Feedback		
	Cutput Clock	k Somonco	Numbor	Source		>

5. 在弹出的对话框中点击 Generate 按钮生成 PLL IP 的设计文件。

🍌 Generate Output Products				
The following output products will be generated.	4			
Preview				
Q, 素 ≑				
✓ ₽ ■ clk_wiz_0.xci (OOC per IP)	<u>^</u>			
Instantiation Template				
Synthesized Checkpoint (.dcp)				
Structural Simulation				
Synthesis Options				
O <u>G</u> lobal				
Out of context per IP				
Run Settings				
Number of jobs: 4				
Apply Generate S	S <u>k</u> ip			

6. 这时一个 clk_wiz_0.xci 的 IP 会自动添加到我们的 pll_test 项目中, 用户可以双击它来修改这个 IP 的配置。

<u>F</u> ile <u>E</u> dit F <u>l</u> ow <u>T</u> ools <u>W</u> indow L	.ayout View Help Q- Quick Access						
🖕							
Flow Navigator 😤 🗢 ? _ PROJECT MANAGER - pll_test							
✓ PROJECT MANAGER	Sources ? _ O C X						
🍄 Settings							
Add Sources							
Language Templates	> Celk wiz 0 (clk wiz 0 xci)						
₽ IP Catalog	= Constraints (1)						
	> 🚍 Simulation Sources (1)						
Y IP INTEGRATOR							
Create Block Design							
Open Block Design							
Generate Block Design							
✓ SIMULATION	Hierarchy IP Sources Libraries Compile Order						
Run Simulation	Source File Properties $? = \Box \ \square \times$						
	₽ clk_wiz_0.xci ← → 🌣						

选择 IP Sources 这页,然后双击打开 clk_wiz_0.veo 文件,这个文件里提供了这个 IP 的实例化模板。我们只需要把框框的中内容拷贝到我们 verilog 程序中,对 IP 进行实例化。

PROJECT MANAGER - pll_test						
Sources	? _ O Ľ X	Project Summary x IP Catalog x clk_wiz_0.veo x				
Q ¥ ≑ +	٥					
✓ □ IP (1) ✓ □ IP (1) ✓ □ Instantiation Ten □ cik_wiz_0.ve	nplate (1) 9	61 // clk_out425.0000.000_50.0_236.428_164 62 // 63 // 64 // Input Clock Freq (MHz) Input Jitter (UI) 65 // 66 //				
Hierarchy IP Sources L	ibraries Compile Order	71 72 73 73 74 clk_wiz_0 instance_name 75 (76 76 77 77 78 79 79 79 70 70 70 70 70 70 70 70 70 70 70 70 70				
Pick wiz 0 veo		77				
	- 3 Åt	78 .elk_out2(elk_out2), // output clk_out2 79 .elk_out3(elk_out3), // output clk_out3 80 .elk_out4(elk_out4) // output clk_out4				
CLASS	file	81 // Status and control signals				
CORE_CONTAINER		82 .reset(reset), // input reset				
FILE_TYPE	Verilog Template 🗸 🗸 🗸 🗸 🗸 🗸	83 .locked(locked), // output locked				
IS_AVAILABLE	~	84 // Ulock in ports 85 clk int(clk int)) // input clk int				
IS ENABLED		86 // INST_TAG_END End INSTANTIATION Template				
IS GENERATED	· · ·	87				

7. 我们再来编写一个顶层设计文件来实例化这个 PLL IP, 编写 pll_test.v 代码如下。

```
timescale 1ns / 1ps
module pll test(
input sys_clk,
                                   //system clock 50Mhz on board
                                   //reset ,low active
input
         rst n,
output
         clk out
                                   //pll clock output
);
wire
          locked;
wire
          pll clk o;
PLL IP calling
****
                clk_wiz_0 clk_wiz_0_inst
(// Clock in ports
.clk in1
          (sys_clk ),
                                 // IN 50Mhz
// Clock out ports
                   ),
.clk out1 (
                                 // OUT 200Mhz
.clk_out2 (
                    ),
                                 // OUT 100Mhz
                                 // OUT 50Mhz
.clk_out3 (
                    ),
.clk out4 (pll clk o ),
                                 // OUT 25Mhz
// Status and control signals
.reset (~rst n
                                  // RESET IN
                     ),
          (locked
.locked
                     )
);
Calling ODDR to make the clock signal output through normal IO
ODDR #(
.DDR CLK EDGE ("SAME EDGE")
)
ODDR inst
(
                                  // 1-bit DDR output data
.0
         (clk out
                     ),
.C
         (pll_clk_o ),
                                  // 1-bit clock input
.CE
          (1'b1 ),
                                  // 1-bit clock enable input
         (1'b1
                     ),
.D1
                                  // 1-bit data input (associated with C)
.D2
                    ),
         (1'b0
                                  // 1-bit data input (associated with C)
          (1'b0
(1'b0
.R
                    ),
                                  // 1-bit reset input
.S
                                   // 1-bit set input
                     )
);
endmodule
```

程序中先用实例化 clk_wiz_0, 把单端 50Mhz 时钟信号输入到 clk_wiz_0 的 clk_in1, 把 clk_out4 的输出付给 pll_clk_o 网络。

