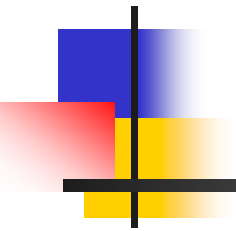

# 第7章

# 宏功能模块应用

# 7.1 计数器LPM模块调用

## 7.1.1 计数器模块文本的调用与参数设置

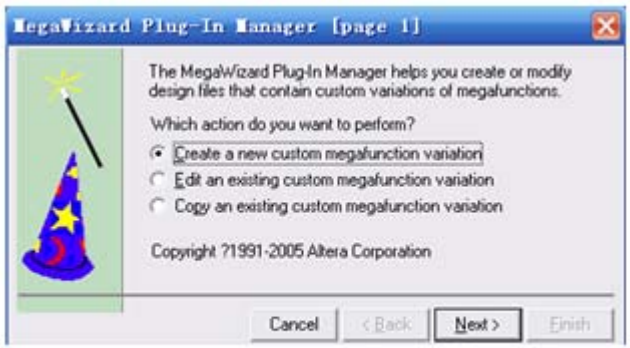


图 7-1　定制新的宏功能块

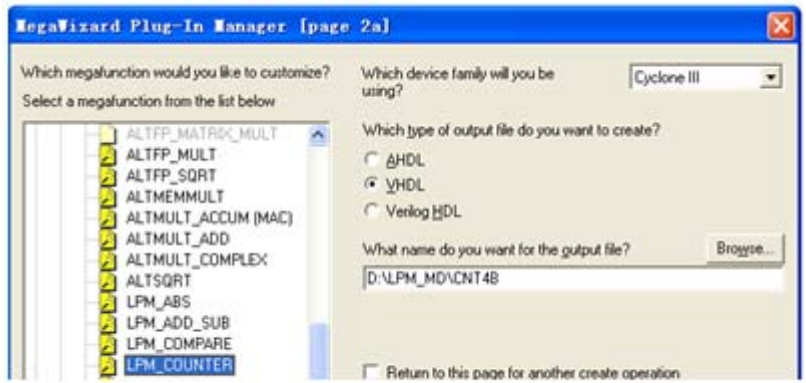


图 7-2　LPM 宏功能块设定

# 7.1 计数器LPM模块调用

## 7.1.1 计数器模块文本的调用与参数设置

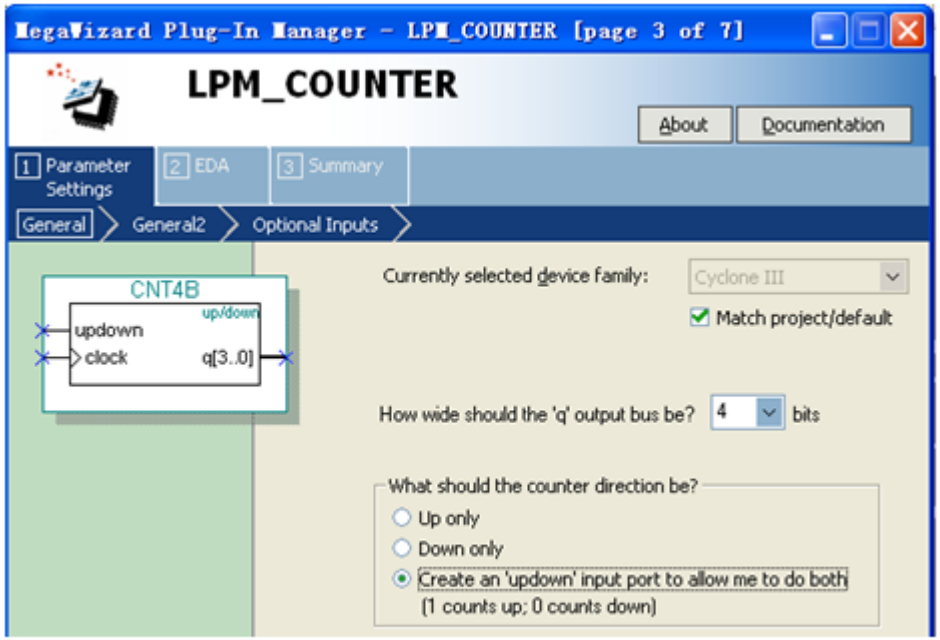


图 7-3 设 4 位可加减计数器

# 7.1 计数器LPM模块调用

## 7.1.1 计数器模块文本的调用与参数设置

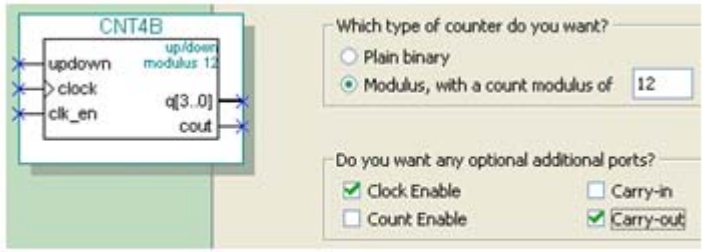


图 7-4 设定计数器，含时钟使能和进位输出

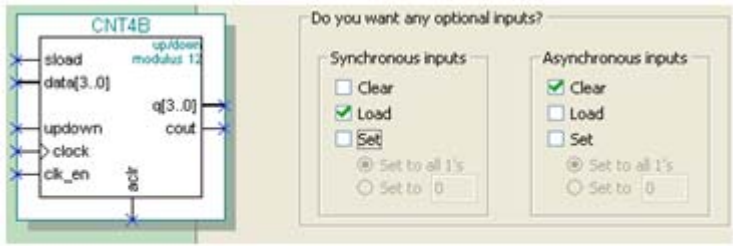


图 7-5 加入 4 位并行数据预置功能

【例 7-1】

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY lpm;                                    --打开 LPM 库
USE lpm.all;                                    --打开 LPM 程序包
ENTITY CNT4B IS
        --异步清 0、时钟使能、时钟输入、同步预置数加载控制、加减控制
  PORT (aclr, clk_en, clock, sload, updown  : IN STD_LOGIC ;
        data      : IN STD_LOGIC_VECTOR (3 DOWNTO 0);  -- 4 位预置数
        cout      : OUT STD_LOGIC ;                          --进位输出
        q         : OUT STD_LOGIC_VECTOR (3 DOWNTO 0) );--计数器输出
END CNT4B;
ARCHITECTURE SYN OF cnt4b IS
        SIGNAL sub_wire0 : STD_LOGIC ;
        SIGNAL sub_wire1 : STD_LOGIC_VECTOR (3 DOWNTO 0);
        COMPONENT lpm_counter  --以下是参数传递说明语句
GENERIC(lpm_direction,lpm_port_updown ,lpm_type : STRING;--参数定义
                lpm_modulus, lpm_width : NATURAL );       --参数定义
PORT (sload, clk_en, aclr, clock, updown  : IN STD_LOGIC ;
                                  cout     : OUT STD_LOGIC ;
                                  q : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
                        data      : IN STD_LOGIC_VECTOR (3 DOWNTO 0));
        END COMPONENT;
```

接下页

# 7.1 计数器LPM模块调用

## 7.1.1 计数器模块文本的调用与参数设置

```
BEGIN
  cout <= sub_wire0;  q <= sub_wire1(3 DOWNTO 0);
 lpm_counter_component : lpm_counter GENERIC MAP (  --参数传递例化语句
   lpm_direction => "UNUSED",                       --单方向计数参数未用
                 lpm_modulus => 12,                  --定义模 12 计数器
                 lpm_port_updown => "PORT_USED",     --使用加减计数
                 lpm_type => "LPM_COUNTER",          --计数器类型
                 lpm_width => 4    )                 --计数位宽
  PORT MAP (sload=>sload,clk_en=>clk_en,aclr=>aclr, clock => clock,
        data => data,updown => updown,cout=>sub_wire0,q => sub_wire1);
END SYN;
```

# 7.1 计数器LPM模块调用

## 7.1.1 计数器模块文本的调用与参数设置

【例 7-2】

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY CNT4BIT IS
   PORT (CLK,RST,ENA ,SLD,UD : IN std_logic;
         DIN  : IN std_logic_vector(3 DOWNTO 0);  COUT : OUT std_logic;
         DOUT   : OUT std_logic_vector(3 DOWNTO 0));
END ENTITY CNT4BIT;
ARCHITECTURE translated OF CNT4BIT IS
    COMPONENT CNT4B
        PORT (aclr,clk_en,clock,sload,updown  : IN STD_LOGIC ;
                data     : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
                cout     : OUT STD_LOGIC ;
                q        : OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
END COMPONENT;
 BEGIN
    U1 : CNT4B PORT MAP (sload => SLD, clk_en => ENA,aclr => RST,
        cout=>COUT, clock=>CLK, data=>DIN, updown=>UD, q=>DOUT);
  END ARCHITECTURE translated;
```

