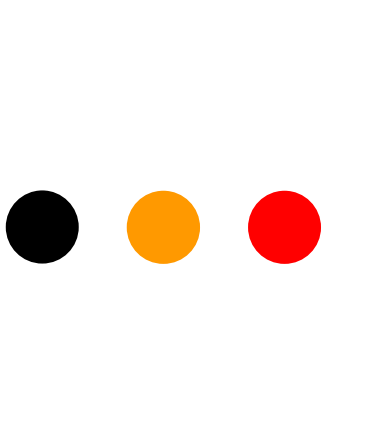

第7章

LPM宏模块的应用

7.1 计数器LPM宏模块调用

7.1.1 计数器LPM模块文本代码的调用

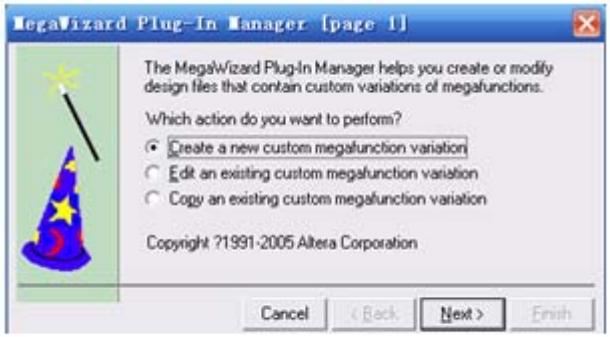

图 7-1　定制新的宏功能块

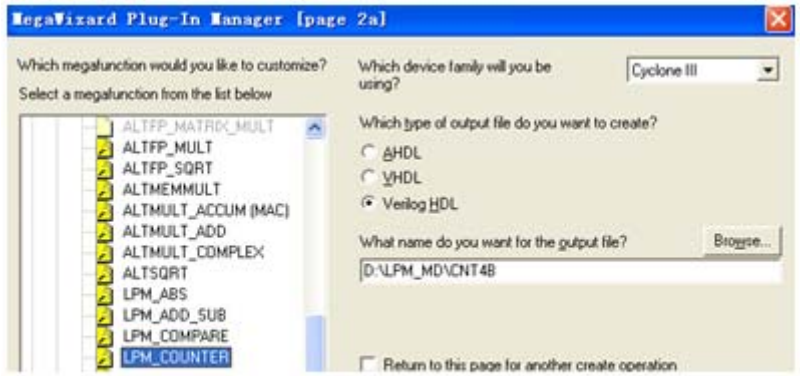

图 7-2　LPM 宏功能块设定

7.1 计数器LPM宏模块调用

7.1.1 计数器LPM模块文本代码的调用

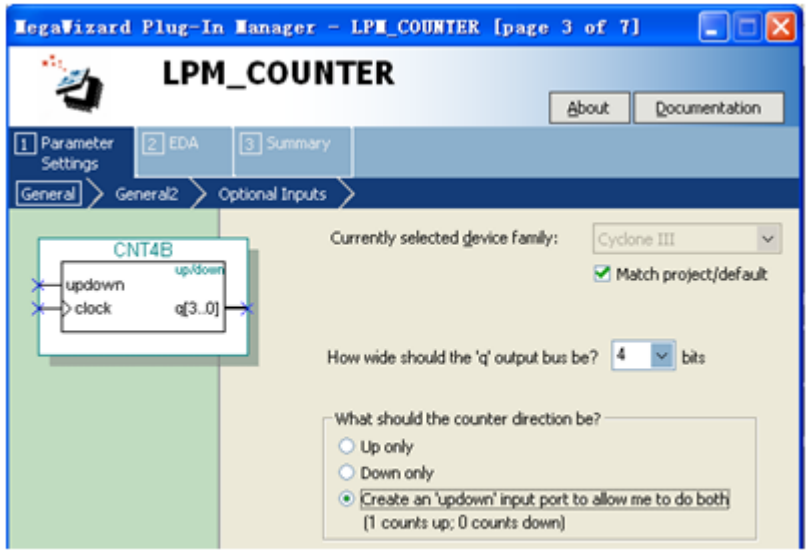

图 7-3 设 4 位可加减计数器

7.1 计数器LPM宏模块调用

7.1.1 计数器LPM模块文本代码的调用

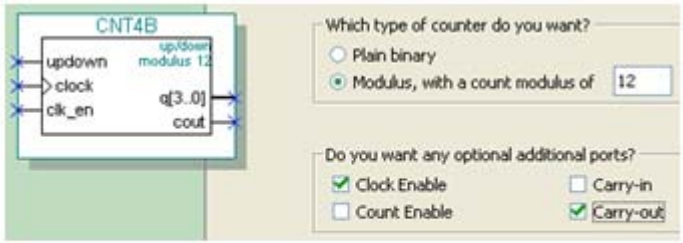

图 7-4 设定计数器，含时钟使能和进位输出

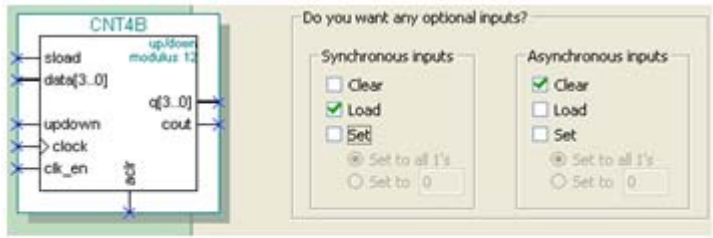

图 7-5 加入 4 位并行数据预置功能

7.1.2 LPM计数器代码与参数传递语句

【例 7-1】

```
module CNT4B (aclr, clk_en, clock, data, sload, updown, cout, q);
      input aclr, clk_en;        // 异步清 0，  1 清 0；时钟使能，1 使能，0 禁止
      input clock, sload;        // 时钟输入；同步预置数加载控制，1 加载，0 计数
      input [3:0] data; input   updown;  // 4 位预置数和加减控制，1 加，0 减
      output  cout;   output [3:0]  q;      // 进位输出和 // 4 位计数输出
      wire  sub_wire0;    wire [3:0] sub_wire1; // 定义内部连线
      wire  cout = sub_wire0;       wire [3:0] q = sub_wire1[3:0];
lpm_counter  lpm_counter_component(   //注意例化语句中未用端口必须接上指定电平
      .sload(sload), .clk_en(clk_en), .aclr(aclr),
      .data(data), .clock(clock), .updown(updown),