注意:例化的目的是在上一级模块中调用例化的模块完成代码功能,在 Verilog 里例化信号的 格式如下:模块名必须和要例化的模块名一致,比如程序中的 clk_wiz_0,包括模块信号名也必须 一致,比如 clk_in1, clk_out1, clk_out2.....。连接信号为 TOP 程序跟模块之间传递的信号,模块 与模块之间的连接信号不能相互冲突,否则会产生编译错误。

模块名 扩展名↔	
(+ ¹	
.模块信号1	(连接信号1),-
.模块信号 2	(连接信号 2),-
.模块信号 3	(连接信号 3),↩
	به
.模块信号 N	(连接信号 N)→
); +	

另外我们程序中添加了一个 ODDR 原语,使得 clk_wiz_0 的 BUFG 输出的时钟信号能够输出到 FPGA 的普通 IO。因为如果直接从 BUFG 上连接到 OBUF 上,在编译器 map 的过程中就会出现错误, 为避免这个错误,通过 ODDR 把两路单端的数据合并到一路上输出,上下沿同时输出数据,上沿 输出 a 路下沿输出 b 路;如果两路输入信号一路恒定为 1,一路恒定为 0,那么输出的信号实际上 就是输入的时钟信号 pll_clk_o。

另一种方法就是在约束文件中加上一条约束,让编译器忽略时序约束的要求,直接通过普通逻辑资源连接。但这样 Clock 输出的时延和抖动 (Jitter)都会变差。

8. 保存工程后, pll_test 自动成为了 top 文件, clk_wiz_0 成为 Pll_test 文件的子模块。



9. 再为工程添加 xdc 管脚约束文件 pll.xdc。

5 仿真

添加一个 vtf_pll_test 仿真文件,运行后 PLL 的 lock 信号会变高,说明 PLL IP 锁相环已经初始 化完成。clk_out 有时钟信号输出,输出的频率为输入时钟频率的 1/2,为 25Mhz。

	pll_test.v × clk_wiz_0.	v × clk_w	iz_0.veo × pll.	xdc × vtf_	pll_test.v ×	Untitled 1* \times	
•	Q 💾 🤤 Q 💱	•I 1 + 14) N 1≛ ≛r	+ F F ∈ .	•ľ ⊡+		
				617.600 ns			
^	Name	Value	615	ns 62	0 ns	525 ns	630 ns
	1‰ rst_n	1					
	15 sys_clk	1					
	น₀ clk_out	1					
	🖫 locked	1					

6 测量 PLL 输出波形

编译工程并生成 pll_test.bit 文件,再把 bit 文件下载到 FPGA 中,接下去我们就可以用示波器 来测量输出时钟波形了。

用示波器探头的地线连接到开发板上的地(AX7035 开发板 J10 的 PIN1, AX7050 开发板 J8 的 PIN1 脚), 信号端连接 AX7035 开发板 J10 的 PIN3 脚或 AX7050 开发板 J8 的 PIN3 脚(测量的时候 需要注意,避免示波器表头碰到其它管脚而导致电源和地短路)。

这时我们可以在示波器里看到 25Mhz 的时钟波形,波形的幅度为 3.3V,占空比为 1:1,波形显示如下图所示:



如果您想输出其它频率的波形,可以修改时钟的输出为 clk_wiz_0 的 clk_out2 或 clk_out3 或 clk_out4。也可以修改 clk_wiz_0 的 clk_out4 为您想要的频率,这里也需要注意一下,因为时钟的输出是通过 PLL 对输入时钟信号的倍频和分频系数来得到的,所以并不是所有的时钟频率都可以用 PLL 能够精确产生的,不过 PLL 也会自动为您计算实际输出接近的时钟频率。

另外需要注意的是,有些用户的示波器的带宽和采样率太低,会导致测量高频时钟信号的时候,高频部分衰减太大,测量波形的幅度会变低。

7 附录

pll_test.v(verilog 代码)

```
`timescale 1ns / 1ps
module pll_test(
input
        sys clk,
                                  //system clock 50Mhz on board
input
         rst n,
                                  //reset ,low active
output
                                  //pll clock output
          clk out
);
          locked;
wire
          pll_clk_o;
wire
PLL IP calling
clk_wiz_0 clk_wiz_0_inst
(// Clock in ports
.clk in1
          (sys clk
                    ),
                                 // IN 50Mhz
// Clock out ports
```

黑金动力社区



```
.clk out1
                                         // OUT 200Mhz
                         ),
             (
.clk_out2
                                         // OUT 100Mhz
            (
                         ),
.clk out3
                                         // OUT 50Mhz
            (
                         ),
.clk out4
            (pll clk o
                                         // OUT 25Mhz
                         ),
// Status and control signals
            (~rst n
                                         // RESET IN
.reset
                         ),
.locked
            (locked
                         )
);
Calling ODDR to make the clock signal output through normal IO
ODDR #(
.DDR CLK EDGE ("SAME EDGE")
)
ODDR inst
(
.Q
            (clk_out
                         ),
                                          // 1-bit DDR output data
• C
            (pll_clk_o
                                          // 1-bit clock input
                         ),
.CE
            (1'b1
                         ),
                                          // 1-bit clock enable input
.D1
            (1'b1
                                          // 1-bit data input (associated with C)
                         ),
            (1'b0
                         ),
                                          // 1-bit data input (associated with C)
.D2
            (1'b0
.R
                                          // 1-bit reset input
                         ),
.S
            (1'b0
                         )
                                          // 1-bit set input
);
endmodule
```