# 7.1 计数器LPM模块调用

## 7.1.2 创建工程与仿真测试

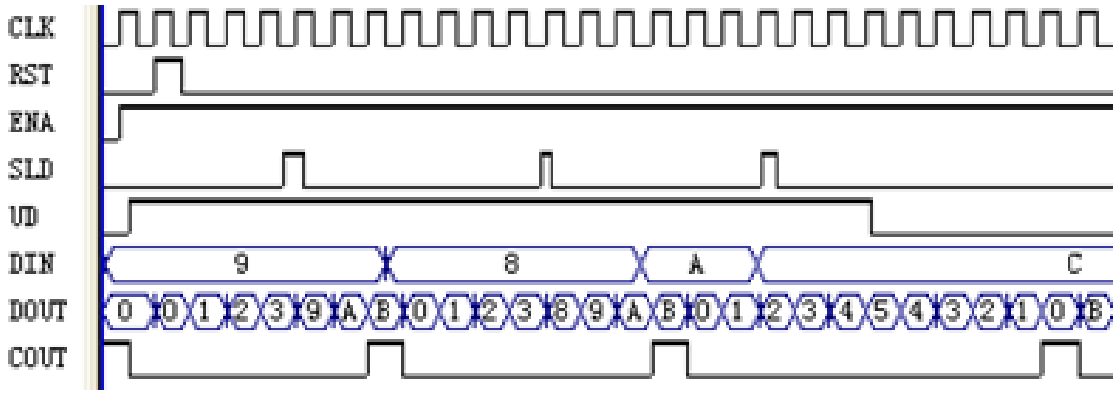


图 7-6 CNT4BIT 的仿真波形

# 7.1 计数器LPM模块调用

## 7.1.2 创建工程与仿真测试

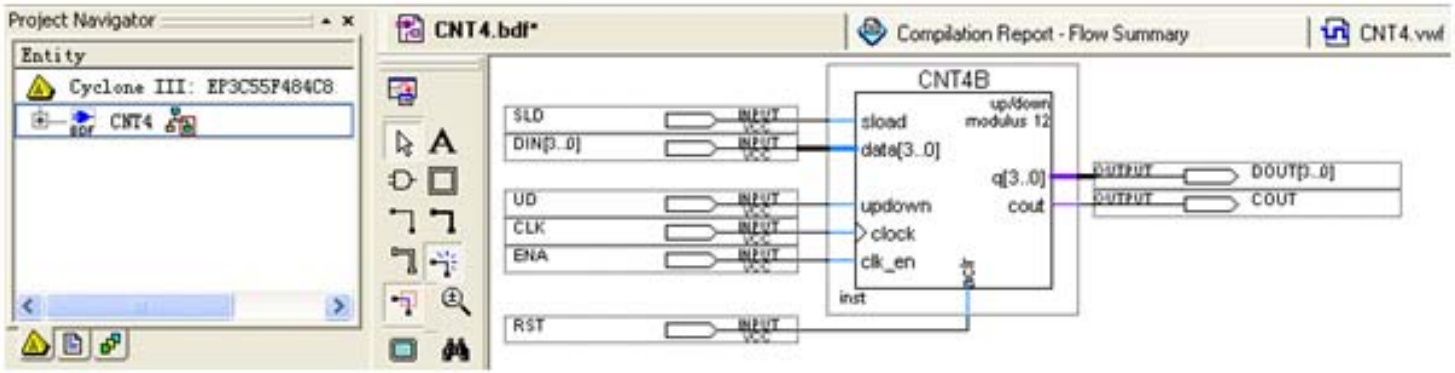


图 7-7 原理图输入设计

# 7.2 利用属性控制乘法器的构建

## 7.1.2 创建工程与仿真测试

【例 7-3】

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL ;
USE IEEE.STD_LOGIC_ARITH.ALL ;
ENTITY MULT8 IS
PORT (A1,B1 : IN  SIGNED(7 DOWNTO 0) ;
      R1 : OUT SIGNED(15 DOWNTO 0));
END ;
ARCHITECTURE bhv OF MULT8 IS
 attribute multstyle : string;
 attribute multstyle of R1 : signal is "LOGIC";--使用逻辑资源构建乘法器
BEGIN
   R1 <= A1 * B1  ;
END bhv;
```


图 7-8　例 7-3 的编译报告

| | |
|---|---|
| Family | Cyclone III |
| Device | EP3C55F484C8 |
| Timing Models | Final |
| Met timing requirements | N/A |
| Total logic elements | 0 / 55,856 ( 0 % ) |
| Total combinational functions | 0 / 55,856 ( 0 % ) |
| Dedicated logic registers | 0 / 55,856 ( 0 % ) |
| Total registers | 0 |
| Total pins | 32 / 328 ( 10 % ) |
| Total virtual pins | 0 |
| Total memory bits | 0 / 2,396,160 ( 0 % ) |
| Embedded Multiplier 9-bit elements | 1 / 312 ( < 1 % ) |
| Total PLLs | 0 / 4 ( 0 % ) |

图 7-9　例 7-3 的编译报告

【例 7-4】

```
 attribute multstyle of R1 : signal is "DSP";--使用 DSP 模块构建乘法器
```

# 7.3 LPM 随机存储器的设置和调用

## 7.3.1 存储器初始化文件

### 1. .mif格式文件

【例 7-5】

```
DEPTH=128; 数据深度，即存储的数据个数
WIDTH=8;                  ：输出数据宽度
ADDRESS_RADIX = HEX;    ：地址数据类型，HEX 表示选择 16 进制数据类型
DATA_RADIX = HEX;         ：存储数据类型，HEX 表示选择 16 进制数据类型
CONTENT                   ：此为关键词
BEGIN                     ：此为关键词
0000      :      0080;
0001      :      0086;
0002      :      008C;
…（数据略去）
007E      :     0073;
007F      :     0079;
END;
```

# 7.3 LPM 随机存储器的设置和调用

## 7.3.1 存储器初始化文件

### 1. .mif格式文件

DATA7X8.mif

| Addr | +0 | +1 | +2 | +3 | +4 | +5 | +6 | +7 |
|---|---|---|---|---|---|---|---|---|
| 00 | B0 | 86 | 8C | 92 | 98 | 9E | A5 | AA |
| 08 | B0 | B6 | BC | C1 | C6 | CB | D0 | D5 |
| 10 | DA | DE | E2 | E6 | EA | ED | F0 | F3 |
| 18 | F5 | F8 | FA | FB | FD | FE | FE | FF |
| 20 | FF | FF | FE | FE | FD | FB | FA | F8 |
| 28 | F5 | F3 | F0 | ED | EA | E6 | E2 | DE |
| 30 | DA | D5 | D0 | CB | C6 | C1 | BC | B6 |
| 38 | B0 | AA | A5 | 9E | 98 | 92 | 8C | 86 |
| 40 | 7F | 79 | 73 | 6D | 67 | 61 | 5A | 55 |
| 48 | 4F | 49 | 43 | 3E | 39 | 34 | 2F | 2A |
| 50 | 25 | 21 | 1D | 19 | 15 | 12 | 0F | 0C |
| 58 | 0A | 07 | 05 | 04 | 02 | 01 | 01 | 00 |
| 60 | 00 | 00 | 01 | 01 | 02 | 04 | 05 | 07 |
| 68 | 0A | 0C | 0F | 12 | 15 | 19 | 1D | 21 |
| 70 | 25 | 2A | 2F | 34 | 39 | 3E | 43 | 49 |
| 78 | 4F | 55 | 5A | 61 | 67 | 6D | 73 | 79 |

图 7-10 mif文件编辑窗

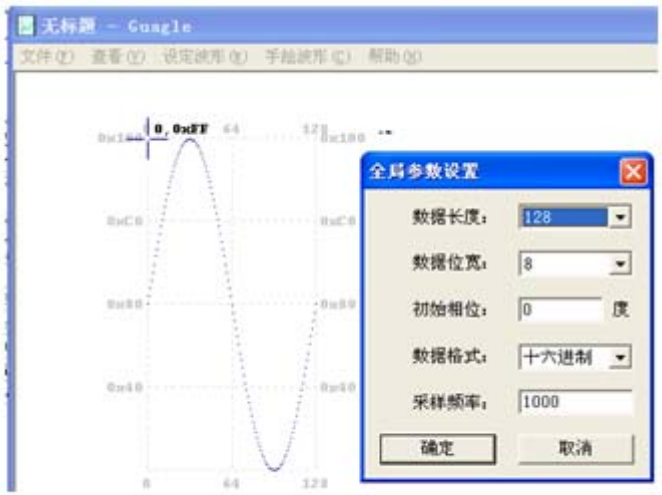


图 7-11 利用 mif 生成器生成 mif 正弦波文件


图 7-12 mif文件

### 2. .hex格式文件

# 7.3 LPM 随机存储器的设置和调用

## 7.3.2 LPM_RAM的设置和调用

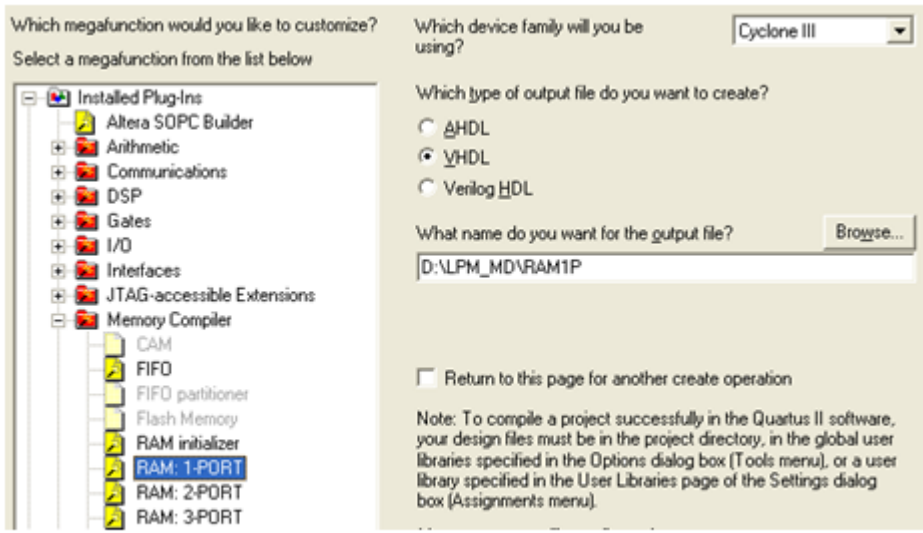


图 7-13 调用单口 LPM RAM

# 7.3 LPM 随机存储器的设置和调用

## 7.3.2 LPM_RAM的设置和调用

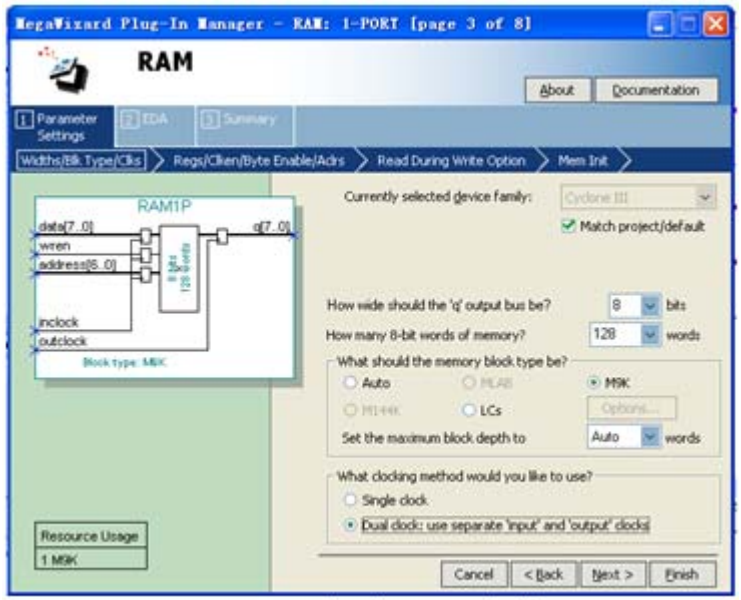


图 7-14 设定 RAM 参数

# 7.3 LPM 随机存储器的设置和调用

## 7.3.2 LPM_RAM的设置和调用

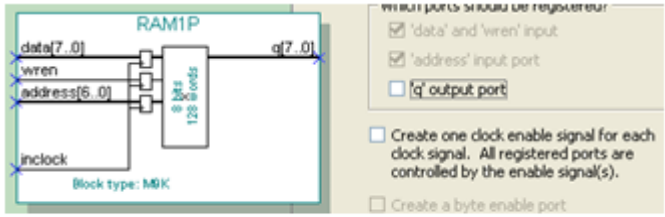


图 7-15 设定 RAM 仅输入时钟控制

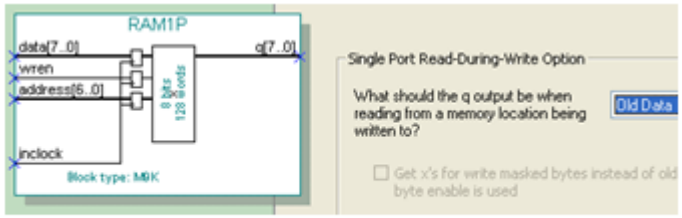


图 7-16 设定在写入同时读出原数据：Old Data

# 7.3 LPM 随机存储器的设置和调用

## 7.3.2 LPM_RAM的设置和调用

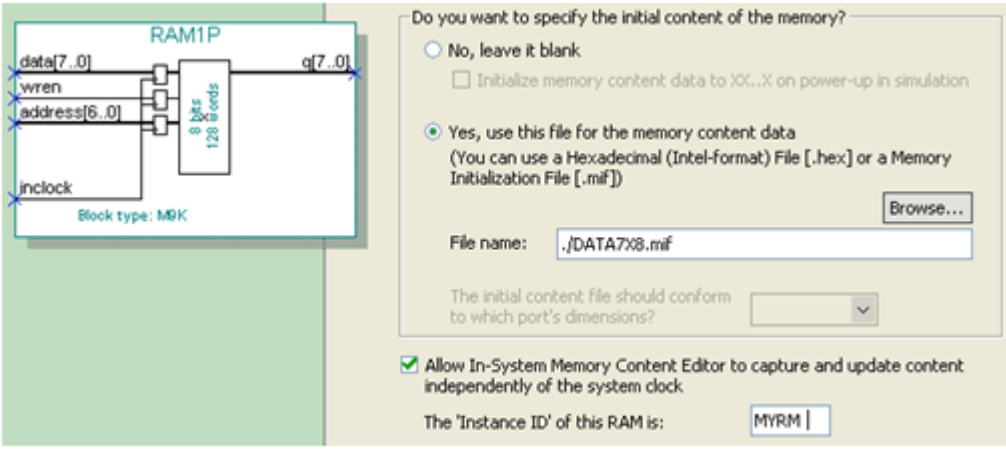


图 7-17 设定初始化文件和允许在系统编辑

# 7.3 LPM 随机存储器的设置和调用

## 7.3.2 LPM_RAM的设置和调用

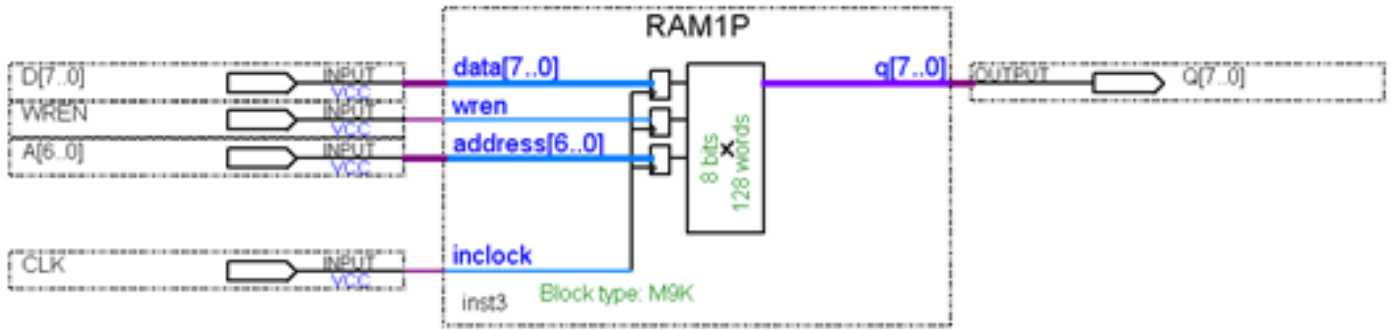


图 7-18 在原理图上连接好的 RAM 模块

# 7.3 LPM 随机存储器的设置和调用

## 7.3.3 仿真测试RAM宏模块

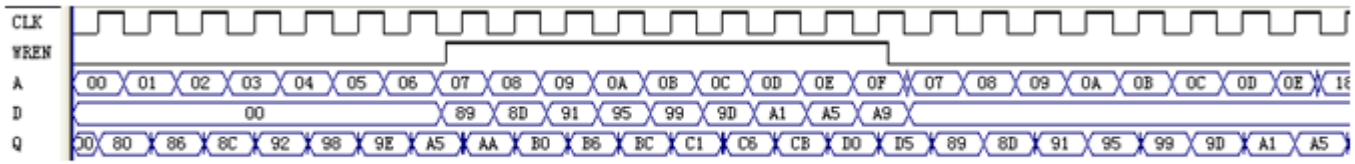


图 7-19 的 RAM 的仿真波形

【例 7-6】

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL ;
USE IEEE.STD_LOGIC_ARITH.ALL ; --此程序包包含转换函数 CONV_INTEGER(A)
USE IEEE.STD_LOGIC_UNSIGNED.ALL ; --此程序包包含算符重载函数
ENTITY RAM78 IS
PORT (CLK,WREN : IN STD_LOGIC ; --定义时钟和写允许控制
        A : IN STD_LOGIC_VECTOR(6 DOWNTO 0) ; --定义 RAM 的 7 位地址输入端口
      DIN : IN STD_LOGIC_VECTOR(7 DOWNTO 0) ; --定义 RAM 的 8 位数据输入端口
       Q : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)); --定义 RAM 的 8 位数据输出端口
END ;
ARCHITECTURE bhv OF RAM78 IS
TYPE G_ARRAY IS ARRAY(0 TO 127) OF STD_LOGIC_VECTOR(7 DOWNTO 0) ;
SIGNAL MEM : G_ARRAY;  --定义信号 MEM 的数据类型是用户新定义的类型 G_ARRAY
BEGIN
  PROCESS (CLK)       BEGIN
     IF  RISING_EDGE(CLK)  THEN
       IF  WREN='1' THEN          --如果时钟有上升沿出现，且写使能为高电平，则
         MEM(CONV_INTEGER(A))<= DIN;--RAM 数据口的数据被写入指定地址的单元
       END IF;      END IF;
IF (FALLING_EDGE(CLK)) THEN  Q<=MEM(CONV_INTEGER(A));--读出存储器中的数据
     END IF;
  END PROCESS ;
 END BHV;
```

