第5章 EDA宏功能资源利用

CHAPTER 5

Intel公司的宏功能(Megafunction)是重要的设计输入资源。由于宏功能是基于Intel公司底层硬件结构最合理的成熟应用模块的表现,因此在代码中尽量使用这类宏功能资源,不但能将设计者从烦琐的代码编写中解脱出来,更重要的是在大多数情况下宏功能的综合和实现结果比用户编写的代码更优。

宏功能包括Intel公司的参数化模块库(Library of Parameterized Modules,LPM)、Intel公司及协作者(Altera Megafunction Partners Program,AMPP)提供的第三方IP核,以及原Altera公司的特定功能IP核(ALT类)。

特别是对于一些与Intel公司器件底层结构相关的特性,必须通过宏功能实现。例如,一些存储器模块(如DPRAM、SPRAM、FIFO、CAM等)、DSP模块、LVDS驱动器、PLL、高速串行收发器(如SERDES)和DDR输入/输出等。另外一些如乘法器、计数器、加法器、滤波器等电路虽然也可以直接用代码描述,然后用通用逻辑资源实现,但是这种描述方法不但费时费力,在速度和面积上与宏功能的实现结果仍然有较大差距。

宏功能的使用方法主要有两种:一种方法是直接在代码中实例化和配置用AHDL(Altera HDL)编写的LPM模块。原Altera公司的大部分宏功能的源文件是用AHDL写成的,文件扩展名是.tdf。另外一种是使用MegaCore/Mega Wizard(近期版本称为IP Catalog)工具调用和配置参数化的IP和底层模块。

5.1 参数化模块库

说到参数化模块库(Library of Parameterized Modules,LPM),就一定要谈谈EDIF(Electronic Design Interchange Format)。EDIF文件是EDA厂商之间和EDA厂商与IC厂商之间传递设计信息的文件格式。LPM最初是作为EDIF标准的附件出现的。

EDIF和LPM的标准化过程如下:

1988年,ANSI/EIA-548:电子设计交互格式(Electronic Design Interchange Format,EDIF),版本2.0.0。

1990年,LPM标准提出,供EIA审核。

1993年,EIA 618:电子设计交互格式(Electronic Design Interchange Format,EDIF),版本3.0.0级别0参考手册,LPM作为EDIF标准的附件,成为EIA的一个过渡标准。

1995年,EIA PN 3714:参数化模块库(Library of Parameterized Modules,LPM),版本2.0.1。

1996年,ANSI/EIA-682:电子设计交互格式(Electronic Design Interchange Format,EDIF),版本4.0.0(EIA-682-96)。

1999年,EIA/IS-103A:参数化模块库(Library of Parameterized Modules,LPM),版本2.0。

从年代上看来,1988年到1990年前后恰好是半定制设计风格超越全定制设计风格成为VLSI芯片设计主流的时期。LPM标准的提出可能正是响应了半定制设计的需求。

EDIF文件是EDA工具之间传递信息的标准格式。画过电路原理图和PCB的读者一定知道,原理图文件绘制完毕后需要"生成网表",进行PCB布局布线之前先要"引入网表",这样才能建立原理图文件和PCB文件之间的"逻辑映射关系"。EDIF文件就是网表文件的一种格式。在很多情况下,原理图文件中的模块图形和PCB文件中的"封装"是一一对应的,这种"物理映射关系"就是通过"库文件"建立的。"库文件"包含了原理图模块的名称和图形,也包含了封装文件的名称和图形,这样一来,"物理映射关系"就建立起来了。在不同的EDA工具之间,比如Protel和Cadence还有PowerPCB,逻辑映射关系是很容易互相通用的,但是由于支持不同的"库文件",物理映射关系往往就建立不起来。

在IC设计领域(包括PLD设计),EDIF文件就遇到了类似的问题:综合工具和实现工具必须达成一致。在LPM标准提出之前,这一点很难实现,毕竟IC设计领域存在太多的实现工艺和EDA工具。

在LPM标准提出之前,对于某些逻辑的描述没有统一的标准,描述方法都是工艺相关(Technology Dependent)的,所以综合工具生成的EDIF文件不具备可移植性。在采用了LPM标准之后,对于LPM库中包含的逻辑,所有的综合工具都采用同一种行为描述方法,生成相同的EDIF文件,实现设计输入和网表的正确映射;实现工具包含各自工艺库与LPM库之间的唯一映射关系,从而能够"读懂"包含LPM描述的EDIF文件,实现网表和工艺之间的正确映射。这样一来,EDIF文件在不同的实现工具之间移植就不成问题了。(LPM并不是唯一的解决方法,比如现在的EDA工具之间往往互相支持对方特定的库文件和网表格式,尤其像Synopsis这样的专业EDA公司,同时支持许多公司的器件和网表格式及宏单元;而原Altera公司和原Xilinx公司就不能互相支持。)

在这一过程中体现的原理是:通过增加一个映射层次,把一次映射关系转化为两次映射关系,两次映射关系的中介——包含LPM描述的EDIF文件——就具备了可移植性。

图5-1可以更清晰地表述上述内容,不过需要细看才能看懂:

LPM标准的提出还解决了设计者面临的图形输入法可移植性差和HDL输入法硅片利用效率低的两难困境。

采用图形输入方法可以很精确地描述底层实现细节,综合工具不需要推测设计者的意图就能很准确地生成EDIF文件,效率很高。但是由于包含了硬件实现细节,只有专用的实现工具(布局布线工具)才能"读懂"这样的EDIF文件。这样一来,就需要设计输入(原理图)工具——综合工具——实现工具严格一致,带来了图形描述文件的可移植性问题。

采用HDL输入方法避免了从门级描述硬件细节,只要综合工具——实现工具达成一致就不存在HDL文件(设计输入文件)的可移植性问题。但是对于同一个逻辑功能,缺乏统一的描述方法,最后的实现效率取决于综合工具,实现效率往往不如图形输入方法。

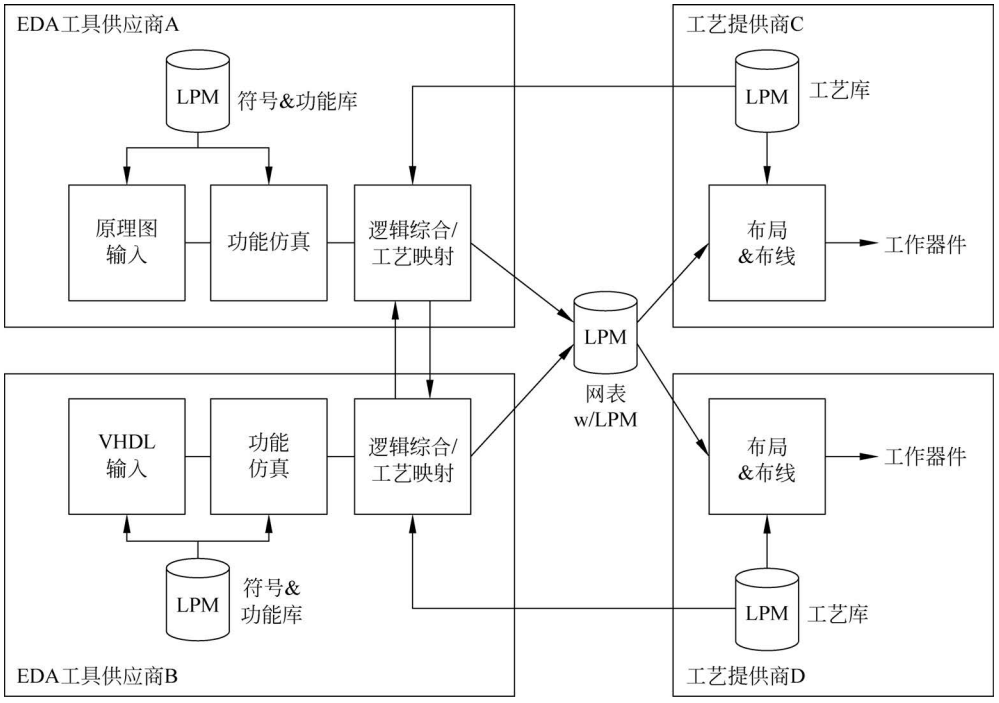


图 5-1　LPM 的可移植性

通过采用 LPM 标准，设计输入工具、综合工具、实现工具对于同一个逻辑功能在不同抽象层次的描述达成了共识：设计输入工具调用 LPM 模块，综合工具或者实现工具保证和实现 LPM 模块描述的逻辑功能和实现工艺之间的唯一映射。从可移植性角度看来，由于在设计输入阶段不需要涉及具体实现工艺，LPM 输入法具备与 HDL 输入法同等的可移植性；从实现效率看来，由于综合工具或实现工具采用了最佳的映射，LPM 输入法具备与图形输入法同等的高效率。LPM 兼具了 HDL 输入法和原理图输入法的优点，而避免了二者各自的缺点。

采用 LPM 设计方法，可以带来四点好处：

(1) 设计文件具备独立于实现工艺的可移植性。

(2) 保证最佳的实现效率。

(3) 保证设计工具之间的互操作性。

(4) 可以完成几乎所有设计需要的逻辑描述。

其中后两点在今天看来还是有问题的：Intel(原 Altera)和 Modelsim 之间就不能实现 LPM 模块的自动同步，Modelsim 需要在仿真 Intel(原 Altera)的 LPM 模块前编译 Intel(原 Altera)的专用仿真库；采用 LPM 模块完成所有的逻辑描述还是有点儿麻烦(相对于 HDL 来说)。

LPM 包含 25 个基本模块，如表 5-1 所示，它们可以通过配置参数实现各种数据宽度的逻辑功能和多种不同的功能特性。

表 5-1　LPM 的 25 个基本模块

CONST	INV	AND	OR	XOR
LATCH	FF	SHIFTREG	RAM_DQ	RAM_IO
ROM	DECODE	MUX	CLSHIFT	COMPARE
ADD_SUB	MULTIPLER	COUNTER	ABS	BUSTRI
FSM	TTABLE	INPAD	OUTPAD	BIPAD

LPM 标准的价值在于是否有足够多的 EDA 厂商采用这一标准，从而保证最佳的互操作性。早在 1993 年，原 Altera 公司就支持 LPM 标准；在 1995 年年底之前，主要的 EDA 厂商也都会支持 LPM 标准；据 1995 年的说法，原 Xilinx 也将会在“近期”支持这一标准。

从上可见，LPM 标准确实有历史了，基本上是二十多年前的事。EDA 技术的更新换代非常迅速，二十多年前的 HDL 综合效率问题在今天看来已经不是主要矛盾。但是从提高资源利用效率和保证代码质量角度看来，LPM 仍然不失为一种有效的设计输入方法，仍然有其用武之地。

值得注意的是，在 QUARTUS 中，还有 MAXPLUS2 库和原语(PRIMITIVE)库，其中原语(PRIMITIVE)库包括常见的最基本的门，而 MAXPLUS2 库则包括各种常见的数字集成芯片，这两个库中的器件只能以器件符号的形式在原理图中出现，不能以 VHDL 的形式嵌入在设计中。而 LPM 模块是可以生成其所对应的 VHDL 描述，并将其嵌入在顶层设计中。同时 LPM 模块也可以以符号的形式出现在原理图中。

5.1.1 计数器

这里介绍利用 Mega Wizard 工具对 LPM 计数器 LPM_COUNTER 实现调用和配置，以及之后的仿真测试过程中遇到的一般性问题，具有示范意义。对于之后较复杂的 IP 模块则主要介绍功能特性仿真测试。关于 Mega Wizard Plug_In Manager 工具的相关软件操作方法，请参考实验视频。

LPM_COUNTER IP 核实现二进制计数器，可以进行递增计数、递减计数或增/减计数。

1. 功能和特点

LPM_COUNTER IP 核的主要功能特点如下所述。

(1) 生成实现递增计数、递减计数或增/减计数的计数器。

(2) 输出数据位宽最大支持 256 位。

(3) 生成如下计数器类型：

- 二进制计数：计数器从 0 开始递增或从 255 开始递减。
- 特定模计数：计数器递增计数到用户指定的特定模或从用户指定的特定模开始递减计数并重复。

(4) 支持可选的同步复位、加载和设置输入。

(5) 支持可选的异步复位、加载和设置输入。

(6) 支持可选的计数使能和时钟使能输入。

(7) 支持可选的进位输入和进位输出。

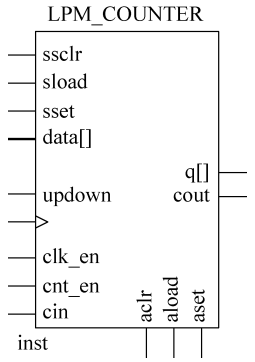


图 5-2 LPM_COUNTER 的接口信号图

2. 接口信号说明

LPM_COUNTER IP 核的接口信号图如图 5-2 所示。

LPM_COUNTER IP 核的接口信号的具体含义请参见表 5-2。

表 5-2 LPM_COUNTER IP 核的接口信号

信号名称	信号方向	说　明
data[]	I	并行数据输入总线，位宽由参数 LPM_WIDTH 决定
clock	I	上升沿触发的输入时钟信号
clk_en	I	时钟使能输入信号

续表

信号名称	信号方向	说　明
cnt_en	I	计数使能输入，低电平时禁止计数但不影响 sload，sset 和 sclr。默认值为 1
updown	I	计数方向控制信号。高电平时递增计数，低电平时递减计数。如果使用 LPM_DIRECTION 参数，则不能连接该端口信号；否则该端口是可选的，默认为 1
cin	I	进位输入到最低位。对于递增计数，cin 输入与 cnt_en 输入相同，默认为 1
aclr	I	异步复位信号，如果 aset 和 aclr 同时使用，则 aclr 的优先级高于 aset，默认为 0
aset	I	异步置位输入。将 q[]输出全部置为 1 或由参数 LPM_AVALUE 指定的值。如果 aset 和 aclr 同时使用，则 aclr 的优先级高于 aset，默认为 0
aload	I	异步加载输入，异步加载数据到计数器。使用该信号时必须已连接 data[]端口，默认为 0
sclr	I	同步复位信号，在该信号在下一个有效的时钟边沿复位计数器。如果 sset 和 sclr 同时使用，则 sclr 的优先级高于 sset，默认为 0
sset	I	同步置位输入。在该信号在下一个有效的时钟边沿置位计数器，将输出值全部置为 1 或由参数 LPM_SVALUE 指定的值。如果 sset 和 sclr 同时使用，则 aclr 的优先级高于 sset，默认为 0
sload	I	同步加载输入，在该信号在下一个有效的时钟边沿同步加载 data[]数据输入到计数器。使用该信号时，必须已连接 data[]端口，默认为 0
q[]	O	计数器的数据输出总线，位宽由参数 LPM_WIDTH 决定。必须连接 q[]或 eq[15..0]总线（至少连接 16 位总线中的 1 位）
eq[15..0]	O	计数器解码输出。该端口仅用于 AHDL，不能用参数编辑器访问。q[]端口或 eq[]端口是必须连接的，最多有 c 个 eq 端口可用（$0 \leqslant c \leqslant 15$）。仅对计数值的低 16 个计数值解码，如果计数值为 c，则 eqc 输出置为高电平。该信号与 q[]异步
cout	O	进位输出，计数器的 MSB 位。可以用于与其他计数器级联来创建更大的计数器

3. 参数设置

LPM_COUNTER IP 核的参数设置详细说明请参见表 5-3。

表 5-3　LPM_COUNTER IP 核的参数设置

参 数 名 称	类　型	说　明
LPM_WIDTH	整数	指定端口 data[]和 q[]的数据位宽
LPM_DIRECTION	字符串	值可以是 UP、DOWN 或 UNUSED。如果使用该参数，则不能连接 updown 端口。当未连接 updown 端口时，该参数的默认值为 UP
LPM_MODULUS	整数	最大计数值加 1，表示计数器时钟周期内独立状态个数
LPM_AVALUE	整数/字符串	在 aset 为高电平时加载的常数值。如果指定的值大于或等于＜modulus＞，则计数器的值是未定义的逻辑电平（X），这里＜modulus＞是 LPM_MODULUS 或 2^LPM_WIDTH
LPM_SVALUE	整数/字符串	在 sset 为高电平时在时钟上升沿加载的常数值
CARRY_CNT_EN	字符串	Intel 公司特定参数。在 VHDL 设计文件中必须用参数 LPM_HINT 指定该参数。值可以是 SMART、ON、OFF 或 UNUSED。用于使能 LPM_COUNTER 可以通过进位链传递 cnt_en 信号。默认值为 SMART，提供面积和速度的最优折中

续表

参数名称	类　型	说　明
LABWIDE_SCLR	字符串	Intel 公司特定参数，在 VHDL 设计文件中必须用 LPM_HINT 参数指定 LABWIDE_SCLR 参数。值可以是 ON、OFF 或 UNUSED，默认值为 ON
LPM_PORT_UPDOWN	字符串	指定 updown 输入端口的使用。值可以是 PORT_USED、PORT_UNUSED 或 PORT_CONNECTIVITY，默认值为 PORT_CONNECTIVITY。设置为默认值时通过检查端口连接来判定是否使用 updown 端口

此外 LPM_COUNTER IP 核还包括配置参数 LPM_HINT、LPM_TYPE 和 INTENDED_DEVICE_FAMILY，请读者自行查阅相关文献。

4. 调用生成元件模型

LPM COUNTER IP 核调用之后自动生成的 VHDL 模型如下所示。

【代码 5-1】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
LIBRARY LPM;      -- 参数化模块库
USE LPM.ALL;
ENTITY CNT8B6M IS
    PORT
    (
        aclr: IN STD_LOGIC ;
        clk_en: IN STD_LOGIC ;
        clock: IN STD_LOGIC ;
        data: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        sload: IN STD_LOGIC ;
        updown: IN STD_LOGIC ;
        cout: OUT STD_LOGIC ;
        q: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
END CNT8B6M;
ARCHITECTURE SYN OF cnt8b6m IS
    SIGNAL sub_wire0: STD_LOGIC ;
    SIGNAL sub_wire1: STD_LOGIC_VECTOR (7 DOWNTO 0);
    COMPONENT lpm_counter
    GENERIC (
        lpm_direction: STRING;
        lpm_modulus: NATURAL;
        lpm_port_updown: STRING;
        lpm_type: STRING;
        lpm_width: NATURAL
    );
    PORT (
          aclr: IN STD_LOGIC ;
          clk_en: IN STD_LOGIC ;
          clock: IN STD_LOGIC ;
          data: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          cout: OUT STD_LOGIC ;
          q: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
          sload: IN STD_LOGIC ;
```