      .cout(sub_wire0), .q(sub_wire1), .aload(1'b0),
      .aset(1'b0), .cin(1'b1), .cnt_en(1'b1),
      .eq(),  .sclr(1'b0), .sset(1'b0));
  defparam
   lpm_counter_component.lpm_direction = "UNUSED",//单方向计数参数未用
   lpm_counter_component.lpm_modulus = 12,          //模 12 计数器
   lpm_counter_component.lpm_port_updown = "PORT_USED", //使用加减计数
   lpm_counter_component.lpm_type = "LPM_COUNTER",       //计数器类型
   lpm_counter_component.lpm_width = 4;                  //计数位宽
endmodule
```

7.1 计数器LPM宏模块调用

7.1.2 LPM计数器代码与参数传递语句

【例 7-2】

```
module REG24B (input [23:0] d, input  clk,  output [23:0] q);
   lpm_ff  U1(.q (q[11:0]), .data (d[11:0]), .clock (clk));
        defparam U1.lpm_width = 12;
    lpm_ff  U2(.q(q[23:12]), .data(d[23:12]), .clock(clk));
       defparam U2.lpm_width = 12;
endmodule
```

【例 7-3】

```
module CNT4BIT (RST,ENA,CLK,DIN,SLD,UD,COUT,DOUT);
  input RST, ENA, CLK, SLD,UD ;   input [3:0]  DIN;
  output  COUT; output [3:0]  DOUT ;
  CNT4B  U1(.sload (SLD), .clk_en (ENA), .aclr (RST), .cout (COUT),
           .clock (CLK), .data (DIN),  .updown (UD), .q (DOUT) );
endmodule
```