# 7.3 LPM 随机存储器的设置和调用

## 7.3.5 数据类型定义语句

### 1. 限定性数组型数据类型定义

```
TYPE 数组名 IS ARRAY(数组范围) OF 基本数据类型 ;

TYPE stb IS ARRAY (7 DOWNTO 0) of STD_LOGIC ;

TYPE MATRIX IS ARRAY (127 DOWNTO 0) of STD_LOGIC_VECTOR(7 DOWNTO 0) ;

TYPE G_ARRAY IS ARRAY(0 TO 127) OF STD_LOGIC_VECTOR(7 DOWNTO 0) ;
SIGNAL MEM : G_ARRAY;
```

# 7.3 LPM 随机存储器的设置和调用

## 7.3.5 数据类型定义语句

### 2. 非限定性数组型数据类型定义

```
TYPE 数组名 IS ARRAY (数组下标名 RANGE <>)OF 数据类型 ;
```

```
Type bit is ('0','1');
Type bit_vector is array(natural rang<>) of bit;
```

# 7.3 LPM 随机存储器的设置和调用

## 7.3.5 数据类型定义语句

### 3. 枚举型数据类型定义

```
TYPE BOOLEAN IS (FALSE,TRUE) ;
```

```
TYPE my_logic IS ( '1' ,'Z' ,'U' ,'0' ) ;
SIGNAL s1 : my_logic ;
s1 <= 'Z' ;
```

TYPE 数据类型名 IS 数据类型定义表述

```
TYPE week IS (sun,mon,tue,wed,thu,fri,sat) ;
```

```
TYPE x is (low, high);
TYPE data_bus IS ARRAY (0 TO 7, x)of BIT ;
```

```
TYPE m_state IS ( st0,st1,st2,st3,st4,st5 ) ;
SIGNAL present_state,next_state : m_state ;
```

# 7.3 LPM 随机存储器的设置和调用

## 7.3.5 数据类型定义语句

### 4. 枚举型子类型数据类型定义

```
SUBTYPE  子类型名 IS 基本数据类型 RANGE 约束范围;

SUBTYPE digits IS INTEGER RANGE 0 to 9 ;
```

# 7.3 LPM 随机存储器的设置和调用

## 7.3.6 存储器配置文件属性定义和结构设置

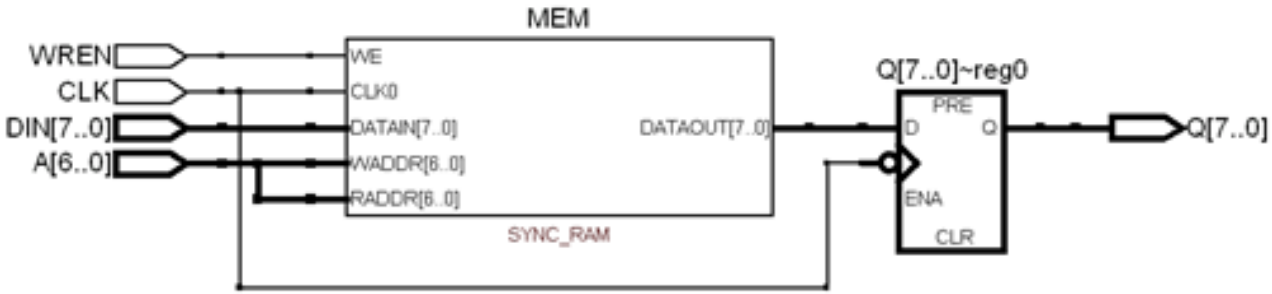


图 7-20 例 7-6 的 RAM78 的 RTL 图

# 7.3 LPM 随机存储器的设置和调用

## 7.3.6 存储器配置文件属性定义和结构设置

| | |
|---|---|
| Family | Cyclone III |
| Device | EP3C5E144C8 |
| Timing Models | Final |
| Met timing requirements | N/A |
| Total logic elements | 1,344 / 5,136 ( 26 % ) |
| Total combinational functions | 832 / 5,136 ( 16 % ) |
| Dedicated logic registers | 1,032 / 5,136 ( 20 % ) |
| Total registers | 1032 |
| Total pins | 25 / 95 ( 26 % ) |
| Total virtual pins | 0 |
| Total memory bits | 0 / 423,936 ( 0 % ) |
| Embedded Multiplier 9-bit elements | 0 / 46 ( 0 % ) |
| Total PLLs | 0 / 2 ( 0 % ) |