```
            updown: IN STD_LOGIC
    );
    END COMPONENT;
BEGIN
    cout    <= sub_wire0;
    q    <= sub_wire1(7 DOWNTO 0);
    LPM_COUNTER_component : LPM_COUNTER
    GENERIC MAP (
        lpm_direction => "UNUSED",
        lpm_modulus => 6,
        lpm_port_updown => "PORT_USED",
        lpm_type => "LPM_COUNTER",
        lpm_width => 8
    )
    PORT MAP (
        aclr => aclr,
        clk_en => clk_en,
        clock => clock,
        data => data,
        sload => sload,
        updown => updown,
        cout => sub_wire0,
        q => sub_wire1
    );
END SYN;
```

上面的程序是 Quartus Ⅱ根据参数配置自动生成的文件。CNT8B6M 是利用 Mega Wizard 工具调用 lpm_counter 时自行命名的生成元件模型名称。lpm_counter 是 LPM 元件名，是可以从 LPM 库中调用的宏模块元件名；而 LPM_COUNTER_component 则是在此文件中为使用和调用 lpm_counter 元件的例化名，即参数传递语句中的宏模块元件例化名；其中的 lpm_direction 等称为宏模块参数名，是被调用的元件(lpm_counter)文件中已定义的参数名，而“UNUSED”等是参数值，它们可以是整数、操作表达式、字符串或在当前模块中已定义的参数。

调用 LPM_COUNTER IP 核，其具体参数设置如下：

(1) 输入端口和输出端口位宽设置为 8 位；

(2) 计数模为 6；

(3) 方向控制、时钟使能、异步复位输入端口和同步置数。

5. 例化和仿真

为了能调用生成的计数器元件 CNT8B6M，并测试和利用硬件实现它，有两种例化方式：其一是利用原理图符号例化，其二是利用 VHDL 程序来进行例化。

1) 利用原理图符号实现例化

在原理图绘制界面中，选择添加符号工具按钮，进入图 5-3 所示界面；然后选择工程文件夹，找到当前所调用宏功能模块所生成的元件符号 CNT8B6M，拖动到原理图绘制区，并给该元件添加引脚如下。将其设定为顶层仿真模块，进行仿真。用符号实现例化如图 5-4 所示。

2) 利用所生成 VHDL 程序完成例化

对宏功能 lpm_counter 调用所生成的元件 CNT8B6M 进行了例化，以验证和测试模块

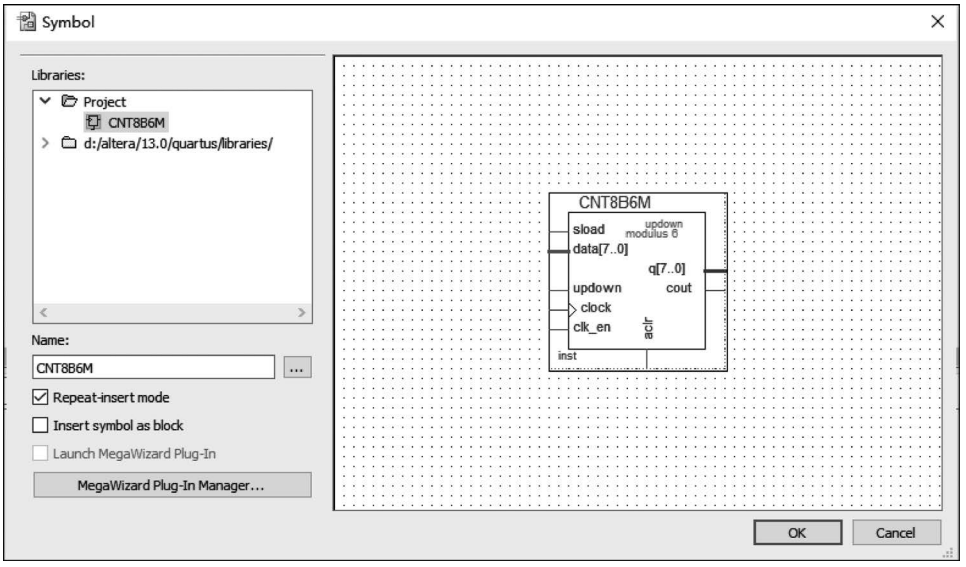


图 5-3　调用工程文件生成符号的界面

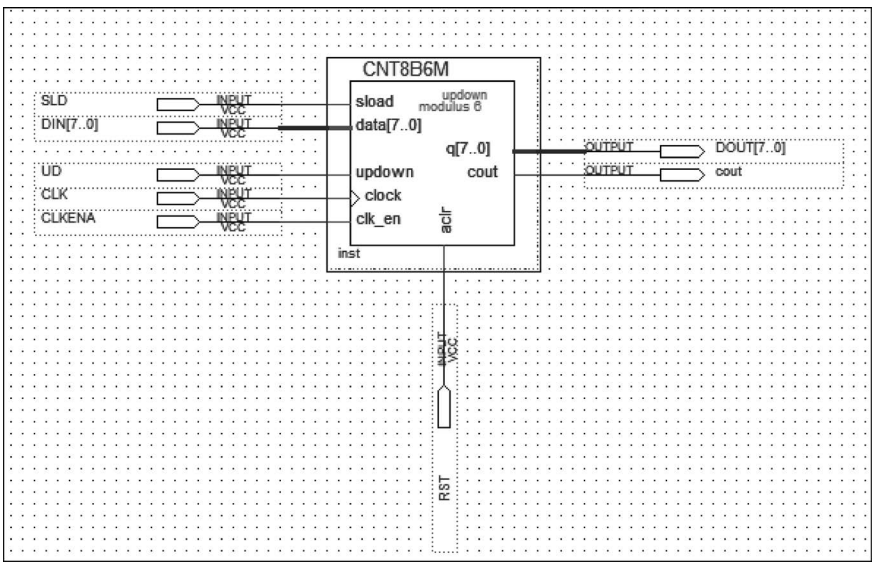


图 5-4　CNT8B6M 的符号例化元件

功能，除了直接利用其生成的元件原理图符号之外，还可以采用 VHDL 程序给出其顶层描述，具体如下所示。

【代码 5-2】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY CNT8B6MTOP IS
PORT (CLK,RST,CLKENA,SLD,UD:IN STD_LOGIC;
DIN : IN STD_LOGIC_VECTOR (7 DOWNTO 0);   COUT : OUT STD_LOGIC; DOUT : OUT STD_LOGIC_VECTOR(7
DOWNTO 0));
END ENTITY CNT8B6MTOP;
```

```
ARCHITECTURE top OF CNT8B6MTOP IS
COMPONENT CNT8B6M
PORT (aclr,clk_en, clock,sload,updown : IN STD_LOGIC ;
data : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
cout : OUT STD_LOGIC ;
q: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END COMPONENT;
BEGIN
U1:CNT8B6M PORT MAP (sload = > SLD, clk_en = > CLKENA, aclr = > RST,
clock = > CLK, data = > DIN, updown = > UD, cout = > COUT, q = > DOUT);
END ARCHITECTURE top;
```

上述两种例化方式得到的仿真结果完全相同，具体仿真结果如图 5-5 所示。

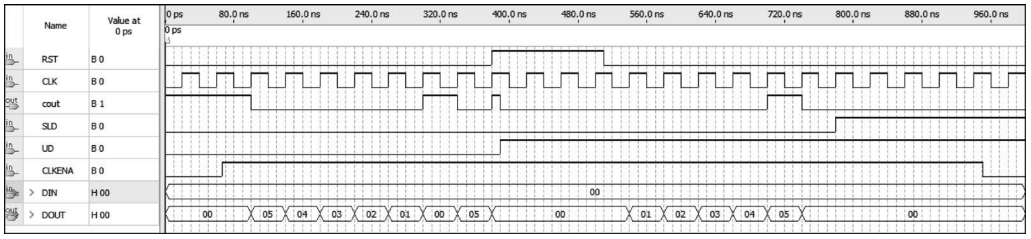


图 5-5　LPM COUNTER IP 核的仿真结果

从仿真结果可以看出，在时钟使能信号 clk_en 置为高电平后例化 LPM_COUNTER IP 核开始工作，此时计数控制信号 updown 为低电平，递减计数，并且计数模为 6，即计数从 5 开始，依次递减至 0；在异步复位信号置位(高电平)后暂停，解除置位后重新开始计数，在 updown 信号变为高电平后开始递增计数。在同步加载 sload 信号变成高电平后，计数器完成数据加载功能，输出端被加载成 data 数据。输出进位端 cout 只有在计数值为 0 时，输出高电平。

5.1.2　基于 ROM 的正弦波发生器

Intel 公司提供了在 FPGA 芯片内部生成内部存储器模块的宏功能模块。在 Quartus Ⅱ 软件中根据目标 FPGA 芯片、存储器模式和 RAM/ROM 的属性设置自动选择合适的宏功能模块。可实现的存储器模式包括单端口 RAM、双端口 RAM、单端口 ROM 和双端口 ROM。

用宏单元生成的 ROM、RAM 都存在于 FPGA 内部的 RAM 中，断电都会丢失。而用 IP 核生成的 ROM 只是提前添加了数据文件。在 FPGA 运行的时候，通过添加数据文件给 ROM 进行初始化，才使得生成的 ROM 模块像一个非易失性存储器，那么添加的文件是由软件产生的两种格式。分别是. hex 与. mif 文件，其中. mif 文件是由 Quartus 软件自己定义的一种文件。

1. 嵌入式存储器 IP LPM_ROM 的使用

1) 功能和特点

嵌入式存储器 IP 核的主要功能特点是：

(1) 存储器模式可配置，存储块类型可选。

(2) 支持读操作触发和写操作触发。

(3) 端口位宽、混合位宽端口(输入端口位宽和输出端口位宽不同)可配置，存储块深度

可配置。

(4) 支持时钟模式、时钟使能及地址时钟使能。

(5) 支持字节使能、写使能、写期间读控制。

(6) 提供可选的异步复位端口。

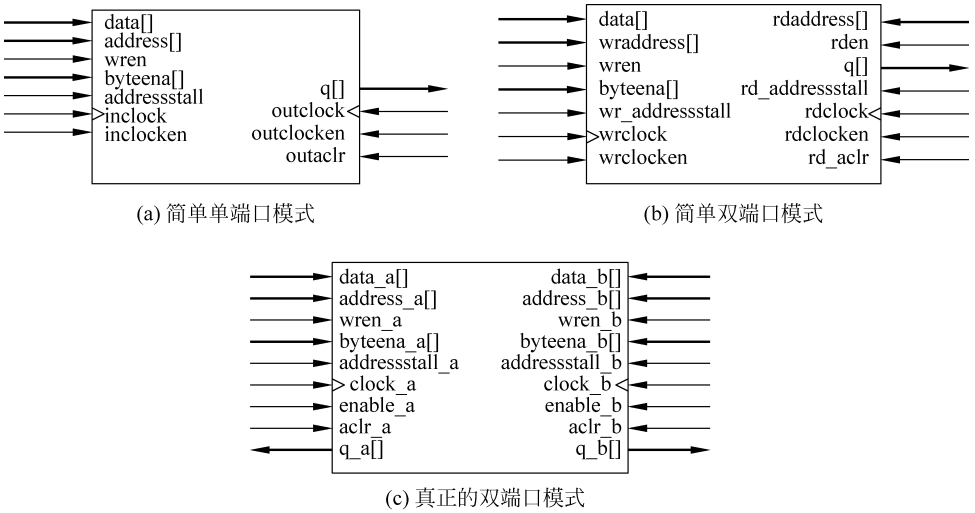
(7) 支持加电条件和存储器初始化。

(8) 支持纠错码。

(9) 支持的 FPGA 包括 Arria、Cyclone, HardCopy, MAX 和 Stratix 系列, 仅对于 MAX 系列 ROM 存储块不可用。

2) 接口信号说明

图 5-6 给出了嵌入式存储器 IP 核的接口信号描述, 分别对应于简单单端口模式、简单双端口模式和真正的双端口模式。所谓简单单端口即允许通过一个端口对存储进行读写访问, 不支持同时读写操作, 而简单双端口模式支持对存储同时进行读写访问。真正的双端口模式则有两个时钟(clock_a & clock_b)、两组输入输出数据线(data_a & data_b)、两组地址线(address_a & address_b)、两个使能端(enable_a & enable_b)、两个写使能端(wren_a & wren_b)。两个端口都可以进行读写操作(Port a 和 Port b 可以一起读或者一起写或者一个读一个写)。整体上, 读、写可以同时进行。



(a) 简单单端口模式

(b) 简单双端口模式

(c) 真正的双端口模式

图 5-6 嵌入式存储器 IP 核的接口信号

嵌入式存储器 IP 核的接口信号如表 5-4 所示。

表 5-4 嵌入式存储器 IP 核的接口信号

信号名	类型	必选/可选	描 述
data_a	输入	可选	存储器端口 A 的数据输入口 如果 operation_mode 被设置成以下任意取值时, 则 data_a 为必选信号: • SINGLE_PORT • DUAL_PORT • BIDIR_DUAL_PORT

续表

信号名	类型	必选/可选	描　　述
address_a	输入	必选	存储器端口 A 的地址输入口 在所有的操作模式下，都需要设置 address_a 信号
wren_a	输入	可选	端口 address_a 的写使能输入 如果 operation_mode 设置成以下任意取值时，则 wren_a 为必选信号 • SINGLE_PORT • DUAL_PORT • BIDIR_DUAL_PORT
rden_a	输入	可选	端口 address_a 的读使能输入口，读使能信号 rden_a 依赖于所选择的存储模式和存储块
byteena_a	输入	可选	字节使能输入端，可以屏蔽 data_a 端口，以便仅写入数据的特定字节、半字节或位 以下情况不支持 byteena_a 端口： • 如果 implement_in_les 参数设置为 ON • 如果 operation_mode 参数设置为 ROM
addressstall_a	输入	可选	地址时钟使能输入，只要 addressstall_a 端口为高电平，它使得地址保持在 address_a 端口的前一个地址
q_a	输出	必选	存储器的 A 端口的数据输出口 如果 operation_mode 被设置成如下任意取值，则 q_a 口为必选信号： • SINGLE_PORT • BIDIR_DUAL_PORT • ROM q_a 口的带宽必须等于 data_a 口的带宽
data_b	输入	可选	存储器的端口 B 的数据输入口 如果 operation_mode 参数被设置成 BIDIR_DUAL_PORT，则 data_b 口为必选信号
address_b	输入	可选	存储器的端口 B 的地址输入口 如果 operation_mode 参数被设置成下面取值之一，则 address_b 口为必选信号 • DUAL_PORT • BIDIR_DUAL_PORT
wren_b	输入	必选	端口 address_b 的写使能输入端 如果 operation_mode 被设置成 BIDIR_DUAL_PORT，则 wren_b 端为必选
rden_b	输入	可选	端口 address_b 的读使能输入端。rden_b 端是否被支持，依赖于所选用的存器模式和存储模块
byteena_b	输入	可选	字节使能输入端，可以屏蔽端口 data_b，以便仅写入数据的特定字节、半字节或位 以下情况不支持 byteena_b 端口： • 如果 implement_in_les 参数设置为 ON • 如果 operation_mode 参数设置为 SINGLE_PORT、DUAL_PORT、ROM
addressstall_b	输入	可选	地址时钟使能输入，只要 addressstall_b 端口为高电平，它使得地址保持在 address_b 端口的前一个地址

续表

信号名	类型	必选/可选	描　　述
q_b	输出	必选	存储器的 B 端口的数据输出口 如果 operation_mode 被设置成如下任意取值，则 q_b 口为必选信号： • DUAL_PORT • BIDIR_DUAL_PORT q_b 口的带宽必须等于 data_b 口的带宽
clock0	输入	必选	以下描述了哪些内存时钟必须连接到 clock0 端口，以及不同时钟模式下的端口同步： • 单时钟：将单源时钟连接到 clock0 端口。所有被寄存的端口都由相同的源时钟同步 • 读/写：将写时钟连接到 clock0 端口。所有与写操作相关的被寄存的端口，如 data_a 端口、address_a 端口、wren_a 端口和 byteena_a 端口由写时钟同步 • 输入输出：将输入时钟连接到 clock0 端口。所有被寄存的输入端口都由输入时钟同步 • 独立时钟：将端口 A 时钟连接到 clock0 端口。端口 A 的所有被寄存的输入和输出端口都由端口 A 时钟同步
clock1	输入	可选	以下描述了哪些内存时钟必须连接到 clock1 端口，以及不同时钟模式下的端口同步： • 单时钟：不适用。所有被寄存的端口都由 clock0 端口同步 • 读/写：将读时钟连接到 clock1 端口。与读操作相关的所有被寄存的端口，如 address_b 端口、rden_b 端口、和 q_b 端口由读时钟同步 • 输入输出：将输出时钟连接到 clock1 端口。所有被寄存的输出端口都由输出时钟同步 • 独立时钟：将端口 B 时钟连接到 clock1 端口。端口 B 的所有被寄存的输入和输出端口都由端口 B 时钟同步
clocken0	输入	可选	clock0 端口的时钟使能输入
clocken1	输入	可选	clock1 端口的时钟使能输入
clocken2	输入	可选	clock0 端口的时钟使能输入
clocken3	输入	可选	clock1 端口的时钟使能输入
aclr0 aclr1	输入	可选	异步清除寄存的输入和输出端口。aclr0 端口影响由时钟 clock0 控制的寄存端口，而 aclr1 端口影响由时钟 clock1 控制的寄存端口 对寄存端口的异步清零效果可以通过其对应的异步清零参数来控制，如 outdata_aclr_a，address_aclr_a 等
eccstatus	输出	可选	一个 3 位宽的纠错状态端口。指示从内存中读取的数据是否有单比特错误并被纠正，或者有致命错误没有纠正，或者没有错误位发生 在 Stratix V 器件中，M20K ECC 状态用两位宽的纠错状态端口来通信。M20K ECC 检测并修复单个比特错误事件、双相邻错误事件，或检测三个相邻错误而不修复错误 如果满足以下所有条件，则支持 eccstatus 端口： • 操作模式 operation_mode 参数设置为 DUAL_PORT • ram_block_type 参数设置为 M144K 或者 M20K • width_a 和 width_b 参数具有相同的值 • 未使用字节使能
data	输入	必选	存储器的数据输入，数据端口是必选的，并且其宽度必须等于 q 端口的宽度

续表

信号名	类型	必选/可选	描　　述
wraddress	输入	必选	存储器的写地址输入，wraddress 端口是必选的，并且其宽度必须等于 rdaddress 端口宽度
wren	输入	必选	wraddress 端口的写使能输入，端口 wren 是必选的端口
rdaddress	输入	必选	存储器的读地址输入端，rdaddress 端口是必选的，并且其宽度必须等于 wraddress 端口的宽度
rden	输入	可选	rdaddress 端口的读使能输入端。当 use_eab 参数被设置成 OFF 时，支持 rden 端口。当 ram_block_type 参数被设置成 MLAB 时，不支持 rden 端口
byteena	输入	可选	字节使能输入端，可以屏蔽数据端口，以便仅写入数据的特定字节、半字节或某些位，当 use_eab 参数被设置成 OFF 时，不支持 byteena 端口。在 Arria Ⅱ GX 以及更新的器件中，如果 ram_block_type 参数被设置成 MLAB，则支持 byteena 端口
wraddressstall	输入	可选	写地址时钟使能输入，只要 wraddressstall 是高电平时，写地址时钟使能输入能够保持 wraddress 端口的前一个写地址
rdaddressstall	输入	可选	读地址时钟使能输入，只要 rdaddressstall 是高电平时，读地址时钟使能输入能够保持 rdaddress 端口的前一个读地址。在较新的器件中，当 rdaddress_reg 被设置成 UNREGISTERED 时，不支持 rdaddressstall 端口
q	输出	必选	存储器的数据输出端。q 端口是必选的，并且必须等于数据 data 端口的宽度
inclock	输入	必选	以下描述了哪些内存时钟必须连接到 inclock 端口，以及不同时钟模式下的端口同步方式： • 单时钟：将单源时钟连接到 inclock 端口和 outclock 端口。所有寄存端口都由相同的源时钟同步 • 读/写：将写时钟连接到 inclock 端口。所有与写操作相关的寄存端口，如 data 端口、wraddress 端口、wren 端口和 byteena 端口都由写时钟同步 • 输入/输出：将输入时钟连接到 inclock 端口。所有寄存输入端口都由输入时钟同步
outclock	输入	必选	以下描述了哪些内存时钟必须连接到 outclock 端口，以及不同时钟模式下的端口同步方式： • 单时钟：将单源时钟连接到 inclock 端口和 outclock 端口。所有寄存端口都由相同的源时钟同步 • 读/写：将读时钟连接到 outclock 端口。所有与读操作相关的寄存端口，如 rdaddress 端口、rdren 端口和 q 端口都由读时钟同步 • 输入/输出：将输出时钟连接到 outclock 端口。所有寄存的 q 端口都由输出时钟同步
inclocken	输入	可选	inclock 端口的时钟使能输入端
outclocken	输入	可选	outclock 端口的时钟使能输入端
aclr	输入	可选	异步清除寄存的输入和输出端口。aclr0 端口影响由时钟 clock0 控制的寄存端口，而 aclr1 端口影响由时钟 clock1 控制的寄存端口。 对寄存端口的异步清零效果可以通过其对应的异步清零参数来控制，如 indata_aclr、wraddress_aclr 等