7.1 计数器LPM宏模块调用

7.1.3 创建工程与仿真测试

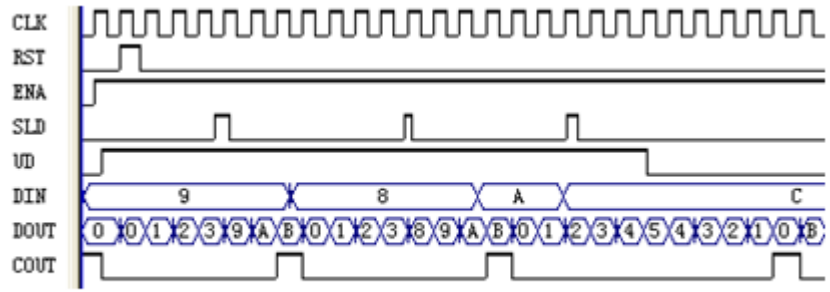

图 7-6 CNT4BIT.v 的仿真波形

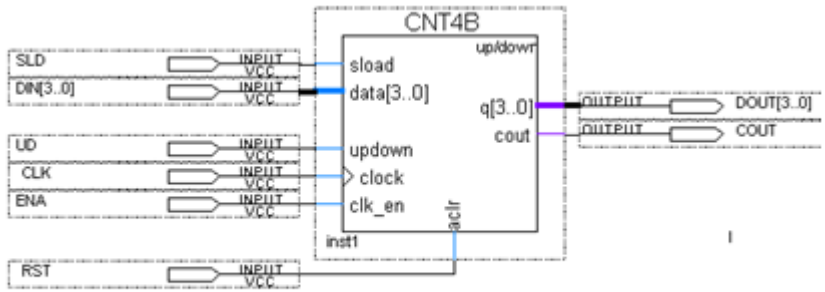

图 7-7 原理图输入设计

7.2 利用属性控制乘法器的构建

```
/* synthesis multstyle = "logic" */

    /* synthesis multstyle = "dsp" */
```

【例 7-4】

```
module MULT8 (A1,B1,A2,B2,R1,R2)  ;
    output signed[15:0]  R1,  R2  ;  // 定义有符号数据类型输出
    input signed[7:0] A1,B1,A2,B2; // 定义有符号数据类型输入
    wire  [15:0]  R2  /* synthesis multstyle = "logic" */;
    wire  [15:0]  R1 /* synthesis multstyle = "dsp" */;
    assign R1 = A1 * B1  ;    assign R2 = A2 * B2  ;
 endmodule
```

```
    module andd(A1,B1,A2,B2,R1,R2)  /* synthesis multstyle = "dsp" */;
```

7.2 利用属性控制乘法器的构建

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	64 / 328 (20 %)
Total virtual pins	0
Total memory bits	0 / 2,396,160 (0 %)
Embedded Multiplier 9-bit elements	2 / 312 (< 1 %)
Total PLLs	0 / 4 (0 %)

图 7-8　编译报告

图 7-9　设置乘法器用 DSP 模块构建

7.3 LPM_RAM宏模块的设置与使用

7.3.1 初始化文件及其生成

1. .mif格式文件

DATA7X8.mif

Addr	+0	+1	+2	+3	+4	+5	+6	+7
00	80	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