图 7-21 例 7-6 的编译报告

| | |
|---|---|
| Family | Cyclone III |
| Device | EP3C5E144C8 |
| Timing Models | Final |
| Met timing requirements | N/A |
| Total logic elements | 0 / 5,136 ( 0 % ) |
| Total combinational functions | 0 / 5,136 ( 0 % ) |
| Dedicated logic registers | 0 / 5,136 ( 0 % ) |
| Total registers | 0 |
| Total pins | 25 / 95 ( 26 % ) |
| Total virtual pins | 0 |
| Total memory bits | 1,024 / 423,936 ( < 1 % ) |
| Embedded Multiplier 9-bit elements | 0 / 46 ( 0 % ) |
| Total PLLs | 0 / 2 ( 0 % ) |

图 7-22 例 7-7 的编译报告

# 7.3 LPM 随机存储器的设置和调用

## 7.3.6 存储器配置文件属性定义和结构设置

【例 7-7】

```
ARCHITECTURE bhv OF RAM78 IS
TYPE G_ARRAY IS ARRAY(0 TO 127) OF STD_LOGIC_VECTOR(7 DOWNTO 0) ;
SIGNAL MEM : G_ARRAY;
 attribute ram_init_file : string;
 attribute ram_init_file of MEM :
 SIGNAL IS  "data7x8.mif";
BEGIN
```