3）参数设置-ROM：1-PORT

这里仅以-ROM：1-PORT 为例，给出了相关的参数设置，如表 5-5 所示。其他情况请

读者自行参考相关手册信息。

表 5-5　ROM：1-PORT 的参数

参　数		合法取值	描　述
参数设置：general 页			
How wide should the ‘q’ output bus be?			规定了输出总线“q”的带宽
How many ＜X＞-bit words of memory?			规定了存储器中字 words 的位数＜X＞
What should the memory block type be?			规定了存储器块的类型，可选的存储器块类型取决于目标器件的种类
Set the maximum block depth to		Auto，32，64，128，256，512，1024，2048，4096	规定了存储器块的最大存储深度，以字数表示
What clocking method would you like to use?			规定了所使用时钟的方法 • 单时钟：单时钟和时钟使能控制了存储器块中的所有寄存器 • 双时钟（输入时钟和输出时钟）：输入时钟控制地址寄存器，输出时钟控制数据输出寄存器，在 ROM 模式下没有写使能，字节使能和数据输入寄存器
参数设置：Regs/Clken/Aclrs 页			
Which ports should be registered? ‘q’ output port		On/Off	规定了是否寄存输出端口“q”
Create one clock enable signal for each clock signal. Note：All registered ports are controlled by the enable signal(s)		On/Off	规定了是否为每个时钟信号创建一个时钟使能信号
MORE OPTION	Use clock enable for port A input registers	On/Off	规定了对于端口 A 的输入寄存器是否使用时钟使能信号
	Use clock enable for port A output registers	On/Off	规定了对于端口 A 的输出寄存器是否使用时钟使能信号
	Create an ‘addressstall_a’ input port。	On/Off	规定了是否创建“addressstall_a”输入端口 可以创建此端口作为地址寄存器的附加低电平有效时钟使能输入
Create an ‘aclr’ asynchronous clear for the registered ports。		On/Off	指定是否为寄存端口创建异步清除端口
More option	‘address’ port	On/Off	规定是否“address”端口被“aclr”端影响
	‘q’ port	On/Off	规定是否“q”端口被“aclr”端影响
Create a ‘rden’ read enable signal		On/Off	规定是否创建一个读使能信号
参数设置：Mem Init 页			
Do you want to specify the initial content of the memory?		Yes，use this file for the memory content data	规定了存储器的初始内容 在 ROM 模式下，必须指定内存初始化文件（‘.mif’）或十六进制（Intel 格式）文件（‘.hex’）。Yes，use this file for the memory content data 选项默认是打开的

续表

参 数	合法取值	描 述
Allow In-System Memory Content Editor to capture and update content independently of the system clock	On/Off	指定是否允许 In-System Memory Content Editor 独立于系统时钟捕获和更新内容
The 'Instance ID' of this ROM is	—	规定了存储器的识别码 ID

4）调用生成元件模型

下面的程序是利用 MegaWizard 工具调用和配置嵌入式存储器 IP ROM：1-PORT 之后，由 Quartus 自动生成的元件模型。其中 datarom 是使用 MegaWizard 工具调用宏模块时自行命名的生成元件模型名称。altsyncram 是调用的宏模块元件名。altsyncram_component 是宏模块元件例化名。

【代码 5-3】

```
LIBRARY IEEE;
USE ieee.STD_LOGIC_1164.ALL;
LIBRARY ALTERA_MF; -- Altera 宏模块仿真库包括 IP 核包 altera_mf_components
USE ALTERA_MF.all;
ENTITY datarom IS
PORT
(
    address                 : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
    clock                   : IN STD_LOGIC  := '1';
    q                       : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
);
END datarom;
ARCHITECTURE SYN OF datarom IS
SIGNAL sub_wire0            : STD_LOGIC_VECTOR (7 DOWNTO 0);
COMPONENT altsyncram
GENERIC (
    address_aclr_a          : STRING;
    clock_enable_input_a    : STRING;
    clock_enable_output_a   : STRING;
    init_file               : STRING;
    intended_device_family  : STRING;
    lpm_hint                : STRING;
    lpm_type                : STRING;
    numwords_a              : NATURAL;
    operation_mode          : STRING;
    outdata_aclr_a          : STRING;
    outdata_reg_a           : STRING;
    widthad_a               : NATURAL;
    width_a                 : NATURAL;
    width_byteena_a         : NATURAL
);
PORT (
        address_a           : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
        clock0              : IN STD_LOGIC ;
        q_a                 : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
);
END COMPONENT;
```

```
    BEGIN
    q    <= sub_wire0(7 DOWNTO 0);
    altsyncram_component : altsyncram
    GENERIC MAP (
        address_aclr_a => "NONE",
        clock_enable_input_a => "BYPASS",
        clock_enable_output_a => "BYPASS",
        init_file => "SINdata.mif",
        intended_device_family => "Cyclone III",
        lpm_hint => "ENABLE_RUNTIME_MOD=NO",
        lpm_type => "altsyncram",
        numwords_a => 512,
        operation_mode => "ROM",
        outdata_aclr_a => "NONE",
        outdata_reg_a => "CLOCK0",
        widthad_a => 9,
        width_a => 8,
        width_byteena_a => 1
    )
    PORT MAP (
        address_a => address,
        clock0 => clock,
        q_a => sub_wire0
    );
END SYN;
```

2. 基于 ROM 调用的正弦波发生器

正弦波发生器的基本设计思想：用一个存储器 ROM 来存放波形数据，这里是正弦波的波形数据，亦可存储三角波、锯齿波等数据，存储器存放的数据就决定了波形发生器所生成的波形。另外，用一个计数器来产生存储器的地址，存储器 ROM 根据计数器所产生的地址，输出对应存储单元里的数据。随着输入时钟的推进，计数器产生地址依次增加，从而 ROM 依次输出各个存储单元的数据，即向外发生正弦波。

下面提供两种实现方案：方案一，分别用 LPM_ROM 和 LPM_COUNTER 两个宏功能模块实现数据存储器和计数器，在顶层原理图中进行简单的连接，如图 5-7 所示，这种实现方案简单易行，不需要编写 VHDL 程序。方案二，则调用 LPM_ROM 宏功能模块实现数据存储器，计数器功能在顶层程序中用 VHDL 编写，而数据存储器则在顶层程序中被当作一个元件 COMPONENT 去使用。方案二提供了一种将宏功能模块混合在 VHDL 语言中的使用方法，其具体的顶层程序如代码 5-4 所示。

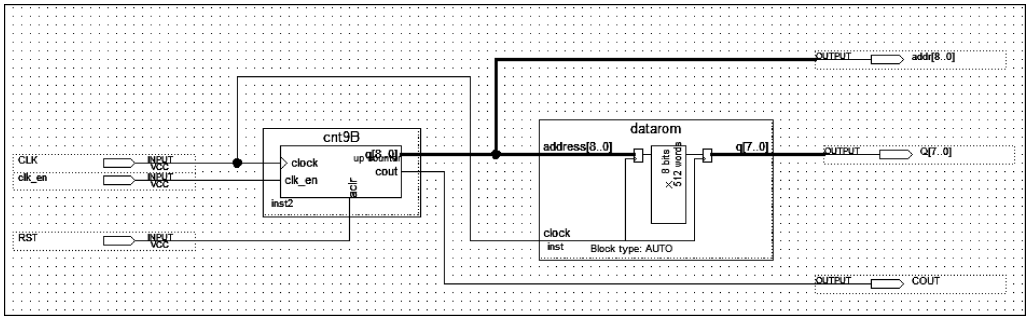


图 5-7　正弦信号发生器电路原理图

【代码 5-4】

```
LIBRARY IEEE;                                          -- 正弦信号发生器源文件
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY SINGT IS
PORT ( CLK : IN STD_LOGIC;                             -- 信号源时钟
clk_en : IN STD_LOGIC;
RST:IN STD_LOGIC;
COUT: OUT STD_LOGIC;
DOUT : OUT STD_LOGIC_VECTOR (7 DOWNTO 0) );            -- 8 位波形数据输出
END;
ARCHITECTURE DACC OF SINGT IS
COMPONENT datarom -- 调用波形数据存储器 LPM_ROM 文件: datarom.vhd 声明
PORT(address : IN STD_LOGIC_VECTOR (8 DOWNTO 0);       -- 9 位地址信号
clock : IN STD_LOGIC : = '1';                          -- 地址锁存时钟
q : OUT STD_LOGIC_VECTOR (7 DOWNTO 0) );
END COMPONENT;
SIGNAL Q1 : STD_LOGIC_VECTOR (8 DOWNTO 0);             -- 设定内部节点作为地址计数器
BEGIN
PROCESS(CLK )                                          -- LPM_ROM 地址发生器进程
BEGIN
IF RST = '1' THEN
   Q1 < = "000000000";
ELSIF clk_en = '1'  THEN
   IF
      CLK'EVENT AND CLK  =  '1' THEN Q1 < = Q1 + 1;    -- Q1 作为地址发生器计数器
   END IF;
END IF;
END PROCESS;
PROCESS(Q1)
BEGIN
IF Q1 = "111111111" THEN
   COUT < = '1';
ELSE
   COUT < = '0';
END IF;
END PROCESS;
   u1 : datarom PORT MAP(address = > Q1, q  = > DOUT,clock = > CLK);      -- 例化
END;
```

存储器 ROM 中存储的初始化数据可以存成两种文件格式：.mif 或者.hex，如图 5-8 所示。最基本的方法是新建这两个格式的文件，然后手工编辑各个存储单元的数据。其中.mif 格式文件也可以利用专用.mif 文件生成器——MifMaker2010 来生成。还可以利用其他一些高级语言，或者单片机相关软件工具来生成数据文件。

仿真结果如图 5-9 所示。当时钟使能信号 clk_en 为有效高电平后，在时钟 clk 的作用下只要复位 RST 为低电平，不同地址单元 addr 中的数据从 Q 端输出。

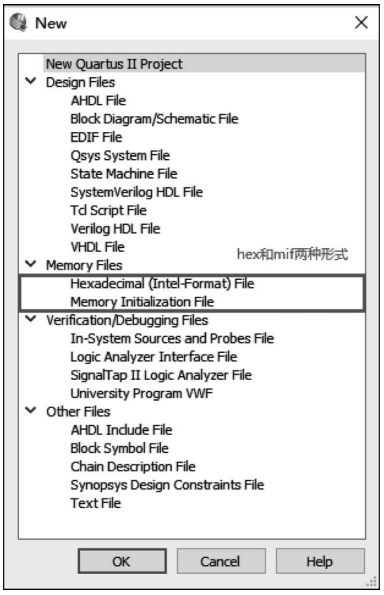


图 5-8　ROM 初始化数据的文件选择

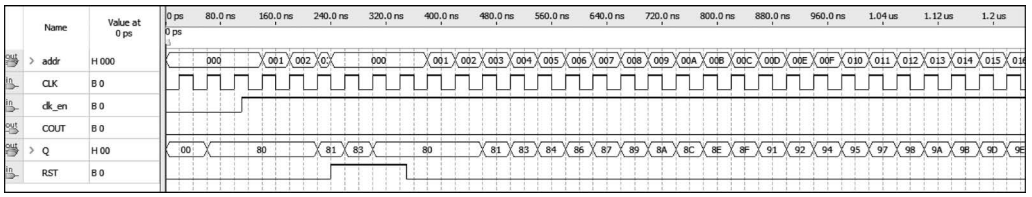


图 5-9　正弦波发生器的仿真波形

5.2　IP 核的使用实例

5.2.1　IP 相关常识概述

1. IP 的概念

IP 设计是目前 FPGA 设计的主流方法之一，是在 SoC 中的集成方式及应用场景。芯片设计中的 IP 核具有特定功能的可复用的标准性和可交易性，已经成为集成电路设计技术的核心与精华。

IP 核设计电路的特点是：

(1) 具有相当的灵活度。

(2) AI 算法推动 IP 核研发加速。

(3) IP 验证贯穿于整个设计流程。

(4) IP 核已经变成系统设计的基本单元，独立设计成果被交换、转让和销售。

芯片行业中所说的 IP 核是指芯片中具有独立功能的电路模块的成熟设计。该电路模块设计可以应用在包含该电路模块的其他芯片设计项目中，从而减少设计工作量，缩短设计周期，提高芯片设计的成功率。该电路模块的成熟设计凝聚着设计者的智慧，体现了设计者

的知识产权，因此，芯片行业就用 IP 核(Intellectual Property Core)来表示这种电路模块的成熟设计。IP 核也可以理解为芯片设计的中间构件。

一般说来，一个复杂的芯片是由芯片设计者自主设计的电路部分和多个外购的 IP 核连接构成的。要设计这样结构的一款芯片，设计公司可以外购芯片中所有的 IP 核，仅设计芯片中自己有创意的、自主设计的部分，并把各部分连接起来。

可见芯片设计过程就像系统电路板开发过程一样，是用已有的、成熟的 IP 核(或者芯片)进行布局、摆放和信号连接的过程，这种过程可以称为对 IP 核(或者芯片)的复用。不同的是，系统电路板上除了芯片和连接线之外，系统开发者很少自主开发自己的芯片。而在芯片设计过程中，芯片上除了采用外购的 IP 核之外，一般说来，芯片设计者还要设计一部分自己的电路，并完成各部分之间的信号连线，最后还要对整个芯片的功能、性能进行制造前的反复检查和验证。

如果以上介绍还显得太过专业，还可以用拼图画来对芯片设计打比方，可以把芯片抽象地理解成拼图画。复杂芯片的设计过程就像要拼好这幅图画一样，用现有的图块(IP 核)拼接美丽图画(复杂芯片)。同时，芯片设计要考量 IP 核的许多参数和指标，并要把各个 IP 核和自主设计部分正确连接，保证整个芯片的功能和性能正确无误。

IP 核被其他芯片设计公司采用，行业内称为 IP 复用。专门设计相对独立的电路功能模块，目的是推广给其他芯片设计公司进行复用，这种设计工作称为 IP 开发。专门从事 IP 开发的公司称为 IP 厂商或者 IP 提供商。IP 厂商把 IP 销售给芯片设计公司是一种 IP 交易行为。

2. IP 的由来和作用

IP 的由来要从早期的芯片设计过程讲起。早期芯片的集成规模有限，设计复杂度不高，芯片上所有的电路都是由芯片设计者自主完成的。设计水平不高、能力有限的芯片公司只能设计规模小的简单的芯片。设计水平高、能力强的芯片公司才可以设计规模大、功能复杂的芯片。这个时期，不论芯片规模大还是小，芯片从“头”到“脚”都是由芯片公司自己设计的。早期的高端芯片基本上都是由为数不多的大型国际芯片公司把持的。

随着现代信息社会对芯片要求提升，芯片的规模呈指数性增加，复杂性急剧增大。中小型芯片公司要独立完成一款复杂芯片设计几乎变得不太可能。特别是 20 世纪 80 年代末，芯片行业出现了晶圆代工(Foundry)商业模式，大批的中小微芯片设计公司(Fabless)应运而生。这个时期，芯片设计行业急需解决小芯片公司无法设计大芯片的难题。

解决这一难题的启发思路很多。例如：搭积木和拼图画玩具；由标准件设计大型机器；由软件子程序(或者中间件)调用设计大型软件；用芯片搭建大型电子系统等。思路都是重复使用预先设计好的成熟的构件来搭建更复杂的系统，省掉对构件内部问题的考虑，化繁为简；重复使用构件，减少重复劳动，节省时间；重复使用构件，提高整个复杂系统搭建的成功率。

芯片设计行业中的 IP 核开发和 IP 复用，就是在这些思路启发下形成的。IP 核就类似于上述的构件。IP 核是预先设计好的具有独立功能的电路模块设计。有了 IP 核这种构件，大的复杂的芯片设计就变得较容易、周期短、易成功。

IP 的作用主要有四个方面，一是使芯片设计化繁为简，缩短芯片设计周期，提高复杂芯片设计的成功率；二是 IP 开发和 IP 复用技术使小公司设计大芯片成为可能；三是使系统

整机企业可以设计自己的芯片，提升自主创新能力和整机系统的自主知识产权含量；四是使芯片设计行业摆脱传统 IDM 模式，成为产业链上独立的行业，促进了芯片设计业迅猛发展。

目前，许多中小微芯片设计公司虽然设计能力和水平有限，但出于抢占市场，缩短芯片设计周期的需要，会外购许多 IP 核来完成自己的芯片设计项目。业界的 IP 开发商、IP 提供商数量不断增加，也变得越来越专业。各种功能、各种类型的 IP 核不断涌现。IP 交易活动也日趋普遍，交易金额也越来越大。

3. IP 核的种类和举例

IP(Intelligent Property)核是具有知识产权核的集成电路核总成，是经过反复验证过的，具有特定功能的宏模块，与芯片制造工艺无关，可以移植到不同的半导体工艺中。IP 核模块有行为(Behavior)、结构(Structure)和物理(Physical)三级不同程度的设计，对应描述功能行为的不同分为三类，即软核(Soft IP core)、完成结构描述的固核(Firm IP core)和基于物理描述并经过工艺验证的硬核(Hard IP core)。这相当于集成电路(器件或部件)的毛坯、半成品和成品的设计技术。

(1) 软核：它是用硬件描述语言(HDL)设计的独立功能的电路模块。从芯片设计程度来看，它只经过了 RTL 级设计优化和功能验证，通常是以 HDL 文本形式提交给用户。所以它不包含任何物理实现信息，因此，软核与制造工艺无关。软核相当于软件编程的库。软核的设计周期短，设计投入少。由于不涉及物理实现，为后续设计留有很大的发挥空间，增大了 IP 的灵活性和适应性。

用户购买了 IP 软核后，可以自行综合出正确的门电路级设计网表，并可以进行后续的结构设计，具有很大的灵活性。借助于 EDA 综合工具，用户可以很容易与其他 IP 软核以及自主设计的电路部分合成一体，并根据各种不同半导体工艺，设计成具有不同性能的芯片。大多数应用于 FPGA 的 IP 核均为软核，软核有助于用户调节参数并增强可复用性。

(2) 固核：它的设计程度介于软核和硬核之间，是软核和硬核的折中。它除了完成软核所有的设计外，还完成了门级电路综合和时序仿真等设计环节。一般地，它以门级电路网表的形式提供给用户。

(3) 硬核：它提供了电路设计最后阶段掩模级的电路模块。它以最终完成的布局布线网表形式提供给用户。硬核既具有结果的可预见性，也可以针对特定工艺或特定 IP 提供商进行功耗和尺寸的优化。