7.3 LPM_RAM宏模块的设置与使用

7.3.1 初始化文件及其生成

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_RAM宏模块的设置与使用

7.3.1 初始化文件及其生成

1. .mif格式文件

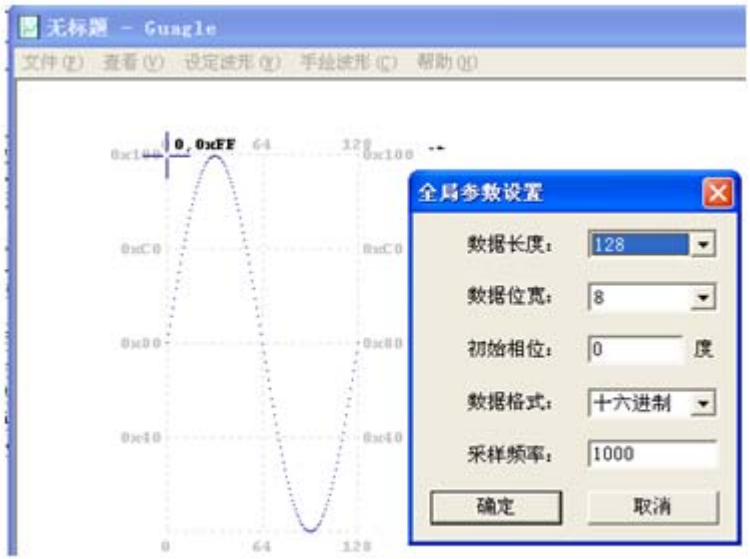

图 7-11 利用 mif 生成器生成 mif 正弦波文件

图 7-12 打开 mif 文件

7.3 LPM_RAM宏模块的设置与使用

7.3.1 初始化文件及其生成

2．.hex格式文件

3．.dat格式文件

00　E5　6D … 34

7.3 LPM_RAM宏模块的设置与使用

7.3.2 以原理图方式对LPM_RAM进行设置和调用

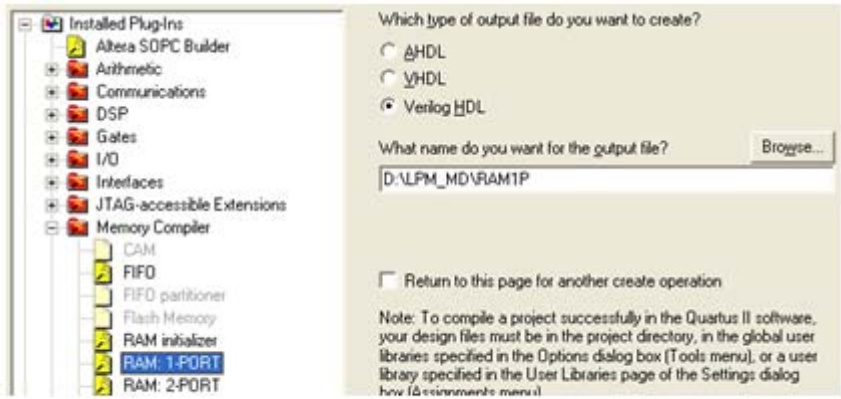

图 7-13 调用单口 LPM RAM

7.3 LPM_RAM宏模块的设置与使用