# 7.4 LPM_ROM的定制和使用示例

## 7.4.1 LPM_ROM的定制调用和测试

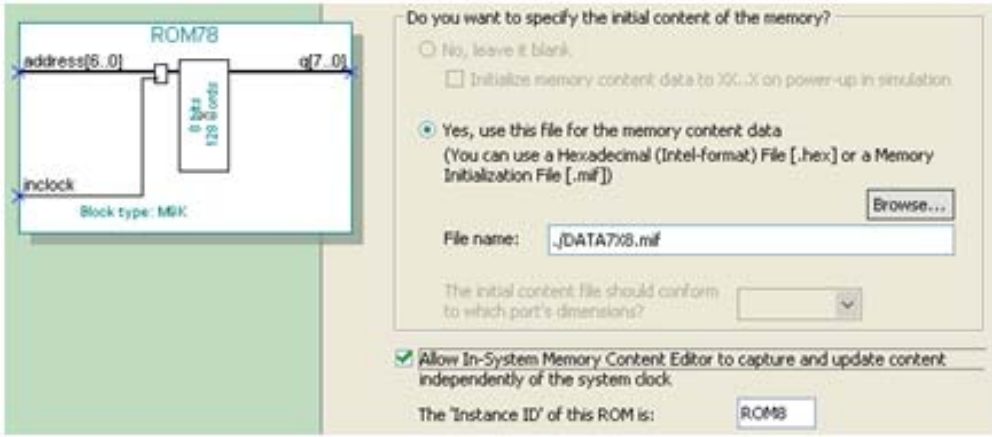


图 7-23 加入初始化配置文件并允许在系统访问 ROM 内容

# 7.4 LPM_ROM的定制和使用示例

## 7.4.2 简易正弦信号发生器设计

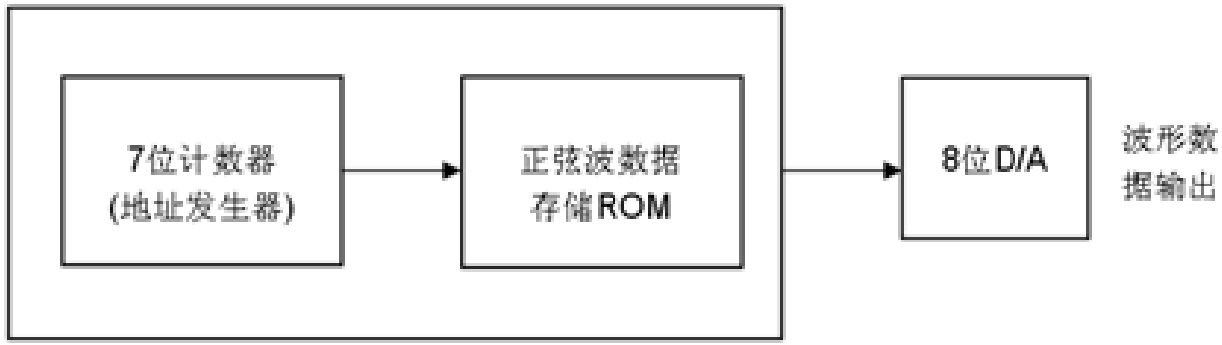


图 7-24 正弦信号发生器结构框图

# 7.4 LPM_ROM的定制和使用示例

## 7.4.2 简易正弦信号发生器设计

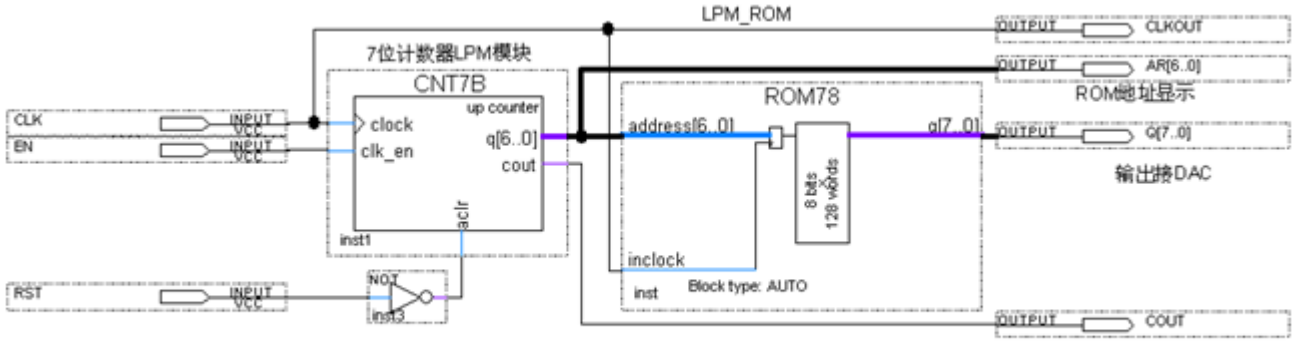


图 7-25　正弦信号发生器电路原理图

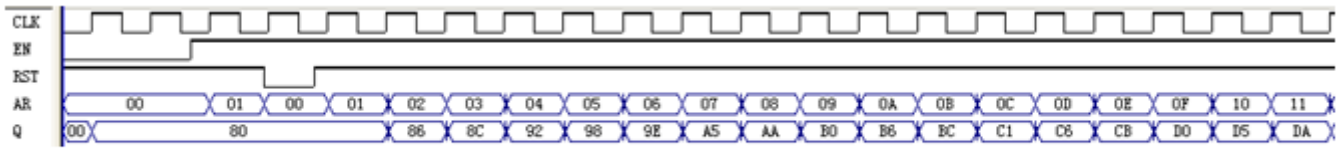


图 7-26 图 7-25 电路仿真波形

# 7.4 LPM_ROM的定制和使用示例

## 7.4.3 正弦信号发生器硬件实现和测试

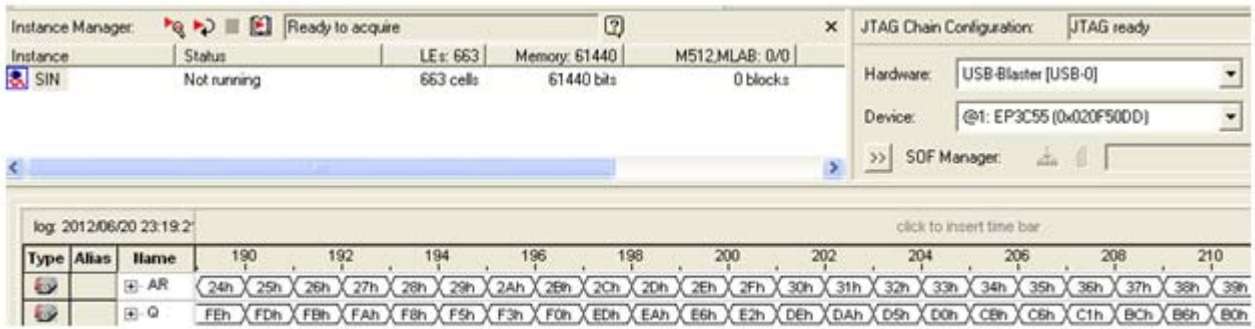


图 7-27 正弦信号发生器数据输出的 SignalTapII 实时测试图

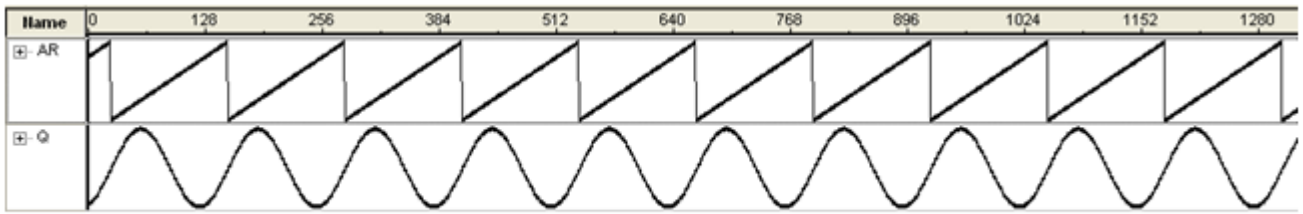


图 7-28 正弦信号发生器的 SignalTapII 的波形显示图

# 7.5 在系统存储器数据读写编辑器应用

（1）打开在系统存储单元编辑窗口。

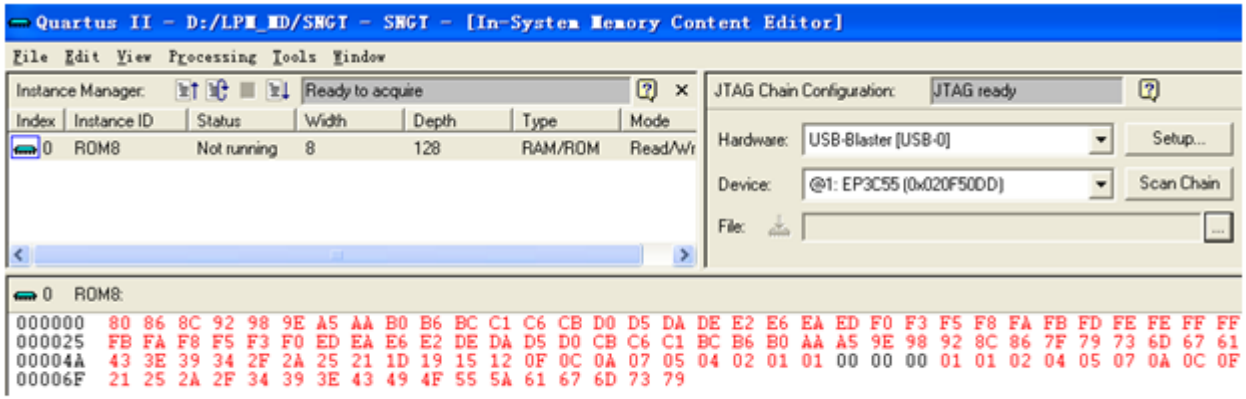


图 7-29 In-System Memory Content Editor 编辑窗

# 7.5 在系统存储器数据读写编辑器应用

（2）读取ROM中的数据。

（3）写数据。

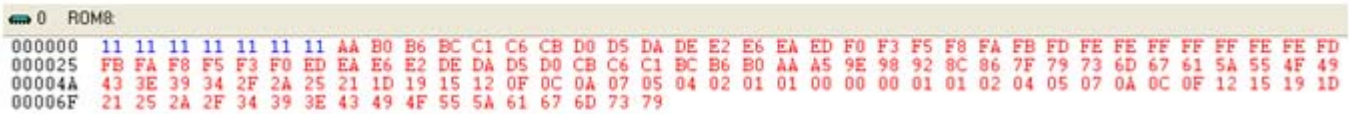


图 7-30 从 FPGA 中的 ROM 读取波形数据并编辑数据

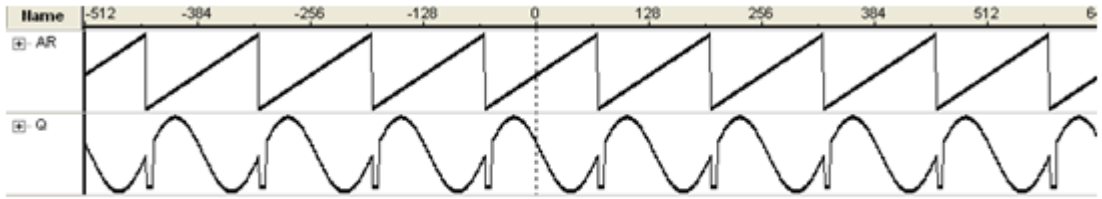


图 7-31 下载编辑数据后的 SignalTap II 采样波形

（4）输入输出数据文件。

# 7.6 LPM嵌入式锁相环调用

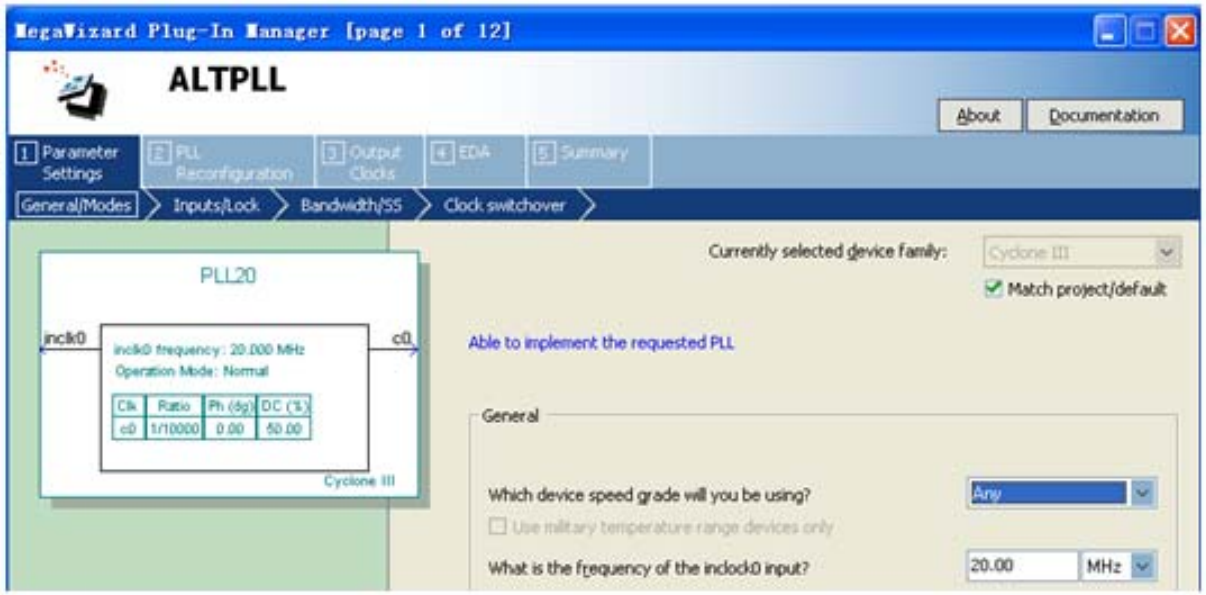