所以，三种类型的 IP 核是电路功能模块设计在不同设计阶段的产物，如图 5-10 所示。

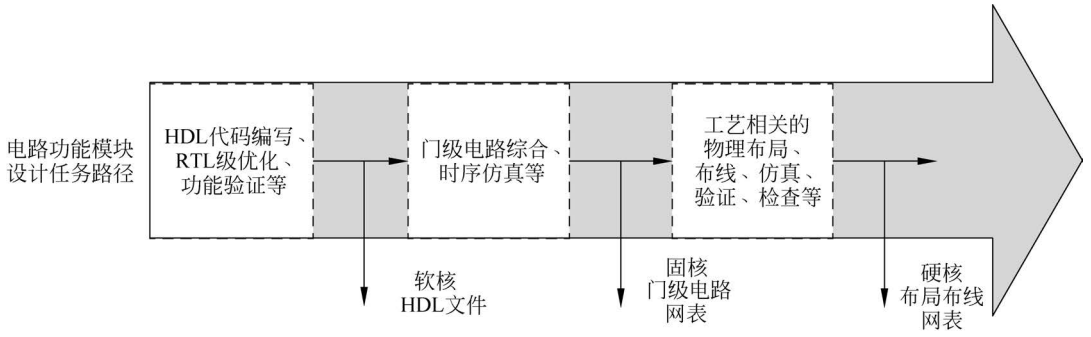


图 5-10　在电路功能模块设计的不同阶段，可得到不同类型的 IP 核

用户经过精心评测和选择，购买了 IP 厂商的 IP 核后，开始设计自己的芯片。前文讲过，一个复杂芯片一般由购买的 IP 核和用户自主设计的电路部分组成。芯片设计过程包括了行为级、结构级和物理级三个阶段。行为级和结构级设计阶段的工作一般称为前端设计，物理级设计阶段的工作一般称为后端设计。图 5-11 的示意图说明，不同类型的 IP 核是在不同的设计阶段中加入整个芯片设计中去的。

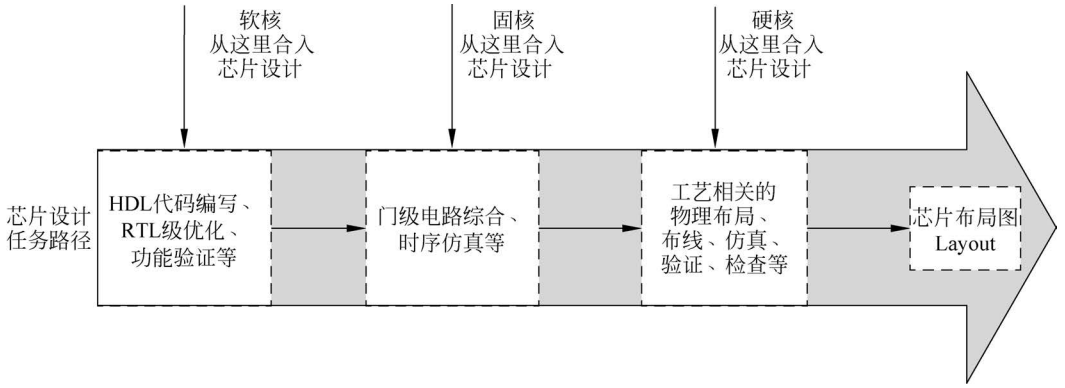


图 5-11　在设计的不同阶段集合 IP 核

三种类型的 IP 核各有优缺点，用户会根据自己的实际需要来选择。以下是三种 IP 核的优缺点简要总结。

软核：它以综合源代码的形式交付给用户，其优点是源代码灵活，在功能一级可以重新配置，可以灵活选择目标制造工艺。灵活性高、可移植性强，允许用户自配置。其缺点是对电路功能模块的预测性较差，在后续设计中存在发生错误的可能性，有一定的设计风险；在一定程度上使后续工序无法适应整体设计，从而需要一定程度的软 IP 修正，在性能上也不可能获得全面的优化。用户可以综合出正确的门电路级网表，并可以进行后续结构设计，具有最大的灵活性。这使得软核的知识产权保护难度较大（如：调用一个 PLL 的 IP 核，通过修改代码参数可以实现不同频率的倍频）。同时，如果后续设计不当，有可能导致整个结果失败。软核又称作虚拟器件。

固核：它的灵活性和成功率介于软核和硬核之间，是一种折中的类型。和软核相比，固核的设计灵活性稍差，但在可靠性上有较大提高。目前，固核是 IP 核的主流形式之一。对于那些对时序要求严格的内核（如 PCI 接口内核），可预测布线特定信号或分配特定的布线资源，以满足时序要求。这些内核可归类为固核，由于内核是预先设计的代码模块，因此这有可能影响包含该内核的整体设计。由于内核的建立（Setup）、保持（Hold）时间和握手信号都可能是固定的，因此其他电路的设计都必须考虑与该内核进行正确地接口。如果内核具有固定布局或部分固定的布局，那么这还将影响其他电路的布局。

硬核：它的最大优点是确保性能，如速度、功耗等达到预期效果。然而，硬核与制造工艺相关，难以转移到新的工艺或者集成到新的结构中去，是不可以重新配置的。硬核不许修改的特点使其复用有一定的困难，因此只能用于某些特定应用，使用范围较窄。尽管硬核由于缺乏灵活性而可移植性差，但由于无须提供寄存器转移级（RTL）文件，因而更易于实现 IP 保护，即硬核的知识产权保护最为方便。

IP 核的举例，最典型有 ARM 公司的各种类型的 CPU IP 核。许多 IP 供应商提供的

DSP IP 核、USB IP 核、PCI-X IP 核、Wi-Fi IP 核、以太网 IP 核、嵌入式存储器 IP 核等，五花八门，品种繁多。

如果按大类分，大体上可分为处理器和微控制器类 IP 核、存储器类 IP 核、外设及接口类 IP 核、模拟和混合电路类 IP 核、通信类 IP 核、图像和媒体类 IP 核等。

Intel 公司的 IP 核分两种：

一种是免费的 IP 核，不需要另外的授权(License)，就是所谓的基本函数的 IP 核，例如浮点运算、普通运算、三角函数、基本的存储器 IP 核、配置功能 IP 核、PLL、所有的桥以及所有的 FPGA 内部的硬核即 NiosII(不含源码)等。

另外一种是收费的 IP 核，需要购买单独的 IP 核的授权(License)，例如各种以太网软 IP 核、PCI-E 软 IP 核、CPRI、Interlaken 协议、PCI、RapidIO 和所有的几十个视频图像 IP 核以及所有的 DDR1/2/3/4 软 IP 核、256 位 AES 硬件加密等。

4. OpenCore Plus IP 核评估

免费 OpenCore Plus 功能允许用户购买之前在仿真和硬件中对获得授权的 MegaCore IP 核进行评估。如果用户决定将自己的设计投入生产，请购买 MegaCore IP 核许可证。OpenCore Plus 支持如下评估：

(1) 在用户的系统中仿真已授权 IP 核的行为。

(2) 快速又简易地验证 IP 核的功能性、大小和速度。

(3) 为包含 IP 核的设计生成有限时的器件编程文件。

(4) 使用自己的 IP 核对器件进行编程并验证自己的硬件设计。

OpenCore Plus 评估支持如下两种操作模式：

(1) Untethered——有限时间内运行包含已授权 IP 核的设计。

(2) Tethered——长期或无限期运行包含已授权 IP 核的设计。此操作需要连接电路板和主机。

注：如果设计中的任何 IP 核超时，则使用 OpenCore Plus 的所有 IP 核同时超时。

5.2.2 8B/10B 核的使用

8B/10B 编译码器主要用于千兆位以太网、光纤信道及其他应用的物理层编码。8B/10B 编码器以字节作为输入，生成直流(DC)平衡数据流(数据中的 0 和 1 个数相等)，最大长度为 5。也有一些独特的 10 比特码中 0 和 1 的个数接近：例如 4 个 0 和 6 个 1，或者 6 个 0 和 4 个 l。对于这种情况，0 和 1 个数之差作为下一个 10 比特码生成的输入，从而在数据流中保持总的 0 和 1 个数平衡。为此，某 8 比特输入会根据输入不一致性而得到 2 个有效的 10 比特码。

Intel 公司 8B/10B 编码器/译码器能够生成满足上述要求的编码器和译码器。

1. 功能和特点

Intel 公司 8B/10B 编码器/译码器 IP 核的主要功能特点是：

(1) 支持 8B/10B 编码和译码。

(2) 可以级联编码和译码。

(3) 支持与工业标准兼容的特殊字符编码。

(4) 易用的 IP 向导简单接口。

（5）提供 Intel 公司支持的、在 VHDL 和 Verilog HDL 仿真器中使用的功能仿真模型。

（6）支持 Arria GX、Arria Ⅱ GX，Cyclone，Cyclone Ⅱ，Cyclone Ⅲ，Cyclone Ⅲ LS，Cyclone Ⅳ（E 和 GX），HardCopy Ⅱ HardCopy Ⅲ，HardCopy Ⅳ（E 和 GX），Stratix，Stratix GX，Stratix Ⅱ Stratix Ⅱ GX，Stratix Ⅲ和 Stratix Ⅳ系列 FPGA。

2. 接口信号说明

8B/10B 编/译码器 IP 核的接口信号描述如图 5-12 所示。

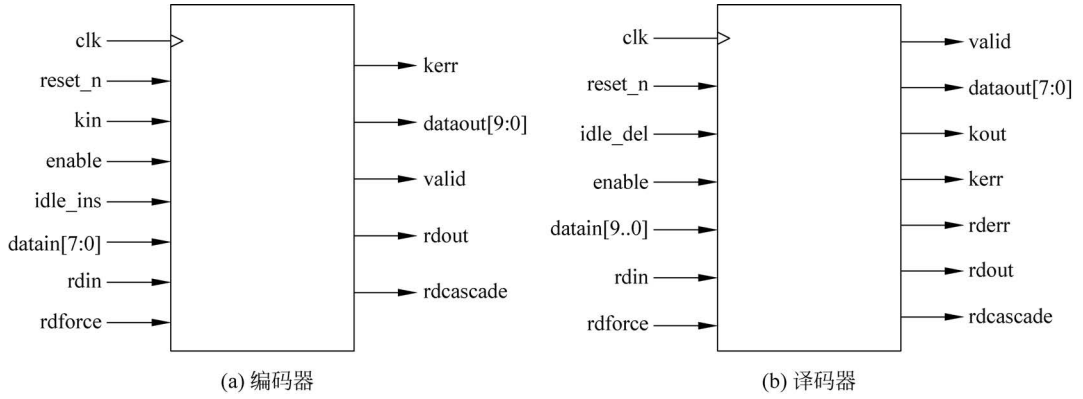


图 5-12　8B/10B 编/译码器 IP 核的接口信号

表 5-6 和表 5-7 分别给出了 8B/10B 编/译码器 IP 核的接口信号说明。

表 5-6　8B/10B 编码器 IP 核的接口信号说明

信号名称	信号方向	说　明
clk	I	系统时钟，带锁存，输入和输出之间有 3 个时钟周期的延时
reset_n	I	低电平有效的异步复位信号，用于复位 IP 核的所有寄存器，须由 clk 的上升沿同步清除
kin	I	命令字节指示，为高电平时表示输入的是命令字节而非数据字节
enable(ena)	I	编码器使能，高电平有效，表示对输入端口 datain 上的当前数据进行编码
idle_ins	I	空闲字符插入，高电平且当 ena 为低电平时插入空闲字符 K28.5
datain[7：0]	I	数据输入，可以是 8 位的字、数据或命令
rdin	I	运行不一致 RD 输入，当 rdforce 为高电平时，该端口的值代替内部生成的运行不一致 RD 值，作为当前的运行不一致 RD 值
rdforce	I	强制运行不一致 RD，该引脚为高电平时，rdin 值覆盖内部生成的运行不一致取值
kerr	O	特殊控制符 K 码错误，当 ena 和 kin 均为高电平且 datain 的值不是有效特殊 K 字符时，该信号置为高电平
dataout[9：0]	O	数据输出，是 10 位的编码输出
valid	O	有效标记信号，高电平时表示在 dataout 端口上出现有效编码字
rdout	O	运行不一致输出，在编码字出现在 dataout 输出之后的当前运行不一致 RD 值
rdcascade	O	级联的运行不一致，仅在级联编码器时使用

表 5-7　8B/10B 译码器 IP 核的接口信号说明

信号名称	信号方向	说　明
clk	I	系统时钟，带锁存，输入和输出之间有 2 个时钟周期的延时
reset_n	I	低电平有效的异步复位信号，用于复位 IP 核的所有寄存器，须由 clk 的上升沿同步清除
enable(ena)	I	译码器使能，高电平有效，表示对输入端口 datain 上的当前数据进行译码
idle_del	I	空闲删除信号，高电平时从数据流中移除空闲字符 K28.5
datain[9：0]	I	数据输入，10 位已编码的输入字
rdin	I	运行不一致 RD 输入，当 rdforce 为高电平时，该端口的值代替内部生成的运行不一致 RD 值，作为当前的运行不一致 RD 值
rdforce	I	强制运行不一致 RD，该引脚为高电平时，rdin 值覆盖内部生成的运行不一致取值
dataout[7：0]	O	数据输出，是 8 位的译码数据或命令
valid	O	有效标记信号，高电平时表示在 dataout 端口上出现有效译码字
kout	O	命令输出，高电平时表示输出的是命令字节而非数据字节
kerr	O	特殊控制符 K 码错误，当接收到无效 10 位字或 10 字节 ERR 字符时，该信号置为高电平
rderr	O	运行不一致错误，为高电平时表示已经违背运行时不一致原则
rdout	O	运行不一致输出，在译码字出现在 dataout 输出之后的当前运动不一致 RD 值
rdcascade	O	级联的运行不一致，仅在级联译码器时使用

8B/10B 编/译码器的设计参数有两个：一个是操作模式，可以选择实现编码器或者译码器；另一个是选择是否寄存输入/输出，该参数只对编码器生效，如果选择寄存，则为 3 个时钟周期的延时，否则是 1 个时钟周期的延时。

3. 功能描述和信号时序分析

Intel 公司 8B/10B 编译码器 IP 核包括一个编码器(ENC 8B/10B)和一个译码器(DEC 8B/10B)IP 核。编码器将输入的 8 位字节编码成 10 位传输码字，译码器接收 10 位码字进行译码得到 8 位字节数据，图 5-13 给出了双向转换过程。

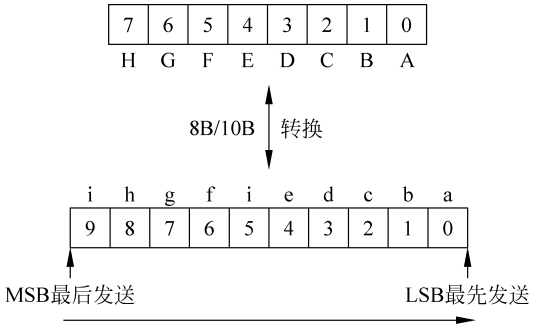


图 5-13　8B/10B 编译码器数据转换过程

假设原始 8 位数据从高到低用 HGFEDCBA 表示，8B/10B 编码将 8 位数据分成高 3 位 HGF 和低 5 位 EDCBA 两个子组。然后经过 5B/6B 编码，将低 5 位 EDCBA 映射成 abcdei；高 3 位经过 3B/4B 编码，映射成 fghj，最后合成 abcdeifghj 发送。发送时是由最低位 LSB，a 先发送。其具体的编码原理如图 5-14 所示。

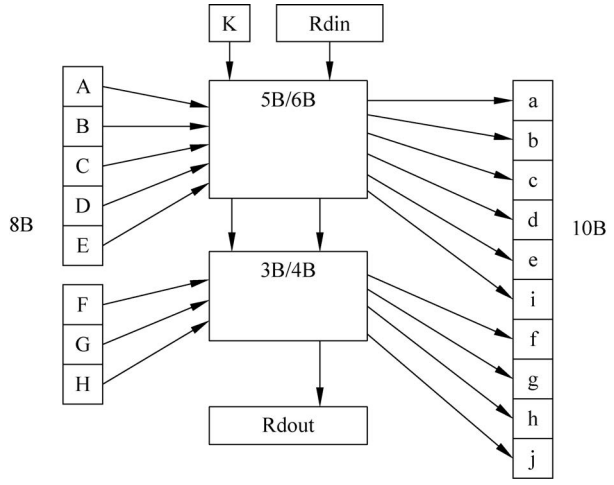


图 5-14　8B/10B 编码原理

通常，认为会将低 5 位 EDCBA 按其十进制数值记为 x，将高 3 位按其十进制数值记为 y，将原始 8 位数据记为 D. x. y。例如 8 位数“101 10101”，即十进制数 181，按照上述划分原则 x＝10101(21)，y＝101(5)，所示这个数被表示为 D. 21. 5。此外，在进行传输时，除了传输数据本身，还需要嵌入一些控制信号。因此，在 8B/10B 编码中，还需用到 12 种控制字符，用来标识传输数据的开始和结束、传输空闲等状态，按照上述规则，将控制字符记为 K. x. y。

8 位原始数据对应 256 个码，加上 12 种控制字符，而编码后的 10 位数据有 1024 个码，肯定有很多是用不到的，故需选择其中一部分来表示 8 位数据，所选的码字 0 和 1 的数量应尽可能相等。具体编码映射关系分别如表 5-8 和表 5-9 所示，特殊控制符 K 的编码映射表如表 5-10 所示。

表 5-8　5B/6B 子模块编码映射关系

Input		RD＝－1	RD＝＋1	Input		RD＝－1	RD＝＋1
symbol	EDCBA	abcdei		symbol	EDCBA	abcdei	
D. 00	00000	100111	011000	D. 16	10000	011011	100100
D. 01	00001	011101	100010	D. 17	10001	100011	
D. 02	00010	101101	010010	D. 18	10010	010011	
D. 03	00011	110001		D. 19	10011	110010	
D. 04	00100	110101	001010	D. 20	10100	001011	
D. 05	00101	101001		D. 21	10101	101010	
D. 06	00110	011001		D. 22	10110	011010	
D. 07	00111	111000	000111	D/K. 23	10111	111010	000101
D. 08	01000	111001	000110	D. 24	11000	110011	001100
D. 09	01001	100101		D. 25	11001	100110	
D. 10	01010	010101		D. 26	11010	010110	
D. 11	01011	110100		D/K. 27	11011	110110	001001
D. 12	01100	001101		D. 28	11100	001110	
D. 13	01101	101100		K. 28	11100	001111	110000
D. 14	01110	011100	101000	D/K. 29	11101	101110	010001
D. 15	01111	010111		D/K. 30	11110	011110	100001
				D. 31	11111	101011	010100

表 5-9　3B/4B 子模块编码映射关系

Input		RD=－1	RD=＋1	Input		RD=－1	RD=＋1
symbol	HGF	fghj		symbol	HGF	fghj	
D. x. 0	000	1011	0100	K. x. 0	000	1011	0100
D. x. 1	001	1001		K. x. 1	001	0110	1001
D. x. 2	010	0101		K. x. 2	010	1010	0101
D. x. 3	011	1100	0011	K. x. 3	011	1100	0011
D. x. 4	100	1101	0010	K. x. 4	100	1101	0010
D. x. 5	101	1010		K. x. 5	101	0101	1010
D. x. 6	110	0110		K. x. 6	110	1001	0110
D. x. P7	111	1110	0001				
D. x. A7	111	0111	1000	K. x. 7	111	0111	1000