图 7-32 选择输入参考时钟 inclk0 为 20MHz

# 7.6 LPM嵌入式锁相环调用


图 7-33 选择控制信号

# 7.6 LPM嵌入式锁相环调用

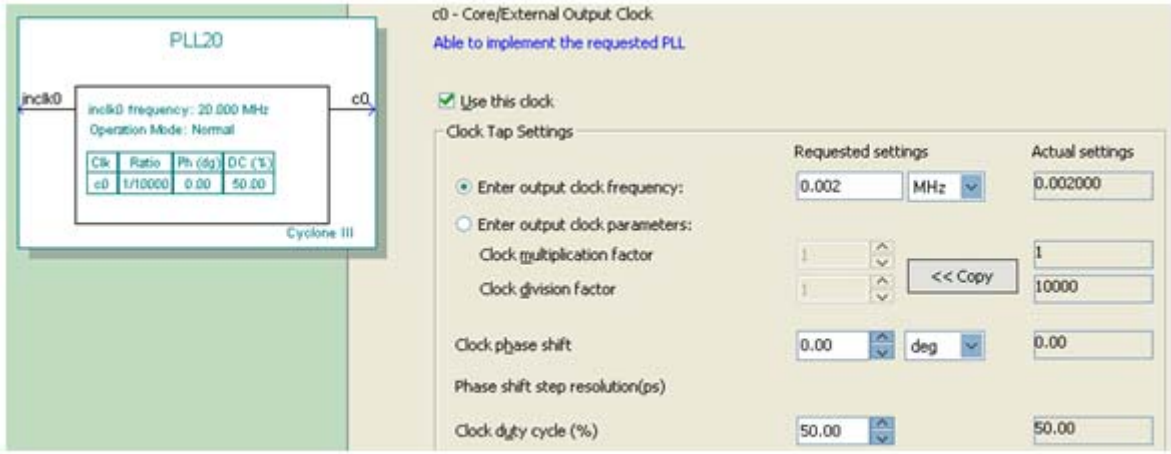


图 7-34 选择 c0 的输出频率为 30MHz

# 7.6 LPM嵌入式锁相环调用

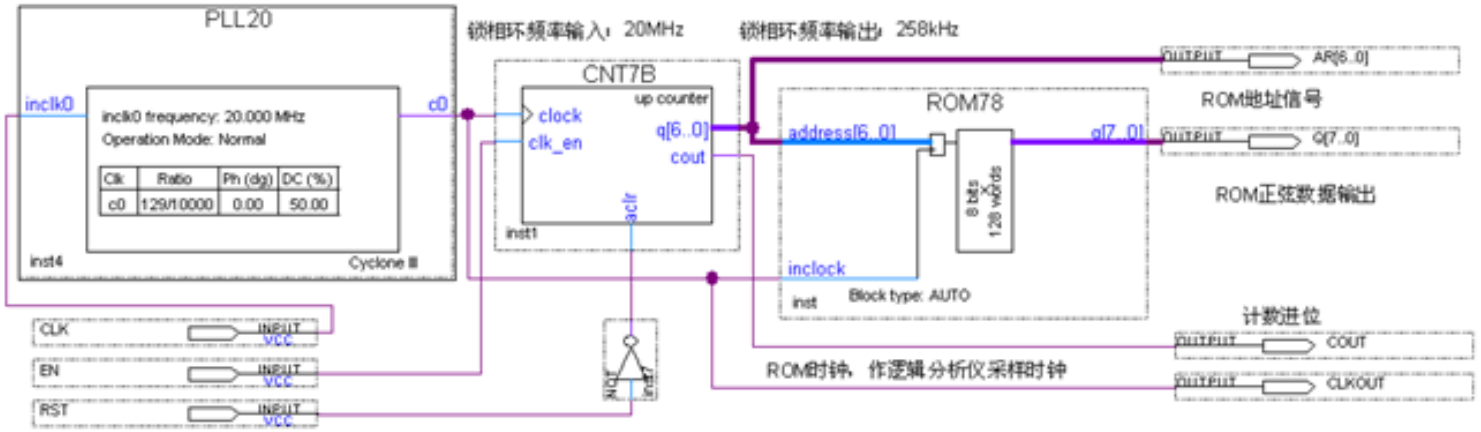


图 7-35 采用嵌入式锁相环作时钟的正弦信号发生器电路图

# 7.7 In-System Sources and Probes Editor使用方法

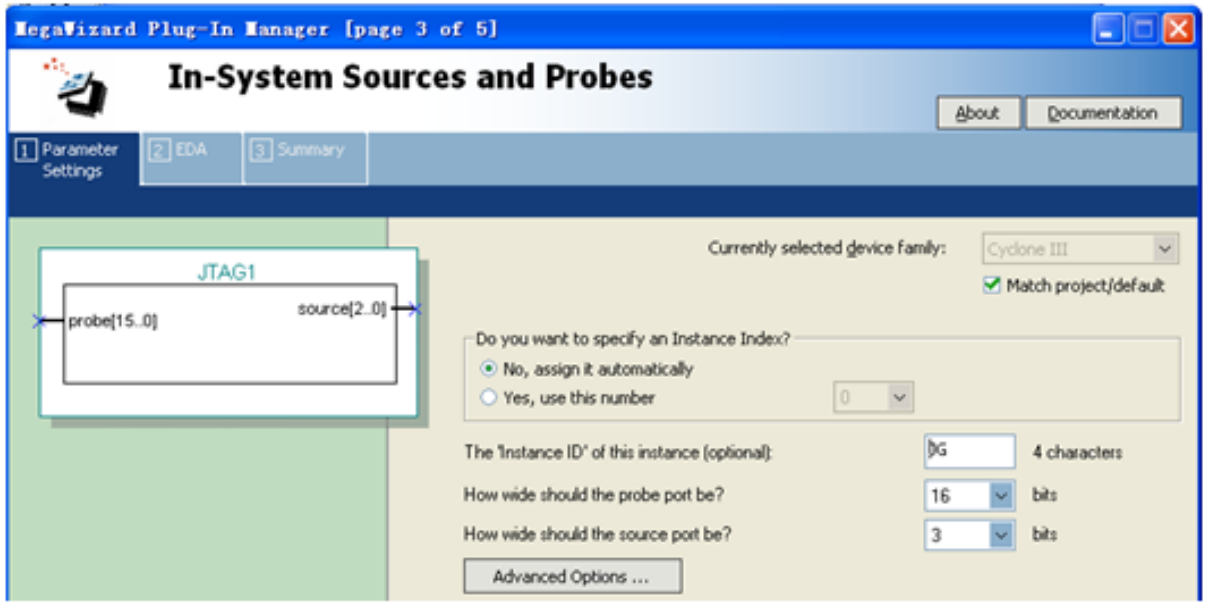


图 7-36 为 In-System Sources and Probes 模块设置参数

# 7.7 In-System Sources and Probes Editor使用方法

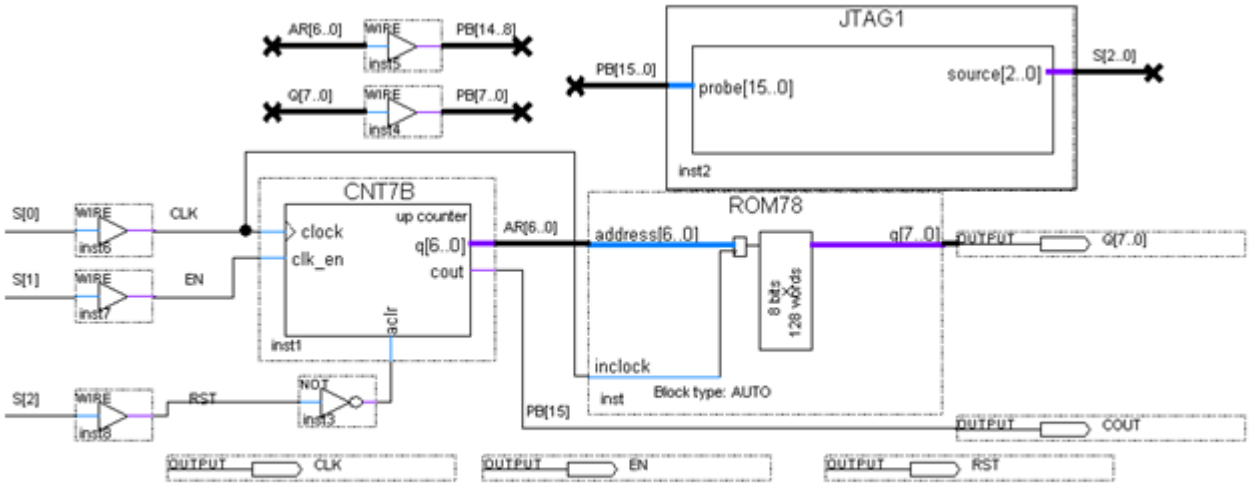


图 7-37 在双十进制计数器设计电路中加入 In-System Sources and Probes 测试模块

# 7.7 In-System Sources and Probes Editor使用方法

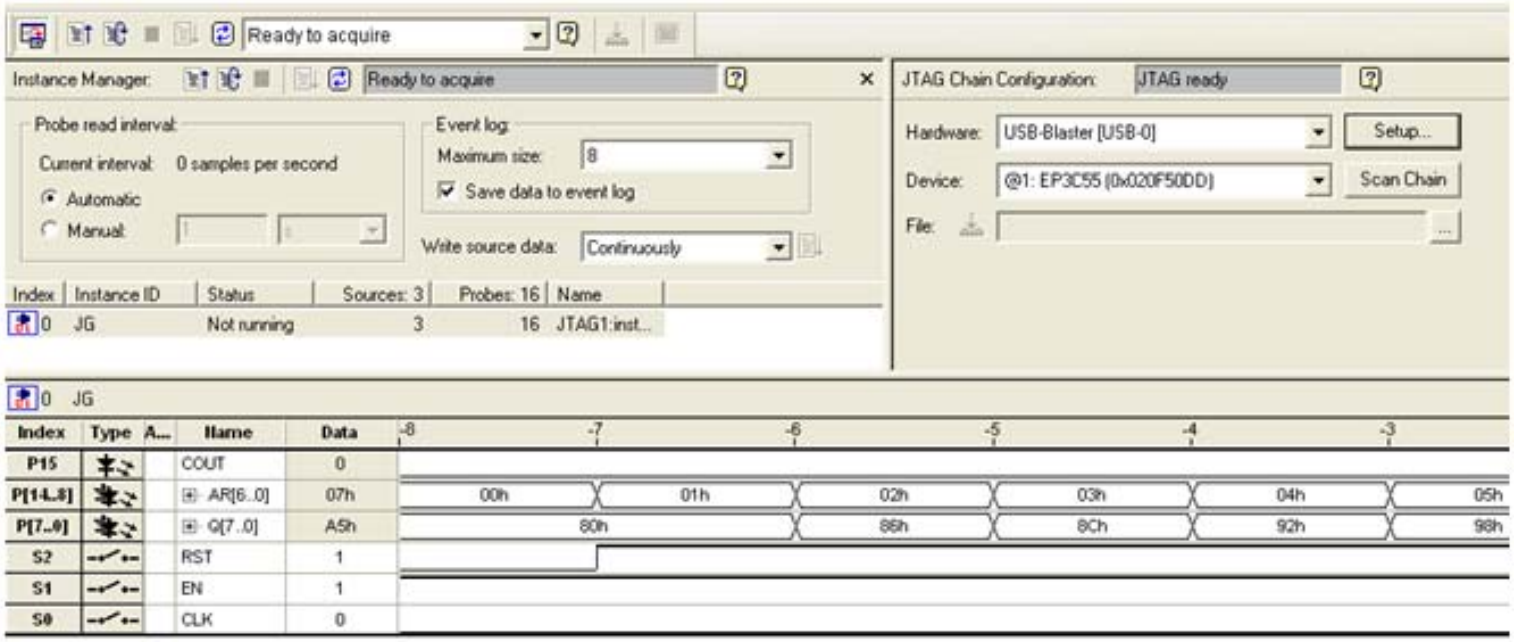


图 7-38 In-System Sources and Probes Editor 的测试情况

# 7.8 NCO核数控振荡器使用方法

（1）定制NCO。

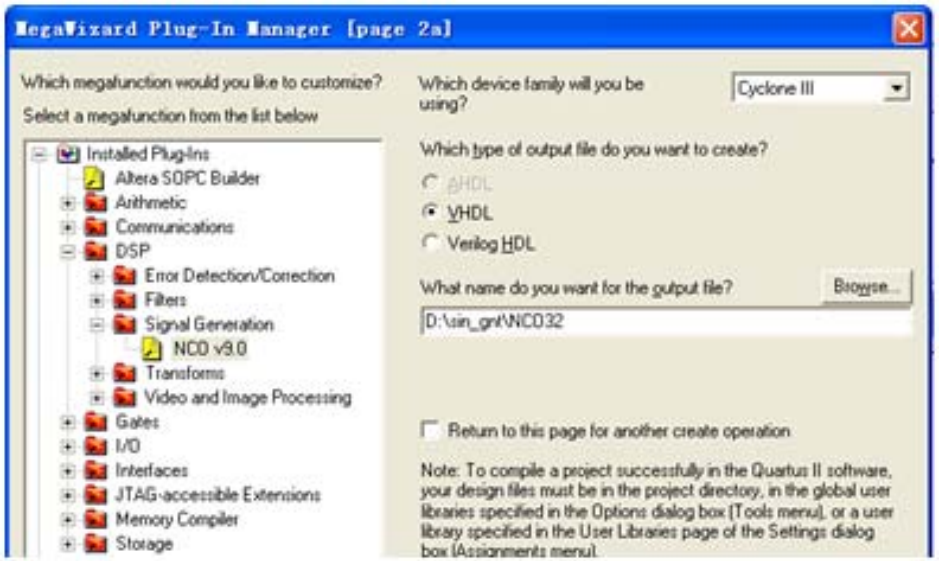


图 7-39 打开 Core 设置管理窗口选择 NCO 核

# 7.8 NCO核数控振荡器使用方法

（2）进入Core文件生成选择窗。


图 7-40 开始进入 Core 文件生成选择窗口

# 7.8 NCO核数控振荡器使用方法

（3）设置参数。

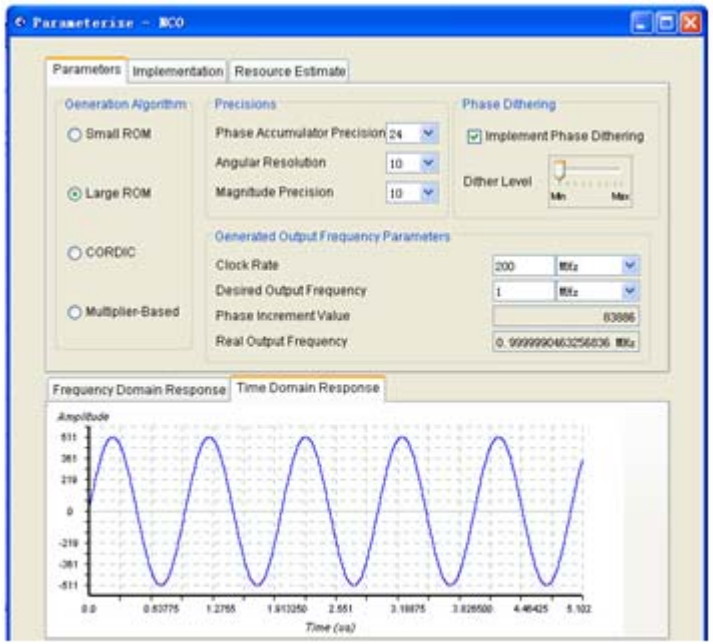


图 7-41 设置 NCO 参数

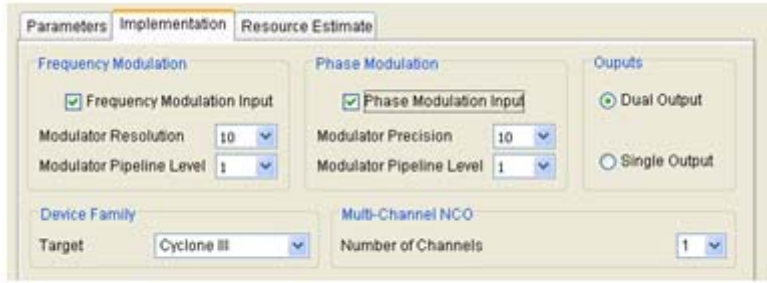


图 7-42 设置 NCO 参数

# 7.8 NCO核数控振荡器使用方法

（4）生成仿真文件。