表 5-10　特殊控制符 K 的编码映射表

Input		RD=－1	RD=＋1
symbol	HGFEDCBA	abcdeifghj	abcdeifghj
K28. 0	000 11100	001111 0100	110000 1011
K28. 1	001 11100	001111 1001	110000 0110
K28. 2	010 11100	001111 0101	110000 1010
K28. 3	011 11100	001111 0011	110000 1100
K28. 4	100 11100	001111 0010	110000 1101
K28. 5	101 11100	001111 1010	110000 0101
K28. 6	110 11100	001111 0110	110000 1001
K28. 7	111 11100	001111 1000	110000 0111
K23. 7	111 10111	111010 1000	000101 0111
K27. 7	111 11011	110110 1000	001001 0111
K29. 7	111 11101	101110 1000	010001 0111
K30. 7	111 11110	011110 1000	100001 0111

1）直流平衡

高速串行总线通常会使用交流（AC）耦合电容，而通过编码技术使得直流平衡（DC Balance）的原理可以从电容“隔直流、通交流”的角度理解。如图 5-15 所示，直流平衡时，位流中的 1 和 0 交替出现，可认为是交流信号，可以顺利地通过电容；直流不平衡时，位流中出现多个连续的 1 或者 0，可认为该时间段内的信号是直流，通过电容时会因为电压位阶的关系导致传输后的编码错误。高速串行总线采用编码技术的目的是平衡位流中的 1 和 0，从而达到直流平衡。大多数串行电路都是交流耦合（AC Coupling），就是会在发送端（TX）串接电容。电容是隔直通交的，如果不做直流平衡，会把直流信号滤除，信号会畸变。但并不是所有的串行电路标准都是交流耦合，比如 HDMI 就是直流耦合（DC Coupling），也就是说 HDMI 标准电气编码并不是直流平衡的。

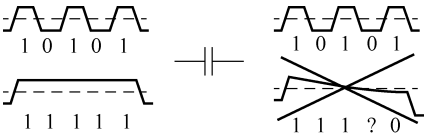


图 5-15　DC 平衡概念

2）不一致性（Disparity）和极性偏差（Running Disparity，RD）

为了明白 8B/10B 的编码原理，首先要明白两个重要的概念：“不一致性”（Disparity）和

“极性偏差”(Running Disparity,RD),也称运行不一致性。

不一致性(Disparity)表示编码后的码型数据中“1”的个数与“0”的个数的差值。由编码规则可知:不一致性的取值只有“+2”(“0”比“1”多两个)、“0”(“0”和“1”数量相等)和“-2”(“0”比“1”少两个)。编码中“1”和“0”数量相等的码字称为“完美平衡码”。

RD是对编码后的数据流不一致性的一个统计,如果“1”的个数大于“0”的个数,则RD取正,记为RD+;如果“1”的个数小于“0”的个数,RD取负,记为RD-。8B/10B编码由3B/4B编码和5B/6B编码两部分组合而成,通过传递RD参数来使整个编码结果具有很好的直流平衡性。

在编码时,RD的初始值为负,即RD-,根据当前的RD值,决定相应的编码输出。例如:在表5-9中,对于D.x.3(011),其对应的4B码字有两种:1100和0011,若此时RD为负,则取1100作为其对应的4B码字输出,同时检验此时的编码是否为完美编码,如果是完美编码,则保持RD的极性不变;否则改变RD的极性。通过控制RD的极性,同时在编码时根据RD的极性选择相对应的编码值,使得编码后的数据流有更好的直流平衡特性。具体关系如图5-16所示。

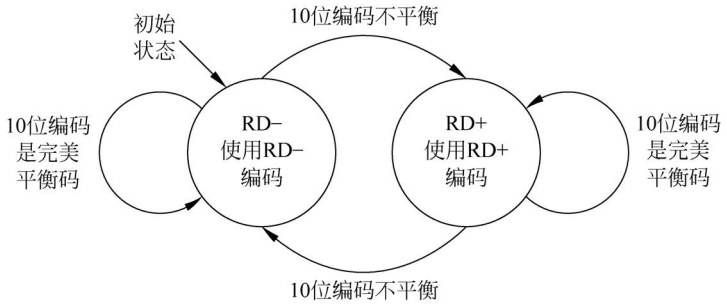


图5-16　根据RD值不同时的编码

编码时,数据不断地进入8B/10B编码器生成10位数据,前面所有已编码的10位数据不一致性累积产生的状态就是运行不一致性,即RD。

在满足下列任何条件时运行不一致错误输出信号rderr置为高电平:

- 当前运行不一致RD为正且6比特码组中1的个数大于0的个数或者为111000;
- 当前运行不一致RD为负且6比特码组中0的个数大于1个或者为000111;
- 在6比特码组之后,运行不一致RD为正且4比特码组中1的个数大于0的个数或者为1100;
- 在6比特码组之后,运行不一致RD为负且4比特码组中0的个数大于1的个数或者为0011。

rderr仅用来标识一些无效的10比特编码,严格根据上述规则确定置为高电平还是低电平,rderr的计算与特殊控制符K码错误(kerr)信号完全无关。对于有效编码但不一致性错误的10比特码字从技术上说是无效码字,但不会使kerr信号置为高电平,仅使rderr信号置为高电平。

3) 通用处理过程

8B/10B编译码器IP核的通用应用框架(GFP)如图5-17所示。

在传输网络的入口,如果译码器接收到无组织的码字(例如,非法码字或存在运行不一

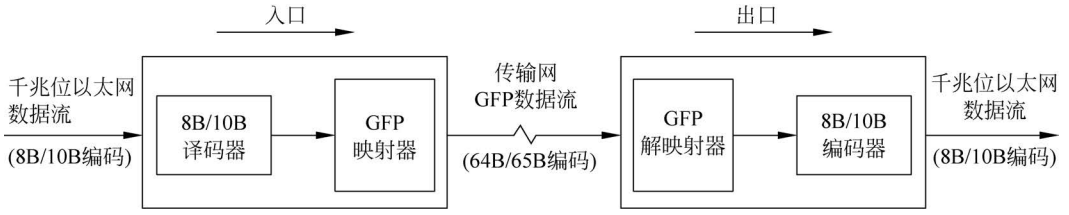


图 5-17　8B/10B 编译码器 IP 核的通用应用框架

致错误的码字),则将 kerr 或 rderr 信号分别置为高电平。通过将这些错误指示信号置位,译码器提示映射器接收到了无效码字,映射器据此产生一个特殊控制字符 10B_ERR 字。另外,映射器在将数据发送到传输网络之前重新映射 8B/10B 码字到 64B/65B 码字。

在传输网络的出口,解映射器解码 64B/65B 码字并将结果送给 8B/10B 编码器。当编码器收到 10B_ERR 码字时,编码器根据运行不一致 RD 的情况选择两个具有中立 RD 的非法码字之一发送:0011110001(RD−)或 110000 1110(RD+)。

4) 特殊 K 码

除了 256 个数据字符外,8B/10B 码定义了 13 个特殊控制字符。256 个数据字符命名为 Dx.y,特殊控制字符命名为 Kx.y(不包括 10B_ERR)。其中 x 表示 5 比特组的值,y 表示 3 比特组的值。

特殊控制字符用于指示数据是空闲、测试数据还是数据分隔符。在应用中,编码的字符是逐比特串行传输的,逗号字符 K28.5 通常用于码字对齐,因其 10 比特码只会出现在 K28.7 字符之后(只有在诊断期间通常才发送 K28.7 字符),而不会出现在比特流的其他位置。除此之外,K28.5 还可以表示 IDLE 码。

表 5-11 给出了 8B/10B 编译码器使用的特殊 K 码。

表 5-11　特殊 K 码

10 比特特殊 K 码	等价的 8 比特码	10 比特特殊 K 码	等价的 8 比特码
K28.0	8'b00011100	K28.1	8'b00111100
K28.2	8'b01011100	K28.3	8'b01111100
K28.4	8'bl0011100	K28.5	8'b10111100
K28.6	8'b11011100	K28.7	8'b11111100
K23.7	8'b11110111	K27.7	8'b11111011
K29.7	8'b11111101	K30.7	8'b11111110
10B_ERR	8'b11111111		—

5) 编码器

要编码 8 比特数据字,必须在 datain 输入端口上提供 8 比特数值且 ena 输入信号必须置为有效的高电平。当插入特殊 10 比特码字时,等价的 8 比特码放置在 datain 输入端口上并且 将 kin 输入信号置为高电平。8B/10B 编译码器进行错误检查以确保特殊控制字符是有效的,如果是无效的,则将 kerr 输出信号置为高电平。但必须注意:尽管 10B_ERR 码被看作无效特殊字符,但插入该码时不会造成 kerr 信号置位(置为高电平)。

如果在 idle_ins 输入为高电平,则在 ena 信号没有置位时,会自动插入空闲字符 K28.5。编码器对无效字符采用与 IDLE(空闲)码 K28.5 一样的方式进行编码,译码器将无效字符

看作空闲码 IDLE。

运行不一致性 RD 可以强制为正或负，从而允许用户插入特殊的重同步模板或不一致错误。当 rdforce 输入信号置为高电平时，rdin 端口上的值就被当作当前的运行不一致性 RD。当 rdin 为 0 时，则强制编码器产生一个负或中立不一致性的编码码字；设置为 1 时，强制编码器产生一个正或中立不一致性的编码码字。

两个编码器级联可以实现 16 比特数据字的编码，通过将高位字节编码器的 rdcascade 输出连接到低位字节编码器的 rdin 输入、同时将低位字节编码器的 rdout 输出连接到高位字节编码器的 rdin 输入来实现编码器的级联。这样连接可以保证正常地运行不一致性计算。如要考虑 rdin 输入端口的值，而不是使用编码器内部生成的运行不一致性 RD 时，必须将编码器的 rdforce 输入信号置为高电平。两个编码器的 ena 输入端口必须同时为高电平或低电平。kin[1]信号对应 datain[15:8]，kin[0]信号对应 datain[7:0]。如果串行传输编码码字，则应该先传输 datain[15:8]总线上的数据。

图 5-18 给出了实现级联编码的两个编码器的连接示意图。

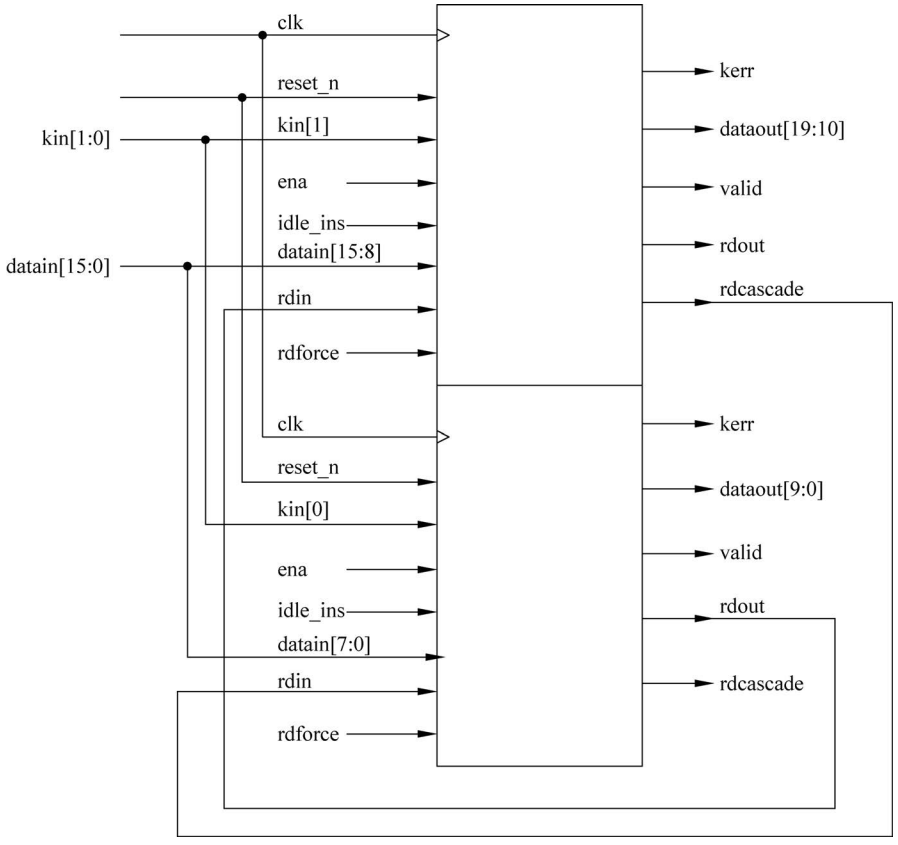


图 5-18　级联编码的连接示意图

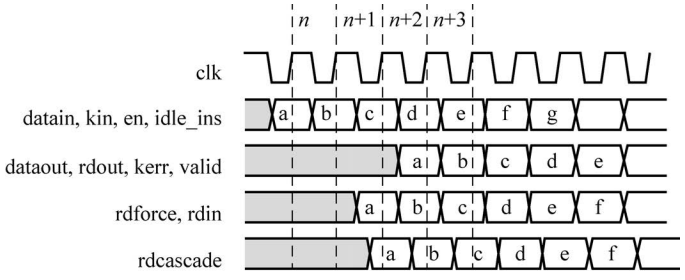
其中，ena、idle_ins 和 rdforce 信号置为高电平(逻辑 1)。

实现编码器时，如果选择了参数“寄存输入/输出”，则此编码器是添加流水结构的，因而，对一个字符的编码需要 3 个时钟周期。如图 5-19(a)所示。编码器在第 n 个时钟周期的上升沿采样 datain 和 kin 端口上的数值，在第 $n+2$ 个时钟上升沿之后开始输出编码，在第

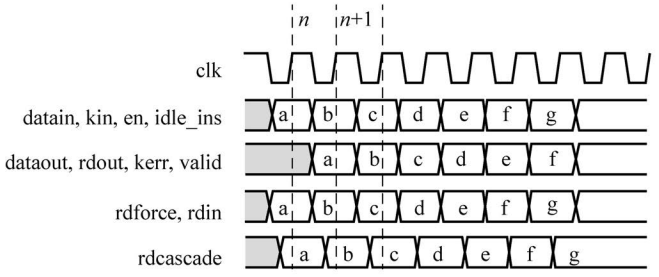
$n+3$ 个时钟的上升沿则可输出稳定编码用于采样。级联编码配置下，输入 rdforce 和 rdin 信号的数据通道是没有加流水的。如果在非级联的编码配置中使用 rdforce 和 rdin 输入信号，则应该将它们相对于 datain 和 kin 信号延时 2 个时钟周期。如图 5-19(b)所示。

如果实现编码器时，将“寄存输入/输出”参数关闭，则对一个字符的编码需要 1 个时钟周期。编码器在第 n 个时钟周期的上升沿采样 datain 和 kin 端口上的数值，用于编码，在第 $n+1$ 个时钟周期的上升沿输出稳定的编码用于采样。

图 5-19 给出了上述两种情况下编码器的信号时序。



(a) 编码定时图——3周期延迟



(b) 编码定时图——1周期延迟

图 5-19　编码器的信号时序

6）译码器

译码器的功能是将接收的 10 比特码字译码成 8 比特字符。当接收到特殊的 10 比特 K 码时，译码器将其转换为 8 比特值并将 kout 信号置为高电平。译码器也检查无效的 10 比特码字，如果接收到无效码字，则置位 kerr 信号为高电平并译码输出一个任意值。另外，译码器将 10B_ERR 字符标记为无效字符，并在接收到该字符时将 kerr 信号置为高电平。

当 idle_del 信号为高电平时，译码器删除所有标记为特殊 IDLE 字符 K28.5 的 10 比特码字。接收机检测到不等式错误时，将 rderr 信号置为高电平。

(1) 级联译码。两个译码器可以同时对两个码字进行译码，级联方式与编码器级联类似：将第一个译码器的输出信号 rdcascade 连接到第二个译码器的 rdin 输入端口，并且将第二个译码器的输出信号 rdout 连接到第一个译码器的 rdin 输入端口。两个译码器的 rdforce 信号必须都置为高电平(连接到逻辑 1)。

在级联译码配置下，馈入 rdin 和 rdforce 信号的数据通道没有加流水。如果在非级联的译码器中使用这两个输入信号，则应将其相对于 datain 和 kin 输入信号延时 1 个时钟周期。

(2) 译码延时。译码器是流水的,需要两个时钟周期完成一个字符的译码。相应于在第 n 个时钟周期上升沿采样的数据 datain 值的译码值,在第 $n+1$ 个时钟周期输出,在第 $n+2$ 个时钟周期的上升沿采样可用。

译码器的信号时序如图 5-20 所示。

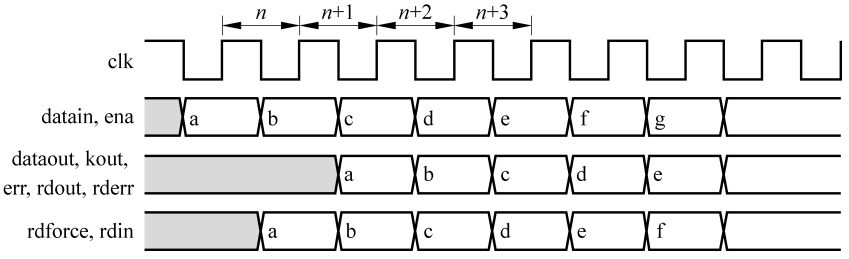


图 5-20　译码器的信号时序

4. 参数设置

在调用过程中,8B/10B 编/译码器 IP 核的参数设置说明请参见表 5-12。

表 5-12　8B/10B 编/译码器 IP 核的参数设置

参　数	说　明
操作模式	选择用 IP 核实现 8B/10B 编码器还是 8B/10B 译码器
寄存输入/输出	选择是否寄存输出和输出,值可以是 On 或 Off。选择 On 时有 3 个时钟周期的延时,选择 Off 时延时为 1 个时钟周期

5. 仿真结果

以编码器为例,从如图 5-21 所示的仿真结果可以看出,在使能信号 ena 置为高电平后例化 8B/10B IP 核开始工作,reset_n 置于无效电平 1,此时 datain 为输入的 8 位码字,可以是数据或者命令,经过 8B/10B 编码器之后,10 位输出编码为 dataout。同时在输出码字期间,valid 信号为有效高电平。rdout 为对应当前码字的当前运行不一致 RD 输出值。对于某些 datain 输入码,比如 6BH,76H,E4H,其对应的输出码字维持不变,直到更新输入码字为止,究其原因是其对应的 10 位编码为完美平衡码,即码中 1 的个数和 0 的个数相等,此时输出编码维持不变。而对于另外一类 datain 输入码,比如 BCH,BOH,3FH,则由于其对应的 10 位编码为不完美平衡码,即码中 1 的个数和 0 的个数不相等,则编码在 RD−编码和 RD+编码之间切换,此时对应的 rdout 输出在 0(即 RD 为−1)与 1(即 RD 为+1)之间不断切换。

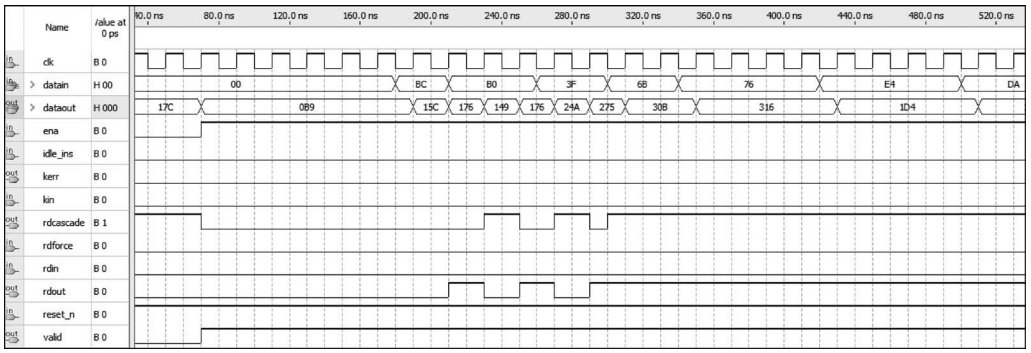


图 5-21　8B/10B 编码器的仿真结果

5.3 原 Altera 公司特定功能 IP 核(ALT 类)