（5）加入IP授权文件。

# 7.8 NCO核数控振荡器使用方法

（6）选择目标器件，然后对生成的模块进行编译及功能检测。

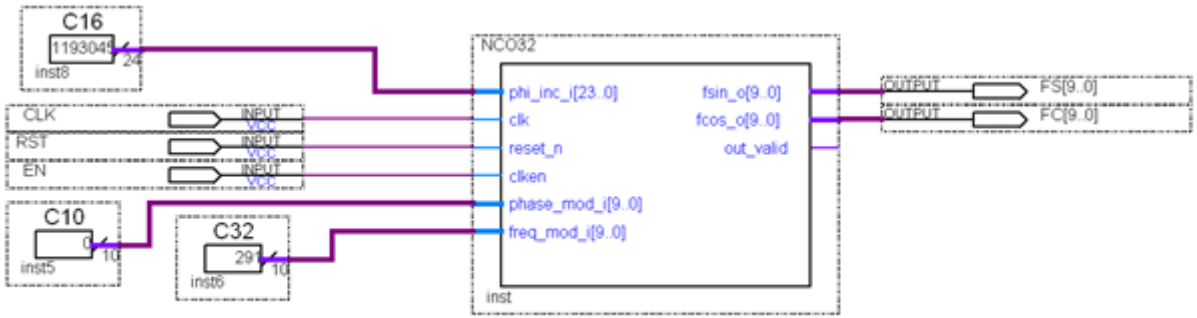


图 7-43 测试 NCO 的电路

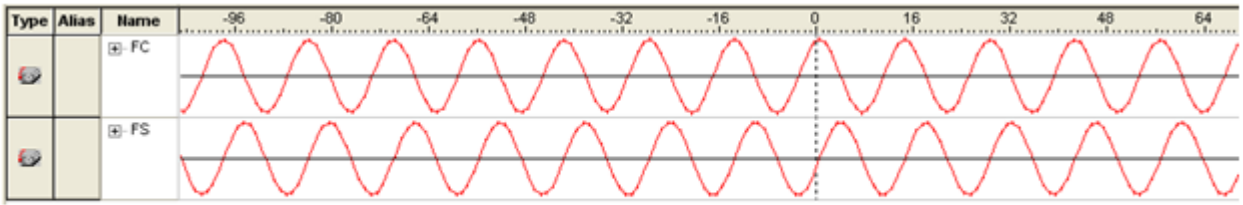


图 7-44 当前 NCO 的逻辑分析仪测试波形

# 7.9 FIR核使用方法

$$H(z)=\sum_{n=0}^{N-1}h(n)z^{-n} \qquad (7\text{-}1)$$

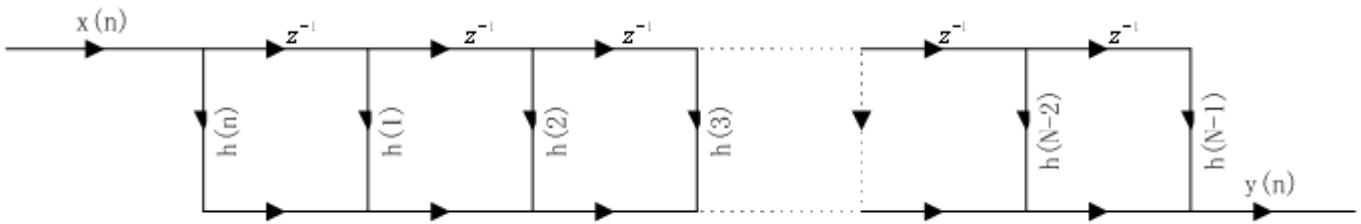


图 7-45 直接型 FIR 滤波器结构

$$Y(n)=\sum_{m=0}^{N-1}h(m)x(n-m) \qquad (7\text{-}2)$$

# 7.9 FIR核使用方法

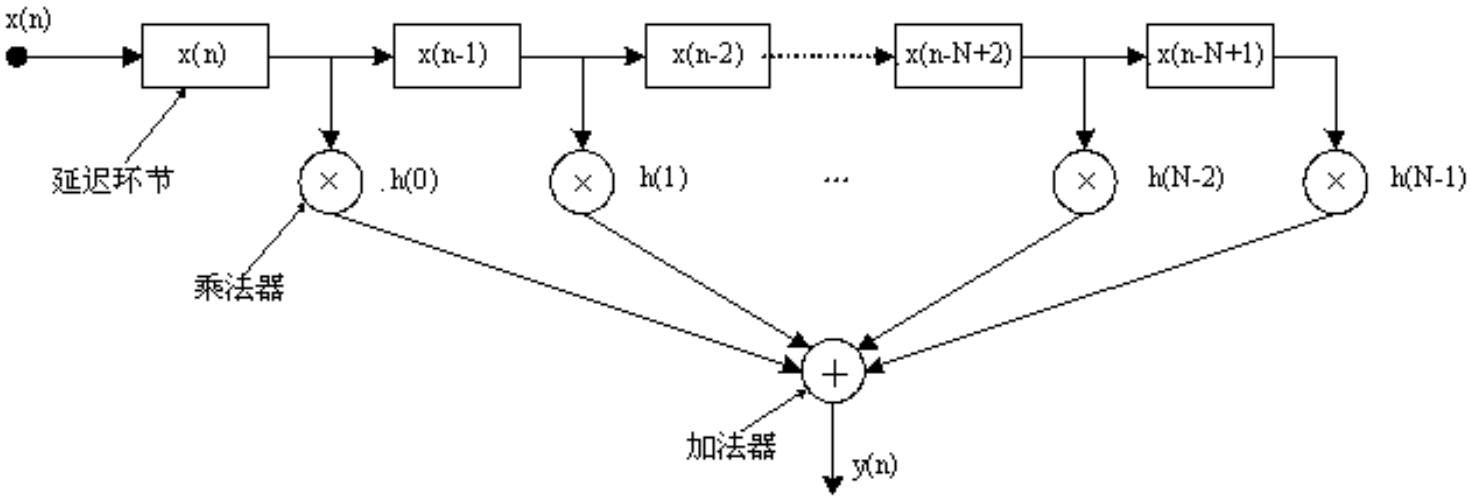


图 7-46 直接型 FIR 实现结构


图 7-47 FIR 滤波器设计示意

# 7.10 DDS实现原理与应用

## 7.10.1 DDS原理

$$S_{out} = A\sin\omega t = A\sin(2\pi f_{out} t) \tag{7-3}$$

$$\theta = 2\pi f_{out} t \tag{7-4}$$

$$\Delta\theta = 2\pi f_{out} T_{clk} = \frac{2\pi f_{out}}{f_{clk}} \tag{7-5}$$

$$\frac{B_{\Delta\theta}}{2^N} = \frac{f_{out}}{f_{clk}}, \quad B_{\Delta\theta} = 2^N \cdot \frac{f_{out}}{f_{clk}} \tag{7-6}$$

$$S_{out} = A\sin(\theta_{k-1} + \Delta\theta) = A\sin\left[\frac{2\pi}{2^N} \cdot (B_{\theta_{k-1}} + B_{\Delta\theta})\right] = Af_{sin}(B_{\theta_{k-1}} + B_{\Delta\theta}) \tag{7-7}$$