5.3.1 ALTMEMMULT IP 核实现整数乘法

ALTMEMMULT IP 核用于创建使用 Intel 公司 FPGA 片上存储块(M512、M4K、M9K 和 MLAB)实现同步乘法的乘法器。

1. 功能和特点

ALTMEMMULT IP 核的主要功能特点是:

(1) 利用 FPGA 片上存储器资源生成基于存储的乘法器。

(2) 支持的数据位宽范围为 1～512 位。

(3) 支持有符号数和无符号数操作。

(4) 在 RAM 存储器中存储多个常数。

(5) 提供选择 RAM 块类型的选项。

(6) 支持可选的同步复位和加载控制输入。

(7) 支持流水,输出延时固定。

2. 接口信号说明

图 5-22 给出了 ALTMEMMULT IP 核的接口信号描述。ALTMEMMULT IP 核的接口信号说明如表 5-13 所示。

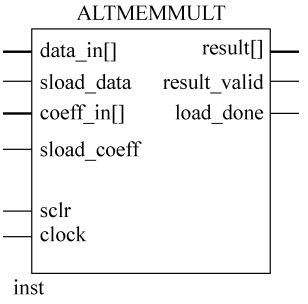


图 5-22 ALTMEMMULT IP 核的接口

表 5-13 ALTMEMMULT IP 核的接口信号说明

信号名称	信号方向	说 明
clock	I	乘法器的时钟信号
coeff_in[]	I	系数输入端口,位宽由参数 WIDTH_C 决定
data_in[]	I	数据输入端口,位宽由参数 WIDTH_D 决定
sclr	I	同步复位信号
sel[]	I	固定系数选择,位宽由参数 WIDTH_S 决定
sload_coeff	I	同步加载系数输入,用 coeff_in 输入的值代替当前选择的系数值
sload_data	I	同步加载数据输入,通知进行新的乘法操作并取消已有操作。如果参数 MAX_CLOCK_CYCLES_PER_RESULT 的值为 1,则该信号无效
result[]	O	乘法器输出,位宽由参数 WIDTH_R 决定
result_valid	O	表示在完成乘法运算时输出数据是否有效,如果参数 MAX_CLOCK_CYCLES_PER_RESULT 的值为 1,则不使用该信号
load_done	O	表示已完成新系数的加载,该信号不为高电平时不能将其他系数值加载到存储器

3. 参数设置

调用过程中,ALTMEMMULT IP 核的参数设置说明请参见表 5-14。

表 5-14 ALTMEMMULT IP 核的参数设置

参 数 名 称	类型	说 明
WIDTH_D	整数	指定端口 data_in[ ]的数据位宽
WIDTH_C	整数	指定端口 coeff_in[ ]的数据位宽
WIDTH_R	整数	指定端口 result[ ]的数据位宽

续表

参 数 名 称	类型	说　　明
WIDTH_S	整数	指定端口 sel[ ]的数据位宽
COEFFICIENT0	整数	指定第 1 个固定系数的值
TOTAL_LATENCY	整数	指定从乘法开始到结果可用的总的时钟周期数
DATA_REPRESENTATION	字符串	指定 data_in[ ]输入和预加载的系数是有符号的还是无符号的
COEFF_REPRESENTATION	字符串	指定 coeff_in[ ]输入和预加载的系数是有符号的还是无符号的
MAX_CLOCK_CYCLES_PER_RESULT	整数	指定得到每个结果需要的时钟周期数
NUMBER_OF_COEFFICIENTS	整数	指定查找表中系数的个数
RAM_BLOCK_TYPE	字符串	规定了 RAM 块的类型，包括 AUTO，SMALL，MEDIUM，M512，M4K 默认为 AUTO

此外模型中还包括配置参数 LPM_HINT、LPM_TYPE 和 INTENDED_DEVICE_FAMILY，请读者自行查阅相关文献。

4. 模块原型

下面的程序是利用 MegaWizard 工具调用和配置 ALTMEMMULT IP 核之后，QUARTUS 软件自动生成的元件模型。其中 examplealtmemmult 是使用 MegaWizard 工具调用 IP 核时自行命名的生成元件模型名称。altsyncram 是调用的宏模块元件名，altsyncram1 是其例化名。examplealtmemmult_altmemmult_ktn 则代表包含基本参数配置的宏模块封装元件名。exmplealtmemmult_altmemmult_ktn_component 则是宏模块封装元件的例化名。

【代码 5-5】

```
LIBRARY ALTERA_MF;
 USE ALTERA_MF.ALL;
 -- synthesis_resources = altsyncram 1
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY  examplealtmemmult_altmemmult_ktn IS
    PORT
    (
       clock                              :IN  STD_LOGIC;
       data_in                            :IN  STD_LOGIC_VECTOR (7 DOWNTO 0);
       result                             :OUT  STD_LOGIC_VECTOR (15 DOWNTO 0);
       result_valid                       :OUT  STD_LOGIC
    );
END examplealtmemmult_altmemmult_ktn;
ARCHITECTURE RTL OF examplealtmemmult_altmemmult_ktn IS
    SIGNAL  wire_altsyncram1_q_a          :STD_LOGIC_VECTOR (15 DOWNTO 0);
    COMPONENT  altsyncram
    GENERIC
    (
       ADDRESS_ACLR_A                     :STRING : = "UNUSED";
       ADDRESS_ACLR_B                     :STRING : = "NONE";
       ADDRESS_REG_B                      :STRING : = "CLOCK1";
       BYTE_SIZE                          :NATURAL : = 8;
       BYTEENA_ACLR_A                     :STRING : = "UNUSED";
```

```
    BYTEENA_ACLR_B                      :STRING : = "NONE";
    BYTEENA_REG_B                       :STRING : = "CLOCK1";
    CLOCK_ENABLE_CORE_A                 :STRING : = "USE_INPUT_CLKEN";
    CLOCK_ENABLE_CORE_B                 :STRING : = "USE_INPUT_CLKEN";
    CLOCK_ENABLE_INPUT_A                :STRING : = "NORMAL";
    CLOCK_ENABLE_INPUT_B                :STRING : = "NORMAL";
    CLOCK_ENABLE_OUTPUT_A               :STRING : = "NORMAL";
    CLOCK_ENABLE_OUTPUT_B               :STRING : = "NORMAL";
    ECC_PIPELINE_STAGE_ENABLED          :STRING : = "FALSE";
    ENABLE_ECC                          :STRING : = "FALSE";
    IMPLEMENT_IN_LES                    :STRING : = "OFF";
    INDATA_ACLR_A                       :STRING : = "UNUSED";
    INDATA_ACLR_B                       :STRING : = "NONE";
    INDATA_REG_B                        :STRING : = "CLOCK1";
    INIT_FILE                           :STRING : = "UNUSED";
    INIT_FILE_LAYOUT                    :STRING : = "PORT_A";
    MAXIMUM_DEPTH                       :NATURAL : = 0;
    NUMWORDS_A                          :NATURAL : = 0;
    NUMWORDS_B                          :NATURAL : = 0;
    OPERATION_MODE                      :STRING : = "BIDIR_DUAL_PORT";
    OUTDATA_ACLR_A                      :STRING : = "NONE";
    OUTDATA_ACLR_B                      :STRING : = "NONE";
    OUTDATA_REG_A                       :STRING : = "UNREGISTERED";
    OUTDATA_REG_B                       :STRING : = "UNREGISTERED";
    POWER_UP_UNINITIALIZED              :STRING : = "FALSE";
    RAM_BLOCK_TYPE                      :STRING : = "AUTO";
    RDCONTROL_ACLR_B                    :STRING : = "NONE";
    RDCONTROL_REG_B                     :STRING : = "CLOCK1";
    READ_DURING_WRITE_MODE_MIXED_PORTS :STRING : = "DONT_CARE";
    read_during_write_mode_port_a       :STRING : = "NEW_DATA_NO_NBE_READ";
    read_during_write_mode_port_b       :STRING : = "NEW_DATA_NO_NBE_READ";
    WIDTH_A                             :NATURAL;
    WIDTH_B                             :NATURAL : = 1;
    WIDTH_BYTEENA_A                     :NATURAL : = 1;
    WIDTH_BYTEENA_B                     :NATURAL : = 1;
    WIDTH_ECCSTATUS                     :NATURAL : = 3;
    WIDTHAD_A                           :NATURAL;
    WIDTHAD_B                           :NATURAL : = 1;
    WRCONTROL_ACLR_A                    :STRING : = "UNUSED";
    WRCONTROL_ACLR_B                    :STRING : = "NONE";
    WRCONTROL_WRADDRESS_REG_B           :STRING : = "CLOCK1";
    INTENDED_DEVICE_FAMILY              :STRING : = "Cyclone III";
    lpm_hint                            :STRING : = "UNUSED";
    lpm_type                            :STRING : = "altsyncram"
);
PORT
(
    aclr0                               :IN STD_LOGIC : = '0';
    aclr1                               :IN STD_LOGIC : = '0';
    address_a                           :IN STD_LOGIC_VECTOR(WIDTHAD_A - 1 DOWNTO 0);
    address_b:IN STD_LOGIC_VECTOR(WIDTHAD_B - 1 DOWNTO 0) : = (OTHERS = > '1');
    addressstall_a                      :IN STD_LOGIC : = '0';
    addressstall_b                      :IN STD_LOGIC : = '0';
    byteena_a:IN STD_LOGIC_VECTOR(WIDTH_BYTEENA_A - 1 DOWNTO 0) : = (OTHERS = > '1');
```

```
        byteena_b:IN STD_LOGIC_VECTOR(WIDTH_BYTEENA_B - 1 DOWNTO 0) := (OTHERS => '1');
        clock0                          :IN STD_LOGIC := '1';
        clock1                          :IN STD_LOGIC := '1';
        clocken0                        :IN STD_LOGIC := '1';
        clocken1                        :IN STD_LOGIC := '1';
        clocken2                        :IN STD_LOGIC := '1';
        clocken3                        :IN STD_LOGIC := '1';
        data_a:IN STD_LOGIC_VECTOR(WIDTH_A - 1 DOWNTO 0) := (OTHERS => '1');
        data_b:IN STD_LOGIC_VECTOR(WIDTH_B - 1 DOWNTO 0) := (OTHERS => '1');
        eccstatus:OUT STD_LOGIC_VECTOR(WIDTH_ECCSTATUS - 1 DOWNTO 0);
        q_a                             :OUT STD_LOGIC_VECTOR(WIDTH_A - 1 DOWNTO 0);
        q_b                             :OUT STD_LOGIC_VECTOR(WIDTH_B - 1 DOWNTO 0);
        rden_a                          :IN STD_LOGIC := '1';
        rden_b                          :IN STD_LOGIC := '1';
        wren_a                          :IN STD_LOGIC := '0';
        wren_b                          :IN STD_LOGIC := '0'
    );
    END COMPONENT;
BEGIN
   result <= ( wire_altsyncram1_q_a(15 DOWNTO 0));
   result_valid <= '0';
   altsyncram1                          :  altsyncram
     GENERIC MAP (
       INIT_FILE => "examplealtmemmult.hex",
       OPERATION_MODE => "ROM",
       OUTDATA_REG_A => "CLOCK0",
       RAM_BLOCK_TYPE => "AUTO",
       WIDTH_A => 16,
       WIDTHAD_A => 8,
       INTENDED_DEVICE_FAMILY => "Cyclone |||"
     )
     PORT MAP (
       address_a => data_in(7 DOWNTO 0),
       clock0 => clock,
       q_a => wire_altsyncram1_q_a
     );
END RTL;                                -- examplealtmemmult_altmemmult_ktn
 -- VALID FILE
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY examplealtmemmult IS
    PORT
    (
       clock                            : IN STD_LOGIC ;
       data_in                          : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
       result                           : OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
    );
END examplealtmemmult;
ARCHITECTURE RTL OF examplealtmemmult IS
    SIGNAL sub_wire0                    : STD_LOGIC_VECTOR (15 DOWNTO 0);
    COMPONENT examplealtmemmult_altmemmult_ktn
    PORT (
       clock                            : IN STD_LOGIC ;
       data_in                          : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
```

```
        result                                  : OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
    );
    END COMPONENT;
BEGIN
    result    <= sub_wire0(15 DOWNTO 0);
    examplealtmemmult_altmemmult_ktn_component : examplealtmemmult_altmemmult_ktn
    PORT MAP (
        clock => clock,
        data_in => data_in,
        result => sub_wire0
    );
END RTL;
```

5. 仿真结果

例化参数设置：输入数据 data_in 为 8 位有符号数；常数乘数系数为 8 位有符号数，初值设置成 2。如图 5-23 所示从仿真结果可以看出，数据输出 DOUT 是数据输入 DIN 乘以系数 2 的乘积，并且输出结果相对于输入数据延时 1 个时钟周期。

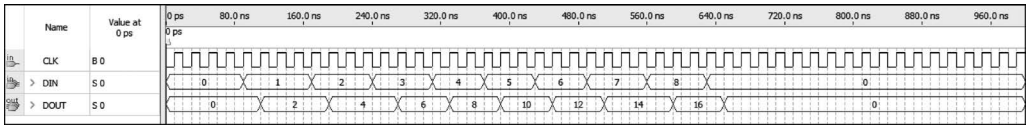


图 5-23　ALTMEMMULT IP 核仿真结果

5.3.2　锁相环 ALTPLL 的调用

锁相环(PLL)IP 核是生成输出时钟(同步于其输入时钟)的闭环频率控制系统，它比较输入信号和压控振荡器(VCO)输出信号之间的相位差并实现相位同步，从而在输入或参考信号的频率上保持固定相位角。同步或系统的负反馈环强制 PLL 锁定相位。PLL 可以配置为频率乘法器、除法器、解调器、跟踪生成器或时钟恢复电路。也可以用 PLL 生成固定频率时钟，配置从有噪信道恢复信号或者在 FPGA 设计中分布时钟信号。

Intel 公司提供了用于实现 PLL 功能的 IP 核 ALTPLL。

1. 功能和特点

ALTPLL IP 核的主要功能和特点如下所述。

(1) 支持五种不同的 PLL 类型(时钟反馈模式)：正常模式、源同步模式、零延时缓存模式、无补偿模式和外部反馈模式。

(2) 支持多种操作模式，具体与所用 FPGA 芯片系列有关。

(3) 支持 pllena、areset 和 pfdena 等控制信号。

(4) 支持两个输入参考时钟的切换。

(5) 提供扩谱时钟。

(6) 支持门锁定和自复位。

(7) 带宽可编程。

(8) 支持 PLL 动态重配置和动态相位配置。

(9) 支持 Intel 公司的 Arria、HardCopy、Cyclone 和 Stratix 系列 FPGA。

2. 接口信号说明

ALTPLL IP 核的接口信号描述如图 5-24 所示。

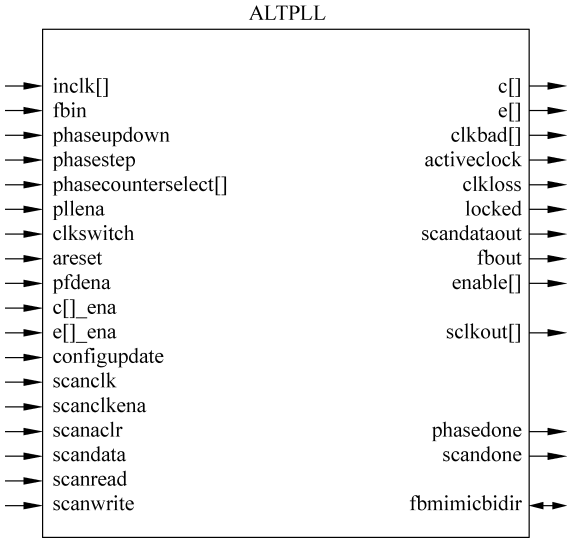


图 5-24　ALTPLL IP 核的接口信号

ALTPLL IP 核的接口信号说明请参见表 5-15。需要说明的是，表中部分信号是针对特定 FPGA 系列芯片的，这里不一一说明。另外，表中的[]在实际例化 IP 核时为具体的整数值，例如 inclk0、inclkl、cl_ena 和 e0_ena 等。

表 5-15　ALTPLL IP 核的接口信号

信 号 名 称	信号方向	说　　明
areset	I	将所有计数器的值复位到初始值，包括 GATE_LOCK_COUNTER 参数
c[]_ena	I	输出时钟 c[]的使能输入端口
clkswitch	I	时钟输入端口 inclk0 和 inclkl 之间动态切换或手动覆盖自动时钟切换的控制输入端口。如果仅创建 inclkl 端口，应该创建该端口
configupdate	I	PLL 的动态配置信号
e[]_ena[]	I	外部输出时钟 e[]的使能输入端口
fbin	I	PLL 的外部反馈输入端口。如果 PLL 操作在外部反馈模式下，则必须创建该端口。为完成反馈环，必须在电路板上将该端口与 PLL 的外部时钟输出端口连接。在 Stratix Ⅲ系列 FPGA 中，如果 PLL 操作在零延时缓存模式且不使用 fbmimicbidir 端口，则必须将 fbin 端口和 fbout 端口连接在一起
inclk[]	I	驱动时钟网络的时钟输入。如果创建多个 inclk[ ]端口，则必须用 clkselect 端口指定使用的时钟。inclk0 总是连接的，如果需要切换，则还要连接其他的时钟输入。专用时钟引脚或 PLL 输出时钟可以驱动该端口
pfdena	I	使能相位频率检测器(PFD)。禁止 PFD 时，PLL 不管输入时钟如何都连续工作。由于 PLL 输出时钟频率一段时间内不会改变，因此可以在某可靠的输入时钟无效时，用该端口信号实现关闭和清除功能
phasecounterselect[]	I	指定计数器选择
phasestep	I	指定动态相位偏移
phaseupdown	I	指定动态相位调整方向

续表

信 号 名 称	信号方向	说　　明
pllena	I	PLL 使能信号。置位时 PLL 驱动输出信号，否则不输出。在该信号重新置位时 PLL 必须重新锁定。FPGA 芯片上的所有 PLL 都共用该端口信号
scanaclr	I	实时可编程扫描链的异步复位信号
scanclk	I	串行扫描链的输入时钟端口
scanclkena	I	串行扫描链的时钟使能端口
scandata	I	串行扫描链的数据
scanread	I	控制串行扫描链从 scandata 端口读取数据的控制端口
scanwrite	I	控制实时可编程扫描链写数据到 PLL 的控制端口
activelock	O	用于在时钟切换电路启动时指定哪个时钟为主参考时钟。如果正在使用 inclk0，则该信号变为低电平；如果正在使用 inclk1，则该信号变为高电平。在主参考时钟没有正确切换时，可以将 PLL 设置为自动启动时钟切换或者用 clkswitch 输入端口信号手动启动时钟切换
c[]	O	PLL 的时钟输出
clkbad[]	O	clkbad0 和 clkbad1 检查输入时钟是否正常翻转。如果 inclk0 不能正常翻转，则 clkbad0 变为高电平；如果 inclk1 不能正常翻转，则 clkbad1 变为高电平
clkloss	O	时钟切换电路启动的指示
enable[]	O	使能脉冲输出端口，仅在 PLL 为 LVDS 工作模式时可用
e[]	O	馈入到专用时钟引脚的 PLL 时钟输出
fbout	O	该端口通过模拟电路连接到 fbin 端口。如果没有连接反馈路径，则编辑器自动将该端口连接到 fbin 端口。另外，会自动添加一个 clkbuf 原语来指定使用的资源类型，与其他时钟网络类似。仅在 PLL 处于外部反馈模式时该端口可用
locked	O	该输出端口标识 PLL 实现了相位锁定。在 PLL 锁定后保持为高电平，失锁时保持为低电平
phasedone	O	标识动态相位重配置完成
scandataout	O	串行扫描链的数据输出。可以用于判断什么时候 PLL 完成了重配置。重配置完成时清除最后的输出
scandone	O	标识扫描链写操作启动。当启动扫描链写操作时变为高电平，当扫描链写操作启动完成后变为低电平
sclkout[]	O	串行时钟输出端口，仅在 PLL 处于 LVDS 模式时可用
vcounverrange	O	指示 VCO 频率超出了合法 VCO 范围
vcounderrange	O	指示 VCO 频率与尚未满足合法 VCO 范围
fbmimicbidir	IO	该双向端口连接到模拟电路，必须连接到位于 PLL 的正反馈专用输出引脚中的双向引脚上

需要说明的是，在 Mega Wizard 插件管理器中配置 ALTPLL IP 核的不同功能选项会影响 IP 核的端口设置，而且这些选项都是与具体芯片相关的，且某些端口互相关联。

3. 功能描述

1) PLL 的结构

图 5-25 给出了典型的 PLL 组成结构。

PLL 由预缩放计数器(分频器 N)、相位-频率检测器(PFD)电路、电荷泵、环路滤波器、VCO、反馈计数器(M)和后缩放器(乘法 K 和除法 V)。

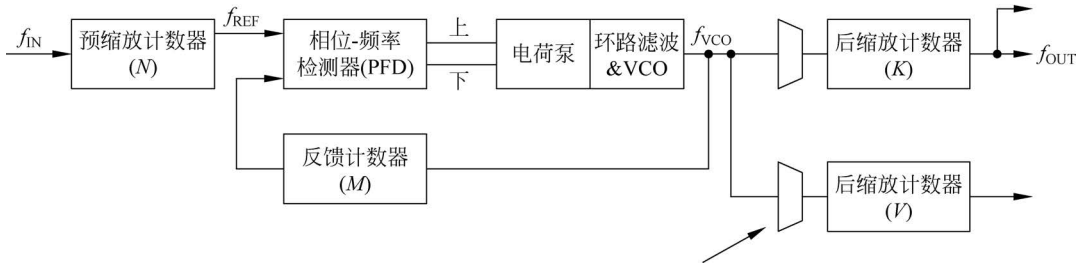


图 5-25　PLL 的组成结构

PFD 检测参考信号(f_{REF})和反馈信号的相位差和频率差,电荷泵将 PFD 的误差信号转换成电流脉冲,环路滤波器对电荷泵输出的电流脉冲进行积分,以生成施加于 VCO 的调谐电压,该电压控制 VCO。VCO 振荡器根据控制电压调整振荡频率,从而改变反馈信号的相位和频率。在 f_{REF} 信号和反馈信号达到相同相位和频率时,PLL 实现相位锁定。在反馈路径中插入计数器 M 的目的是使反馈信号振荡在 f_{REF} 信号频率的 M 倍上。f_{REF} 信号等于输入时钟被预缩放计数器 N 分频后的结果。参考频率定义为 $f_{REF}=f_{IN}/N$,VCO 输出频率为 $f_{VCO}=\dfrac{f_{IN}M}{N}$,PLL 的输出频率为 $f_{OUT}=(f_{IN}*M)/(N*K)$。

衡量 PLL 性能的主要参数有:

(1) PLL 锁定时间,也称为 PLL 的捕获时间,是 PLL 在上电后、可编程输出频率改变后或 PLL 复位后 PLL 达到目标频率和相位关系所要求的时间。

(2) PLL 分辨率,PLL VCO 的最小频率增量值,根据计数器 M 和计数器 N 的比特数确定。

(3) PLL 采样速率。在 PLL 中实现正确的相位和频率所要求的采样频率 F。

2) PLL 的类型

不同系列的 FPGA 支持的 PLL 类型不同,一般支持一种或两种 PLL 类型。例如,Stratix 系列 FPGA 支持两种类型的 PLL,Cyclone 系列 FPGA 仅支持一种类型的 PLL。两种类型的 PLL 在模拟部分是相同的,在数据部分稍有不同(例如,某种类型支持的计数器更多)。

不同系列 FPGA 中可用 PLL 的个数及类型请参见表 5-16。

3) 操作模式

ALTPLL 支持五种不同的时钟反馈模式,每种模式都允许对时钟进行乘除、相位偏移和占空比设置。在不同的芯片系列中,ALTPLL 支持不同的时钟反馈模式。

(1) 标准模式:PLL 反馈路径源是全局或局域时钟网络,对于该时钟类型和特定的 PLL 输出,这可以最小化到寄存器的时钟延时。在此模式下,还可以指定被补偿的 PLL 输出。

表 5-16　不同 FPGA 芯片系列的可用 PLL 个数及类型

芯片系列	PLL 总数	PLL 类型	芯片系列	PLL 总数	PLL 类型
Arria GX	8	增强的快速 PLL	Arria Ⅱ GX	6	左-右型
Stratix Ⅳ	12	上-下型和左-右型	Stratix Ⅲ	12	上-下型和左-右型
Stratix Ⅱ	12	增强的快速 PLL	Stratix Ⅱ GX	8	增强的快速 PLL
Stratix	12	增强的快速 PLL	Stratix GX	8	增强的快速 PLL
Cyclone Ⅳ	4	Cyclone Ⅳ PLL	Cyclone Ⅲ	4	Cyclone Ⅲ PLL
Cyclone Ⅱ	4	Cyclone Ⅱ PLL	Cyclone	2	Cyclone PLL

(2) 源同步模式：数据和时钟信号同时到达各自的输入引脚。在此模式下，可以确保在任何输入/输出使能寄存器的时钟和数据端口，信号均具有相同的相位关系。

(3) 零延时缓存模式：PLL 反馈路径局限于专用 PLL 外部时钟输出引脚上。片外驱动的时钟端口与时钟输入是相位对齐的，且时钟输入和外部时钟输出之间的延时最小。

(4) 无补偿模式：PLL 反馈路径局限于 PLL 环路中，没有时钟网络或其他外部源。处于无补偿模式的 PLL 没有时钟网络补偿，但其时钟抖动是最小化的。

(5) 外部反馈模式：PLL 输出 fout 补偿到 PLL 的反馈输入 fbin，这样可以最小化输入时钟引脚和反馈时钟引脚之间的延时。

4) 输出时钟

ALTPLL 能够产生的时钟输出信号个数与 PLL 类型和选择实现 ALTPLL 的 FPGA 芯片系列有关。例如，对于 Stratix Ⅳ系列 FPGA，一个左-右型 PLL 可以生成 7 个时钟输出信号，一个上-下型 PLL 可以生成 10 个时钟输出信号。生成的时钟输出信号可以作为 IP 核的时钟或者 IP 核外其他外部模块的时钟。

ALTPLL 没有专用的输出使能端口，但可以禁止 PLL 输出。可以用 pllena 信号或 areset 信号禁止 PLL 输出计数器，进而禁止 PLL 输出时钟。另外一种方法是将 PLL 输出时钟信号馈入 ALTIOBUF IP 核，然后用缓存器的使能输出端口去禁止这些信号。

可以通过参数设置输出时钟的频率、相位偏移和占空比等参数。

5) 高级特性

Intel 公司 FPGA 芯片提供片上 PLL 高级特性，包括门控锁定、时钟切换、动态重配置、带宽可编程、带宽重配置、扩频时钟、后缩放计数器级联，这些高级特性不仅可以提高系统和芯片性能，还可以提供高端时钟接口。

(1) 高级控制信号。可以用信号 pllena，areset 和 pfdena 观测和控制 PLL 操作和重同步。

① pllena：用 pllena 信号可以使能和禁止 PLL，该信号置为低电平时 PLL 不驱动任何输出时钟信号，从而失锁。PLL 中的所有计数器(包括门控锁定计数器)都返回到默认状态；该信号置为高电平时，PLL 驱动输出时钟信号并实现锁定。每个 FPGA 芯片的所有 PLL 共用一个 PLL 使能端口。默认情况下，该信号内部连接到 Vcc。

② areset：该信号是每个 PLL 的复位或重同步输入，芯片输入引脚或内部逻辑可以驱动该信号。该信号为高电平时，PLL 中的所有计数器(包括门控锁定计数器)都复位到初始值，清除 PLL 输出且 PLL 处于解锁状态。VCO 也复位到标称设置；该信号为低电平时，PLL 重同步其输入并实现锁定。

③ pfdena：该信号使能或禁止 PFD 电路。默认情况下 PFD 电路是使能的。禁止 PFD 电路时，PLL 输出与输入时钟无关，并可能漂移到锁定窗之外。默认情况下，该信号内部连接到 Vcc。

(2) 时钟切换。时钟切换特性允许 PLL 在两个输入参考时钟之间进行切换。该属性可以用于在不同频率的时钟输入之间进行切换，广泛应用于通信、存储和服务器中。ALTPLL IP 核支持两种时钟切换模式。

① 自动切换：PLL 监视当前使用的时钟信号，如果它停止在 0～1 之间翻转或失锁，PLL 自动切换到另外一个时钟信号(inclk0 或 inclk1)。

② 手动切换：用 clkswitch 信号控制时钟切换。

(3) 扩频时钟。扩频技术可以减小系统中的电磁干扰，将时钟能量分散到较宽的频带范围来实现。扩频时钟特性将基本时钟频率能量分散到整个设计频带内来最小化特定频率点上的能量尖峰。通过减小频谱尖峰的幅度，使用户设计能够满足电磁干扰(EMI)发射兼容标准，并减小关于传统 EMI 抑制的代价。

为使用扩展频谱时钟属性，必须将带宽可编程属性设置为 Auto(自动)。

(4) 门控锁定和自复位。PLL 的锁定时间定义为在芯片上电后、PLL 输出频率改变后或 PLL 复位后 PLL 达到目标频率和相应相位所要求的时间。

下列原因可能造成 PLL 失锁：

① 输入时钟抖动程度较大。

② PLL 的时钟输入上有较大切换噪声。

③ 电源噪声较大导致输出抖动较大和可能的失锁。

④ PLL 的输入时钟故障或停止。

⑤ 通过置位 PLL 的 areset 端口或 pllena 端口复位 PLL。

⑥ 尝试重配置 PLL 时可能导致计数器 M、计数器 N 或相位偏移发生变化，导致 PLL 失锁。但是后缩放计数器的改变不影响 PLL 锁定信号。

⑦ PLL 输入时钟频率漂移到了锁定范围之外。

⑧ 使用 pfdena 端口禁止了 PFD。这种情况下，PLL 输出相位和频率可能漂移到锁定窗以外。

ALTPLL IP 核允许用名称为 locked 的信号监视 PLL 锁定过程并允许在 PLL 失锁时自动复位。PLL 锁定由相位频率检测器中的两个输入信号决定，锁定信号 locked 与 PLL 的输出是异步的。PLL 输出时钟锁定时间与 PLL 输入时钟的门控锁定电路有关。用 PLL 的最大锁定时间除以 PLL 输入时钟的周期可以计算得到实现锁定的时钟周期数。

PLL 锁定复位使 PLL 能够实现锁定在最小和最大输入时钟频率之间。改变输入时钟频率可能会导致 PLL 失锁，但如果输入时钟频率在最小频率和最大频率之间，PLL 总是能够实现锁定。

某些 FPGA 芯片支持门控锁定信号，可以配置可编程的 20 位计数器，用以在用户指定的输入时钟转换次数内保持锁定信号为低。这对消除在 PLL 开始跟踪参考时钟后的锁定信号发生错误翻转是非常有用的。门控锁定允许 PLL 在 locked 信号置位之前锁定，以提供稳定的锁定信号。

locked 信号置位时表示 PLL 时钟输出已经与 PLL 输入参考时钟对齐。locked 信号可能在 PLL 开始跟踪参考时钟时出现翻转。使用门控锁定信号可以避免这样的错误锁定指示。门控 locked 信号或非门控 locked 信号可以馈入逻辑阵列或输出引脚。在必须复位门控计数器时，将 areset 信号或 pllena 信号置位来复位 PLL。

PLL 支持上述原因导致失锁时自动复位 PLL。

(5) 带宽可编程。PLL 带宽定义为 PLL 跟踪输入时钟和抖动的能力，带宽用 PLL 中闭环增益的－3dB 频率衡量，或近似为 PLL 开环响应的单位增益点。Intel 公司 FPGA 芯片提供 PLL 带宽可编程特性，用于配置 PLL 环路滤波器的属性。大部分环路滤波器包含电阻和电容等组件，需要占用板上空间。Intel 公司 FPGA 已经包含了这些组件，而且使用

带宽可编程特性可以控制这些组件对 PLL 带宽的影响。这包括控制电荷泵的电流、环路滤波电阻和改频电容的值。电荷泵的电流直接影响 PLL 带宽，电流越大，PLL 的带宽越高。

(6) 高级 PLL 参数。ALTPLL 参数编辑器提供用 ALTPLL 高级参数来生成输出文件的选项。这些高级参数包括 charge_pump_current、loop_filter_r 和 loop_filter_c，所有的 Intel 公司 FPGA 都支持该选项。

该选项主要给那些了解 PLL 配置细节的用户或非常理解这些参数而且知道如何优化参数的高级用户使用。ALTPLL 参数编辑器不能重用生成的文件，这是因为 ALTPLL 模块的输出文件是用高级参数定义的。Quartus Ⅱ编辑器不能改变这些文件。

(7) PLL 动态重配置。PLL 动态重配置属性允许重配置 PLL。可以用以下端口控制配置过程。

① 输入端口：scanclks，scandata，scanclkena 和 configupdate。

② 输出端口：scandataout 和 scandone，其中端口 scandone 对于 Cyclone、Cyclone Ⅱ、Stratix 和 Stratix GX 系列 FPGA 芯片不可用。

可以用扫描链动态重配置 Stratix 和 Stratix GX 系列 FPGA 的增强型 PLL。根据 PLL 的类型，有两种扫描链可用：长扫描链和短扫描链。长扫描链允许用 6 个核时钟和 4 个外部时钟配置这些 PLL，而短扫描链限制用 6 个核时钟(无外部时钟)配置这些 PLL。

并不是对于所有支持 PLL 的 FPGA 芯片系列都可以用扫描链进行动态重配置。支持标准动态重配置方案的 FPGA 芯片使用配置文件[如十六进制文件(.hex)]或存储器初始化文件(.mif)。这些文件与 ALTPLL_RECONFIG IP 核一起使用，来实现动态配置。

(8) 动态相位重配置。动态相位配置属性允许某 PLL 输出相位根据其他输出及参考时钟来动态调整，而不需要通过相应 PLL 的扫描链发送串行数据。该属性一般称为动态相位步进属性。

可以用该属性实时地快速调整输出时钟的相位偏移。这通过增加或减小到计数器 C 或计数器 M 的 VCO 相位抽头选择实现。默认情况下，相位偏移步长为 VCO 频率的 1/8。然而，可以用 ALTPLL 参数编辑器来修改 PLL 输出时钟的相位偏移步长精度。

为使动态相位偏移正常工作，PLL 必须有以下端口。

① 输入端口：phasecounterselect[3..0]、phaseupdown、phasestep 和 scanclk。

② 输出端口：phasedone。

动态相位配置属性仅对于 Cyclone Ⅳ、Cyclone Ⅲ Arria Ⅱ GX、Stratix Ⅲ和 Stratix Ⅳ系列 FPGA 芯片可用。

4. 参数设置

在 ALTPLL 的参数中，c[]和 e[]端口分别对应 CLK[]和 EXTCLK[]，根据计数器 C 和计数器 E 的参数来区分。另外，表中的[]应该是具体的整数值，例如，参数 C[]_HIGH 最多可以有 10 个变形，分别是 C0_HIGH，C1_HIGH，C2_HIGH，…，C9_HIGH。表 5-17 给出了 ALTPLL IP 核的参数配置设置说明。需要说明的是，其中部分参数是针对具体的 FPGA 系列芯片存在或可用的，这里不一一说明。

表 5-17　ALTPLL IP 核的参数设置

参 数 名 称	类型	说　　明
BANDWIDTH	字符串	指定 PLL 的带宽值，单位是 MHz。不指定该参数值时，编译器自动决定该参数的值以满足其他 PLL 设置
BANDWIDTH_TYPE	字符串	指定 BANDWIDTH 参数的带宽类型。值可以是 AUTO，LOW，MEDIUM 或 HIGH，默认值为 AUTO。对于低带宽选项，PLL 有更好的抗抖动性能，但锁定时间较慢；对于高带宽选项，PLL 锁定较快但抖动跟踪时间长
C[]_HIGH	整数	指定计数器 C[9..0]的高电平周期计数值。默认值为 1
C[]_INITIAL	整数	指定计数器 C[9..0]的初始值。默认值为 1
C[]_LOW	整数	指定计数器 C[9..0]的低电平周期计数值。默认值为 1
C[]_MODE	字符串	指定计数器 C[9..0]的操作模式。值可以是 BYPASS，ODD 或 EVEN，默认值为 BYPASS
C[]_PH	整数	指定计数器 C[9..0]的相位抽头。默认值为 0
C[]_TEST_SOURCE	整数	指定计数器 C[9..0]的测试源。默认值为 0
C[]_USE_CASC_IN	字符串	指定计数器 C[9..1]是否使用级联输入。值可以是 ON 或 OFF，默认值为 OFF
CHARGE_PUMP_CURRENT	整数	指定电荷泵的电流，单位是毫安(mA)
CLK[]_COUNTER	字符串	指定输出时钟端口 CLK[9..0]的计数器。值可以是 UNUSED、C0、Cl、C2、C3、C4、C5、C6、C7、C8 或 C9，默认值为 C0
CLK[]_DIVIDE_BY	整数	指定输出时钟端口 CLK[9..0]的 VCO 频率的整数除数因子，值必须大于 0。仅对使用的相应 CLK[9..0]指定该参数值，默认值为 0
CLK[]_DUTY_CYCLE	整数	指定输出时钟端口 CLK[9..0]的占空比。默认值为 50，即 50%
CLK[]_MULTIPLY_BY	整数	指定输出时钟端口 CLK[9..0]相对于 VCO 频率的整数乘法因子，值必须大于 0。仅对使用的相应 CLK[9..0]指定该参数值，默认值为 0
CLK[]_OUTPUT_FREQUENCY	整数	指定输出时钟端口 CLK[9..0]的输出频率。仅对使用的相应 CLK[9..0]指定该参数值，默认值为 0
CLK[]_PHASE_SHIFT	字符串	指定输出时钟端口 CLK[9..0]的相位偏移，单位是 ps。默认值为 0
CLK[]_TIME_DELAY	字符串	指定输出时钟端口 CLK[9..0]应用的延时值，单位是 ps。该参数仅影响相应的 CLK[9..0]，与 CLK[9..0]_PHASE_SHIFT 参数无关。有效取值范围是－3～6ns(步长为 0.25ns)。仅在使用实时可编程接口对 PLL 重编程时使用该参数
CLK[]_USE_EVEN_COUNTER_MODE	字符串	指定 CLK[9..0]时钟输出是否强制使用偶计数模式。值可以是 ON 或 OFF，默认值为 OFF
CLK[]_USE_EVEN_COUNTER_VALUE	字符串	指定 CLK[9..0]时钟输出是否强制使用偶计数值。值可以是 ON 或 OFF，默认值为 OFF
COMPENSATE_CLOCK	字符串	指定输出时钟端口补偿位置。如果 OPERATION_MODE 参数设置为正常模式，则值可以是 CLK[]、GCLK[]、LCLK[]或 LVDSCLK[]，默认值为 CLK0；如果 OPERATION_MODE 参数设置为零延时缓存模式，则值为 EXTCLK。默认值为 EXTCLK0；如果 OPERATION_MODE 参数设置为源同步模式，则值可以是 CLK[]、GCLK[]、LCLK[]或 LVDSCLK[]
DOWN_SPREAD	字符串	指定降频谱的百分比。取值范围是 0～0.5