$$B_{\theta_{k-1}} \approx \frac{\theta_{k-1}}{2\pi} \cdot 2^N \tag{7-8}$$

# 7.10 DDS实现原理与应用

## 7.10.1 DDS原理

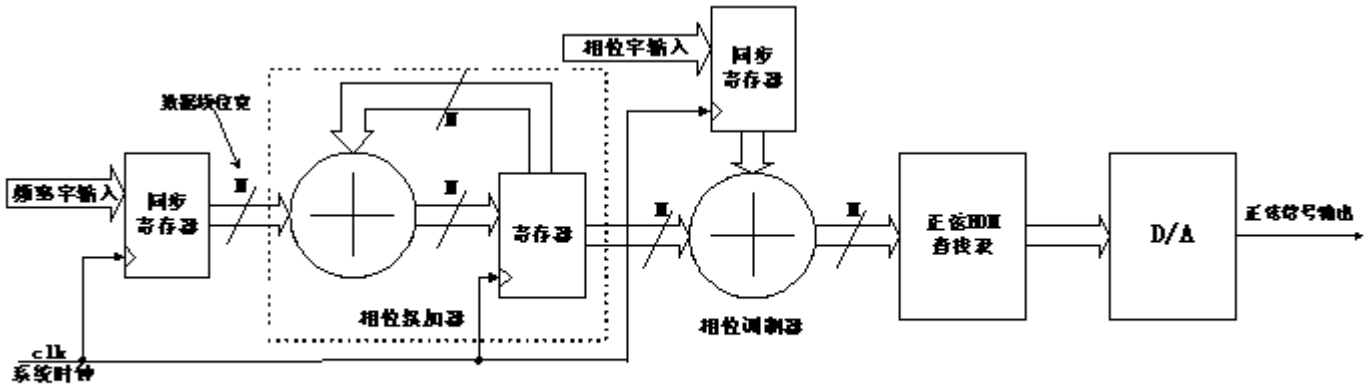


图 7-48 基本 DDS 结构

$$f_{out} = \frac{B_{\Delta\theta}}{2^N} \cdot f_{clk} \qquad (7\text{-}9)$$

$$f_{out} = \frac{f_{clk}}{2^N} \qquad (7\text{-}10)$$

# 7.10 DDS实现原理与应用

## 7.10.2 DDS信号发生器设计示例

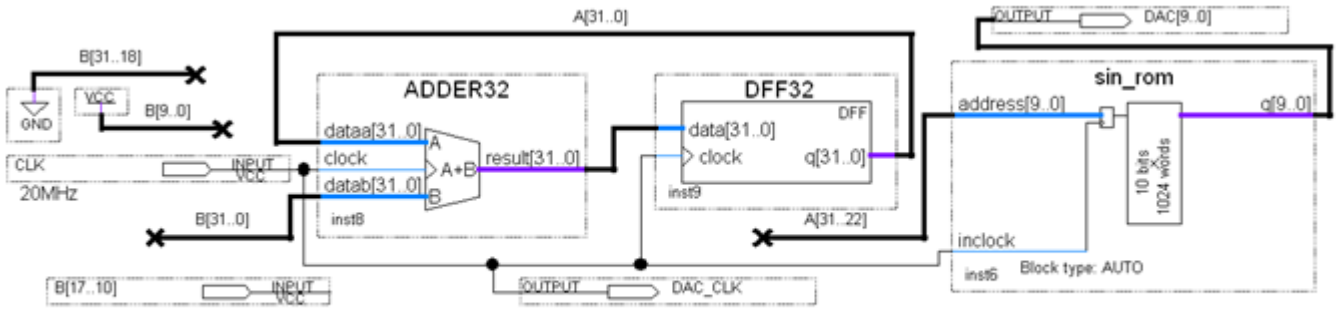


图 7-49　DDS 信号发生器电路顶层原理图

$$f_{out} = \frac{B[31..0]}{2^{32}} \cdot f_{clk} \qquad (7\text{-}11)$$

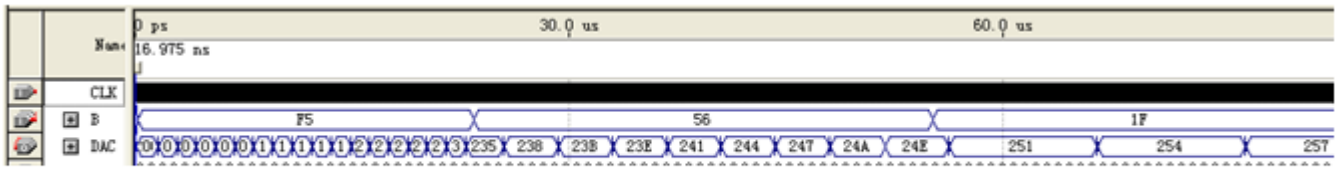


图 7-50　图 7-49 的仿真波形

# 实验与设计

**7-1**．查表式硬件运算器设计

**7-2** 正弦信号发生器设计

**7-3 DDS**正弦信号发生器设计

# 实验与设计

## 7-4. 简易逻辑分析仪设计

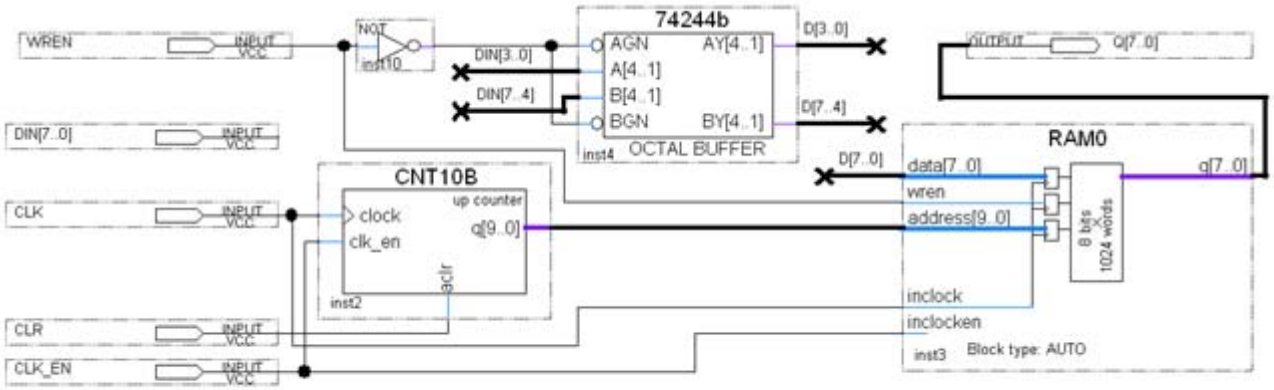


图 7-51 逻辑数据采样电路顶层设计

# 实验与设计

## 7-4. 简易逻辑分析仪设计

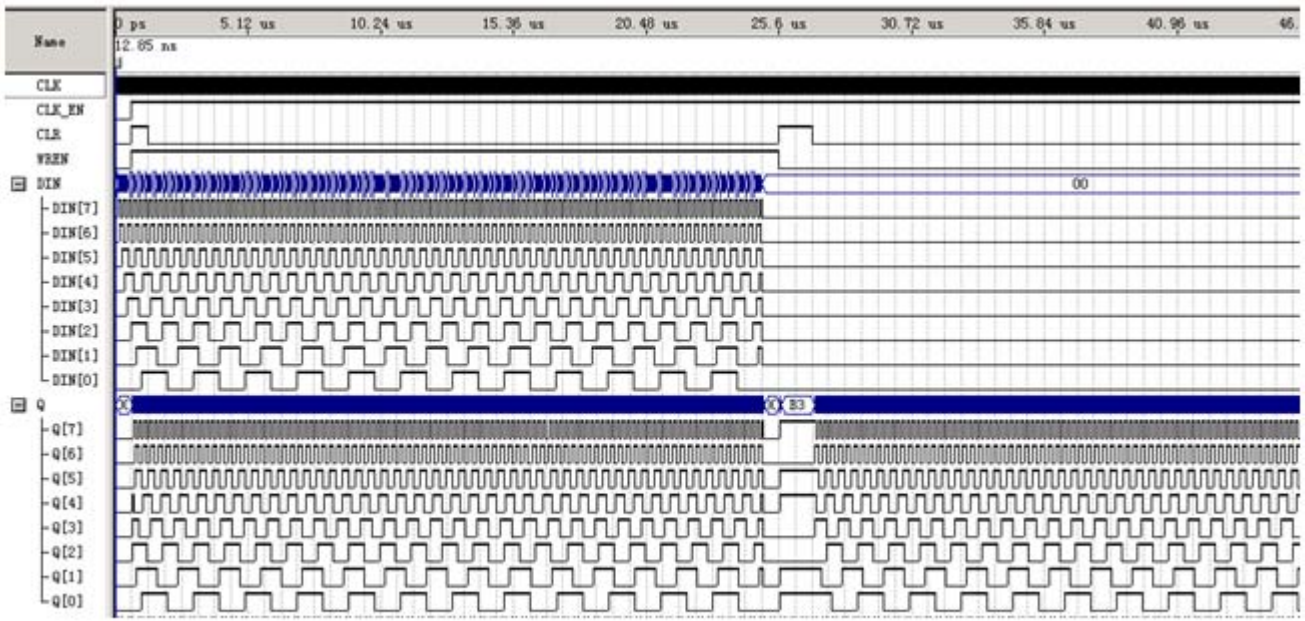


图 7-52 逻辑数据采样电路时序仿真波形

# 实验与设计

## 7-5 移相信号发生器设计

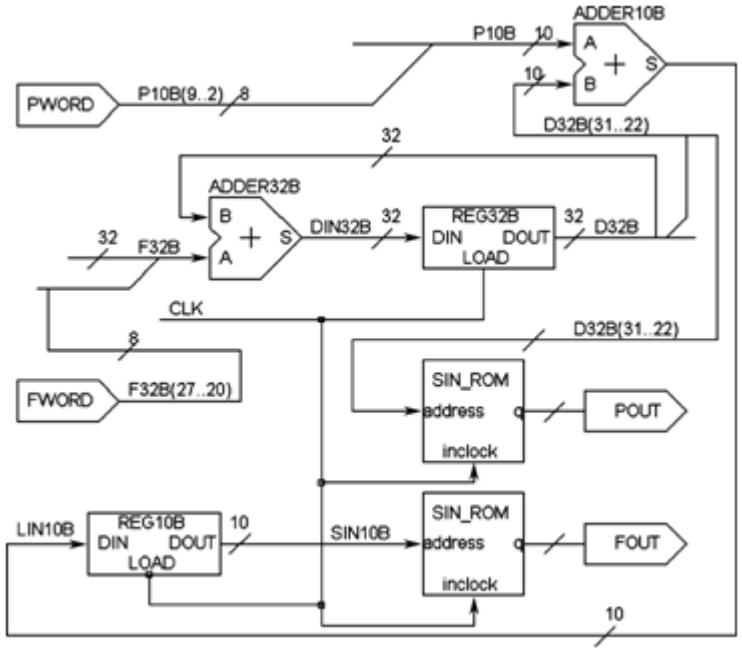


图 7-53 全数字移相信号发生器电路模型图

【例 7-8】

```
LIBRARY ieee;  --图象显示顶层程序
USE ieee.std_logic_1164.all;
ENTITY vgaV IS
 Port ( clk50MHz :  IN  STD_LOGIC;
        hs, vs, r, g, b :  OUT  STD_LOGIC );
END vgaV;
ARCHITECTURE modelstru OF vgaV IS
 component vga640480                --VGA显示控制模块
 PORT(clk : IN STD_LOGIC;
      rgbin : IN STD_LOGIC_VECTOR(2 downto 0);
      hs, vs, r, g, b : OUT STD_LOGIC;
      hcntout, vcntout : OUT STD_LOGIC_VECTOR(9 downto 0)   );
end component;
component imgrom                   --图象数据ROM，数据线3位；地址线12位
 PORT(inclock : IN STD_LOGIC;
      address : IN STD_LOGIC_VECTOR(11 downto 0);
      q : OUT STD_LOGIC_VECTOR(2 downto 0)  );
end component;
signal  rgb :  STD_LOGIC_VECTOR(2 downto 0);
signal  clk25MHz : std_logic;
signal  romaddr :  STD_LOGIC_VECTOR(11 downto 0);
signal  hpos, vpos  :  std_logic_vector(9 downto 0);
```

接下页

# 实验与设计

## 7-6 VGA简单图像显示控制模块设计

```
BEGIN
romaddr <= vpos(5 downto 0) & hpos(5 downto 0);
process(clk50MHz) begin
if clk50MHz'event and clk50MHz='1' then clk25MHz<=not clk25MHz; end if;
end process;
i_vga640480 : vga640480 PORT MAP(clk => clk25MHz, rgbin => rgb, hs => hs,
vs => vs, r => r, g => g, b => b, hcntout => hpos, vcntout => vpos);
i_rom : imgrom PORT MAP(inclock => clk25MHz, address => romaddr, q => rgb);
end;
```

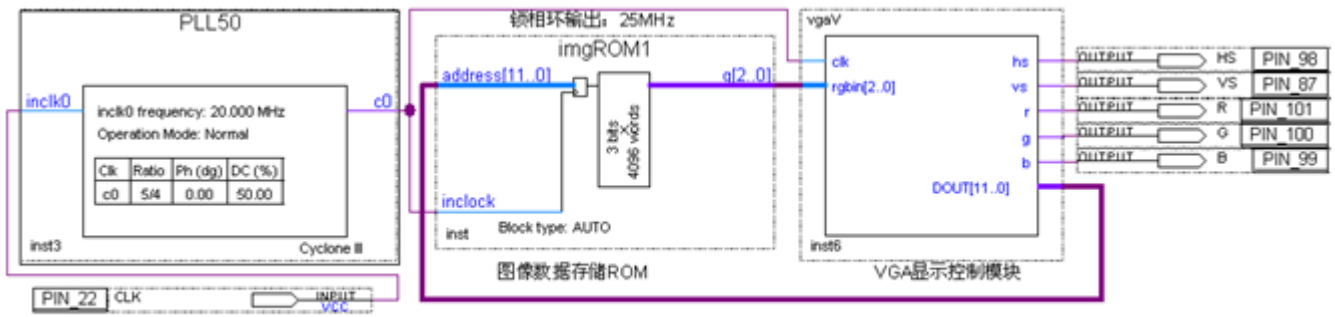


图 7-54　VGA 图像显示控制模块原理图