续表

参数名称	类型	说明
E[]_HIGH	整数	指定计数器 E[3..0]的高电平周期计数值。取值范围是 1～512，默认值为 1
E[]_INITIAL	整数	指定计数器 E[3..0]的初始值。取值范围是 1～512，默认值为 1
E[]_LOW	整数	指定计数器 E[3..0]的低电平周期计数值。取值范围是 1～512，默认值为 1
E[]_MODE	字符串	指定计数器 E[3..0]的操作模式。值可以是 BYPASS，ODD 或 EVEN，默认值为 BYPASS
E[]_PH	整数	指定计数器 E[3..0]的相位抽头。取值范围是 0～7，默认值为 0
E[]_TIME_DELAY	字符串	指定计数器 E[3..0]的延时值。取值范围是 0～3ns，默认值为 0
ENABLE[]_COUNTER	字符串	指定 ENABLE[1..0]端口的计数器。值可以是 L0 或 L1
ENABLE_SWITCH_OVER_COUNTER	字符串	指定是否使用切换器计数器。值可以是 ON 或 OFF，默认值为 OFF
EXTCLK[]_COUNTER	字符串	指定外部时钟输出端口 EXTCLK[3..0]的外部计数器。值可以是 E0、El、E2 或 E3，默认值为 E0
EXTCLK[]_DIVIDE_BY	整数	指定外部时钟输出端口 EXTCLK[3..0]相对于输入时钟频率的整数除法因子，值必须大于 0。仅对使用的相应 EXTCLK[3..0]指定该参数值，默认值为 1
EXTCLK[]_DUTY_CYCLE	整数	指定外部时钟输出端口 EXTCLK[3..0]的占空比。默认值为 50，即 50%
EXTCLK[]_MULTIPLY_BY	整数	指定外部时钟输出端口 EXTCLK[3..0]相对于输入时钟频率的整数乘法因子，值必须大于 0。仅对使用的相应 EXTCLK[3..0]指定该参数值，默认值为 1
EXTCLK[]_PHASE_SHIFT	字符串	指定外部时钟输出端口 EXTCLK[3..0]的相位偏移
EXTCLK[]_TIME_DELAY	字符串	指定外部时钟输出端口 EXTCLK[3..0]应用的延时值，单位是 ps。该参数仅影响相应的 EXTCLK[3..0]，与 EXTCLK[3..0]_PHASE_SHIFT 参数无关。有效取值范围是－3～6ns(步长为 0.25ns)。仅在使用实时可编程接口对 PLL 重编程时使用该参数
FEEDBACK_SOURCE	字符串	指定在板上哪个时钟输出连接到 fbin 端口。如果参数 OPERATION_MODE 设置为 EXTERNAL_FEEDBACK，则需要设置该参数。值可以为 EXTCLK[]，默认值为 EXTCLK0
G[]_HIGH	整数	指定计数器 G[3..0]的高电平周期计数值。取值范围是 1～512，默认值为 1
G[]_INITIAL	整数	指定计数器 G[3..0]的初始值。取值范围是 1～512，默认值为 1
G[]_LOW	整数	指定计数器 G[3..0]的低电平周期计数值。取值范围是 1～512，默认值为 1
G[]_MODE	字符串	指定计数器 G[3..0]的操作模式。值可以是 BYPASS，ODD 或 EVEN，默认值为 BYPASS
G[]_PH	整数	指定计数器 G[3..0]的相位抽头。取值范围是 0～7，默认值为 0
G[]_TIME_DELAY	字符串	指定计数器 G[3..0]的延时值。取值范围是 0～3ns，默认值为 0
GATE_LOCK_COUNTER	整数	指定门控 Locked 信号输出端口的 20 位计数器
GATE_LOCK_SIGNAL	字符串	指定 Locked 端口是否用 20 位可编程计数器从内部被门控，以使其在初始加电期间不会振荡。取值为 NO 或 YES，默认为 NO

续表

参数名称	类型	说明
INCLK[]_INPUT_FREQUENCY	整数	指定输入时钟端口 inclk[1..0]的输入频率。编译器用 clk0 端口的频率计算 PLL 参数，也分析和报告 clkl 端口的相位偏移
INTENDED_DEVICE_FAMILY	字符串	该参数用于建模和行为仿真，默认值为 NONE
INVALID_LOCK_MULTIPLIER	整数	指定缩放因子，以半个时钟周期为单位，即时钟输出端口必须在 locked 引脚信号变为低电平之前变为失锁状态的限定时间
L[]_HIGH	整数	指定计数器 L[1..0]的高电平周期计数值。取值范围是 1～512，默认值为 1
L[]_INITIAL	整数	指定计数器 L[1..0]的初始值。取值范围是 1～512，默认值为 1
L[]_LOW	整数	指定计数器 L[1..0]的低电平周期计数值。取值范围是 1～512，默认值为 1
L[]_MODE	字符串	指定计数器 L[1..0]的操作模式。值可以是 BYPASS，ODD 或 EVEN，默认值为 BYPASS
L[]_PH	整数	指定计数器 L[1..0]的相位抽头。取值范围是 0～7，默认值为 0
L[]_TIME_DELAY	字符串	指定计数器 L[1..0]的延时值。取值范围是 0～3ns，默认值为 0
LOCK_HIGH	整数	指定在 locked 端口信号变为高电平之前，输出时钟必须被锁定的半时钟周期数，仅在使用第三方仿真器时设置
LOCK_LOW	整数	指定在 locked 端口信号变为低电平之前，输出时钟必须解除锁定的半时钟周期数，仅在使用第三方仿真器时设置
LOCK_WINDOW_UI	字符串	指定 LOCK_WINDOW_UI 设置的值，默认值为 0.05
LOOP_FILTER_C	字符串	指定环路电容的值，单位是 pF。取值范围是 5～20pF，默认值为 10
LOOP_FILTER_R	字符串	指定环路电阻的值，单位是千欧姆。取值范围是 1k～20k
LPM_HINT	字符串	用于在 VHDL 设计文件(.vhd)中指定原 Altera 特定参数，默认值为 UNUSED。
LPM_TYPE		用于识别 VHDL 设计文件(.vhd)中参数化模型(LPM)实体名称的库。
M	整数	指定计数器 M 的模，提供对内部 PLL 参数的直接访问。如果设置了该参数，则必须使用所有的高级参数。取值范围是 1～512，默认值为 0
M_INITIAL	整数	指定计数器 M 的初始值，提供对内部 PLL 参数的直接访问。如果设置了该参数，则必须使用所有的高级参数。取值范围是 1～512，默认值为 0
M_PH	整数	指定计数器 M 的相位抽头。取值范围是 0～7，默认值为 0
M_TEST_SOURCE	整数	指定计数器 M 的测试源。默认值为 5
M_TIME_DELAY	整数	指定计数器 M 的延时值。取值范围是 0～3ns，默认值为 0
M2	整数	指定计数器 M 的扩频模，提供对内部 PLL 参数的直接访问。如果设置了该参数，则必须使用所有的高级参数。取值范围是 1～512，默认值为 1
N	整数	指定计数器 N 的模，提供对内部 PLL 参数的直接访问。如果设置了该参数，则必须使用所有的高级参数。取值范围是 1～512，默认值为 1
N_TIME_DELAY	整数	指定计数器 N 的延时值。取值范围是 0～3ns，默认值为 0

续表

参数名称	类型	说明
N2	整数	指定计数器 N 的扩频模，提供对内部 PLL 参数的直接访问。如果设置了该参数，则必须使用所有的高级参数。取值范围是 1～512，默认值为 1
OPERATION_MODE	字符串	指定 PLL 的操作模式。值可以是 EXTERNAL_FEEDBACK，NO_COMPENSATION，NORMAL，ZERO_DELAY_BUFFER 或 SOURCE_SYNCHRONOUS，默认值为 NORMAL
PFD_MAX	整数	指定 PFD 引脚的最大值。默认值为 0
PFD_MIN	整数	指定 PFD 引脚的最小值。默认值为 0
PLL_TYPE	字符串	指定例化的 PLL 类型。值可以是 AUTO，ENHANCED，FAST，TOP_BOTTOM 或 LEFT_RIGHT，默认值为 AUTO
PORT_ACTIVECLOCK	字符串	指定 ACTIVECLOCK 端口的连接情况。值可以是 PORT_USED 或 PORT_UNUSED，默认值为 PORT_UNUSED
PORT_ARESET	字符串	指定 ARESET 端口的连接情况。值可以是 PORT_USED 或 PORT_UNUSED，默认值为 PORT_UNUSED
PORT_CLK[]	字符串	指定 CLK[9..0]端口的连接。值可以是 PORT_USED 或 PORT_UNUSED，默认值为 PORT_UNUSED
PORT_CLKBAD[]	字符串	指定 CLKBAD[1..0]端口的连接。值可以是 PORT_USED 或 PORT_UNUSED，默认值为 PORT_UNUSED
PORT_CLKENA[]	字符串	指定 CLKENA[5..0]端口的连接。值可以是 PORT_USED 或 PORT_UNUSED，默认值为 PORT_UNUSED
PORT_CLKLOSS	字符串	指定 CLKLOSS 端口的连接。值可以是 PORT_USED 或 PORT_UNUSED，默认值为 PORT_UNUSED
PORT_CLKSWITCH	字符串	指定 CLKSWITCH 端口的连接。值可以是 PORT_USED 或 PORT_UNUSED，默认值为 PORT_UNUSED
PORT_CONFIGUPDATE	字符串	指定 CONFIGUPDATE 端口的连接。值可以是 PORT_USED 或 PORT_UNUSED，默认值为 PORT_UNUSED
PORT_ENABLE[]	字符串	指定 ENABLE[1..0]端口的连接。值可以是 PORT_USED 或 PORT_UNUSED，默认值为 PORT_UNUSED
PORT_EXTCLK[]	字符串	指定 EXTCLK[3..0]端口的连接。值可以是 PORT_USED 或 PORT_UNUSED，默认值为 PORT_UNUSED
PORT_EXTCLKENA[]	字符串	指定 EXTCLKENA[3..0]端口的连接。值可以是 PORT_USED 或 PORT_UNUSED，默认值为 PORT_UNUSED
PORT_FBIN	字符串	指定 FBIN 端口的连接。值可以是 PORT_USED 或 PORT_UNUSED，默认值为 PORT_UNUSED
PORT_FBOUT	字符串	指定 FBOUT 端口的连接。值可以是 PORT_USED 或 PORT_UNUSED，默认值为 PORT_UNUSED
PORT_INCLK[]	字符串	指定 INCLK[1..0]端口的连接。值可以是 PORT_USED 或 PORT_UNUSED，默认值为 PORT_UNUSED
PORT_LOCKED	字符串	指定 LOCKED 端口的连接。值可以是 PORT_USED 或 PORT_UNUSED，默认值为 PORT_UNUSED
PORT_PFDENA	字符串	指定 PFDENA 端口的连接。值可以是 PORT_USED 或 PORT_UNUSED，默认值为 PORT_UNUSED

续表

参数名称	类型	说明
PORT_PHASECOUNTERSELECT	字符串	指定 PHASECOUNTERSELECT 端口的连接。值可以是 PORT_USED 或 PORT_UNUSED,默认值为 PORT_UNUSED
PORT_PHASEDONE	字符串	指定 PHASEDONE 端口的连接。值可以是 PORT_USED 或 PORT_UNUSED,默认值为 PORT_UNUSED
PORT_PHASESTEP	字符串	指定 PHASESTEP 端口的连接。值可以是 PORT_USED 或 PORT_UNUSED,默认值为 PORT_UNUSED
PORT_PHASEUPDOWN	字符串	指定 PHASEUPDOWN 端口的连接。值可以是 PORT_USED 或 PORT_UNUSED,默认值为 PORT_UNUSED
PORT_PLLENA	字符串	指定 PLLENA 端口的连接。值可以是 PORT_USED 或 PORT_UNUSED,默认值为 PORT_UNUSED
PORT_SCANACLR	字符串	指定 SCANACLR 端口的连接。值可以是 PORT_USED 或 PORT_UNUSED,默认值为 PORT_UNUSED
PORT_SCANCLK	字符串	指定 SCANCLK 端口的连接。值可以是 PORT_USED 或 PORT_UNUSED,默认值为 PORT_UNUSED
PORT_SCANCLKENA	字符串	指定 SCANCLKENA 端口的连接。值可以是 PORT_USED 或 PORT_UNUSED,默认值为 PORT_UNUSED
PORT_SCANDATA	字符串	指定 SCANDATA 端口的连接。值可以是 PORT_USED 或 PORT_UNUSED,默认值为 PORT_UNUSED
PORT_SCANDONE	字符串	指定 SCANDONE 端口的连接。值可以是 PORT_USED 或 PORT_UNUSED,默认值为 PORT_UNUSED
PORT_SCANREAD	字符串	指定 SCANREAD 端口的连接。值可以是 PORT_USED 或 PORT_UNUSED,默认值为 PORT_UNUSED
PORT_SCANWRITE	字符串	指定 SCANWRITE 端口的连接。值可以是 PORT_USED 或 PORT_UNUSED,默认值为 PORT_UNUSED
PORT_SCLKOUT[]	字符串	指定 SCLKOUT[1..0]端口的连接。值可以是 PORT_USED 或 PORT_UNUSED,默认值为 PORT_UNUSED
PORT_VCOOVERRANGE	字符串	指定 VCOOVERRANGE 端口的连接。值可以是 PORT_USED 或 PORT_UNUSED,默认值为 PORT_NUSED
PORT_VCOUNDERRANGE	字符串	指定 VCOUNDERRANGE 端口的连接。值可以是 PORT_USED 或 PORT_UNUSED,默认值为 PORT_UNUSED
PRIMARY_CLOCK	字符串	指定 PLL 的主参考时钟。值可以是 INCLK0 或 INCLK1,默认值为 INCLK0
QUALIFY_CONF_DONE	字符串	指定配置是否完成。值可以是 ON 或 OFF,默认值为 OFF
SCAN_CHAIN	字符串	指定扫描链的长度。值可以是 LONG 或 SHORT,默认值为 LONG。值为 LONG 时,扫描链是长度为 10 的计数器;值为 SHORT 时,扫描链是长度为 6 的计数器
SCLKOUT[]_PHASE_SHIFT	字符串	指定 sclkout[1..0]端口的相位偏移,单位是 ps。最大值是 VCO 周期的 7/8。VCO 相位抽头与时钟输出 clk[1..0]共用,且必须具有相同的相位数量且要小于一个 VCO 周期。在 LVDS 模式中,该参数的默认值为 0
SELF_RESET_ON_GATED_LOSS_LOCK	字符串	指定是否使用自复位门控失锁属性。值可以是 ON 或 OFF,默认值为 OFF

续表

参数名称	类型	说明
SELF_RESET_ON_LOSS_LOCK	字符串	指定是否使用自复位失锁属性。值可以是 ON 或 OFF，默认值为 OFF
SKIP_VCO	整数	指定是否忽略 VCO。值可以是 ON 或 OFF，默认值为 OFF
SPREAD_FREQUENCY	整数	指定扩频的调制频率，单位是 ps。默认值为 0
SS	整数	指定扩频计数器的模，提供对内部 PLL 参数的直接访问。如果指定了该参数，则必须使用所有高级参数。取值范围是 1～32768，默认值为 1
SWITCH_OVER_COUNTER	整数	指定切换电路启动后用于切换输入时钟的时钟周期数。取值范围是 0～31，默认值为 0
SWITCH_OVER_ON_GATED_LOCK	字符串	指定是否门控锁定条件可以启动切换器。值可以是 ON 或 OFF，默认值为 OFF
SWITCH_OVER_ON_LOSSCLK	字符串	指定是否失锁条件可以启动切换器。值可以是 ON 或 OFF，默认值为 OFF
SWITCH_OVER_TYPE	字符串	指定切换类型。值可以是 AUTO 或 MANUAL，默认值为 AUTO
USING_FBMIMICBIDIR_PORT	字符串	指定是否使用 fbmimicbidir 端口。值可以是 ON 或 OFF，默认值为 OFF
VALID_LOCK_MULTIPLIER	整数	指定缩放因子，以半个时钟周期为单位。在时钟锁定引脚值变为高电平之前，时钟输出端口必须已经被锁定所限定的时间
VCO_CENTER	整数	指定 VCO 引脚的中心值。仅用于仿真
VCO_DIVIDE_BY	整数	指定 VCO 引脚的整数除法因子。默认值为 0。如果参数 VCO_FREQUENCY_CONTROL 设置为 MANUAL_PHASE，则将 VCO 频率指定为相移步进值，是 VCO 周期的 1/8
VCO_FREQUENCY_CONTROL	字符串	指定 VCO 引脚的频率控制方法。值可以是 AUTO，MANUAL_FREQUENCY 或 MANUAL_PHASE.，默认值为 AUTO。当值为 AUTO 时，自动设置 VCO 频率，而忽略参数 VCO_MULTIPLY_BY 和 VCO_DIVIDE_BY 的值；当值为 MANUAL_FREQUENCY 时，VCO 频率为输入频率的倍数；当值为 MANUAL_PHASE 时，VCO 频率作为相移步进值使用
VCO_MAX	整数	指定 VCO 引脚的最大值。仅用于仿真
VCO_MIN	整数	指定 VCO 引脚的最小值。仅用于仿真
VCO_MULTIPLY_BY	整数	指定 VCO 引脚的整数乘法因子。默认值为 0
VCO_PHASE_SHIFT_STEP	整数	指定 VCO 引脚的相位偏移。默认值为 0
VCO_POST_SCALE	整数	指定 VCO 操作范围。VCO 的后缩放除法器值为 1 或 2，默认值为 1
WIDTH_CLOCK	整数	指定时钟宽度。不同系列 FPGA 的该参数值不同：对于 Cyclone II 和 Cyclone IV 系列，值为 5；对于 Stratix III 系列，值为 10；对于其他系列，值为 6。默认值为 6

5. 例化和仿真

如图 5-26 所示，由仿真结果可知，复位信号 areset 为高电平 1，系统复位；当 areset 为低电平 0 时，系统进入正常工作状态，在经历一小段时间的 IP 核自身初始化过程之后，输出

时钟 c0 为输入时钟 inclk0 的两倍频，输出时钟 c1 为输入时钟 inclk0 的四倍频，同时 locked 信号输出高电平 1。

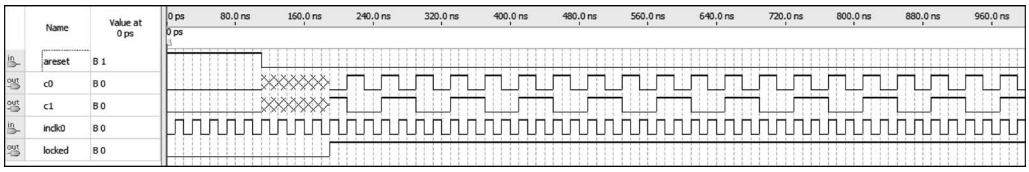


图 5-26　ALTPLL IP 核仿真结果