7.3.2 以原理图方式对LPM_RAM进行设置和调用

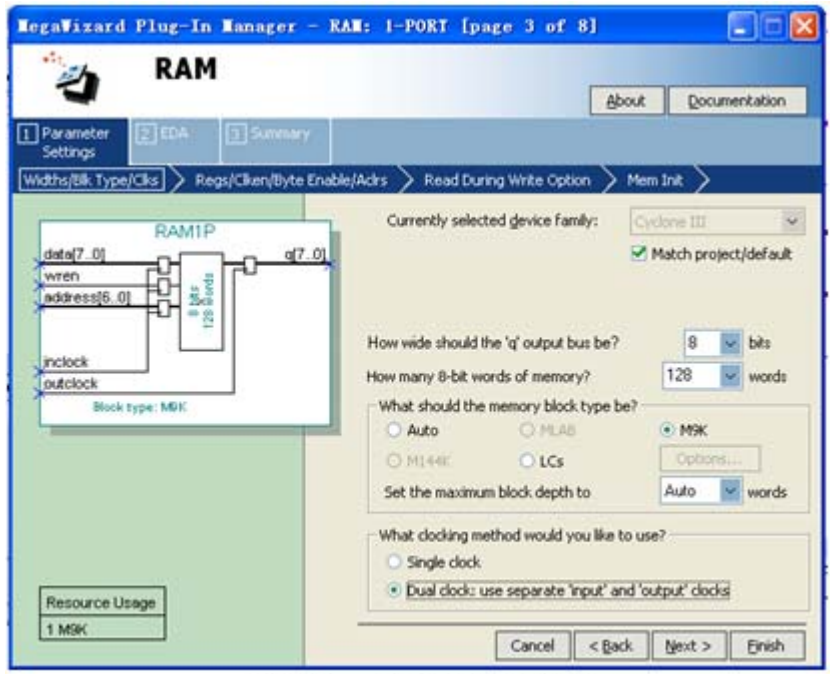

图 7-14 设定 RAM 参数

7.3 LPM_RAM宏模块的设置与使用

7.3.2 以原理图方式对LPM_RAM进行设置和调用

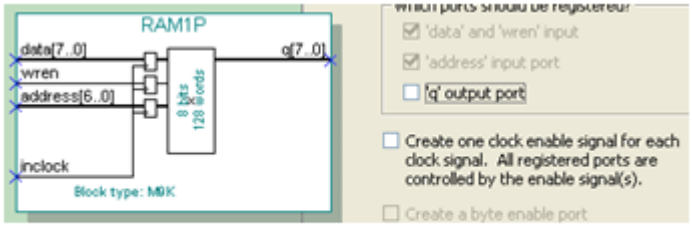

图 7-15 设定 RAM 仅输入时钟控制

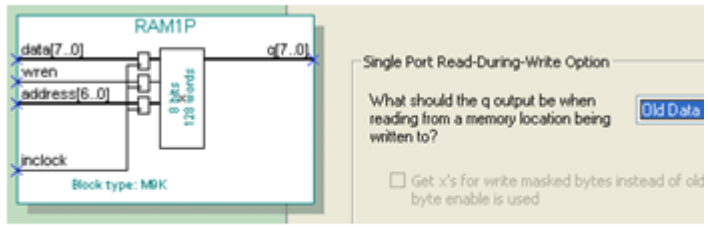

图 7-16 设定在写入同时读出原数据：Old Data

7.3 LPM_RAM宏模块的设置与使用

7.3.2 以原理图方式对LPM_RAM进行设置和调用

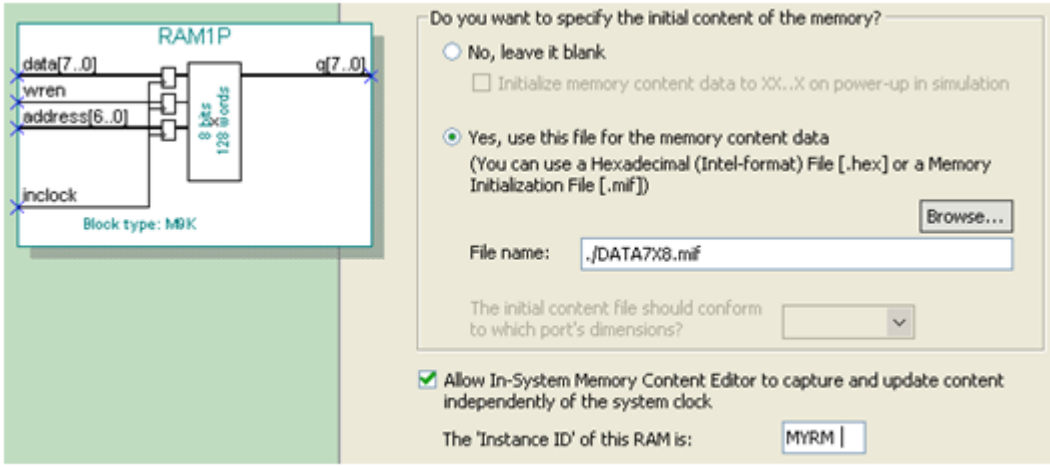

图 7-17 设定初始化文件和允许在系统编辑

7.3 LPM_RAM宏模块的设置与使用

7.3.2 以原理图方式对LPM_RAM进行设置和调用

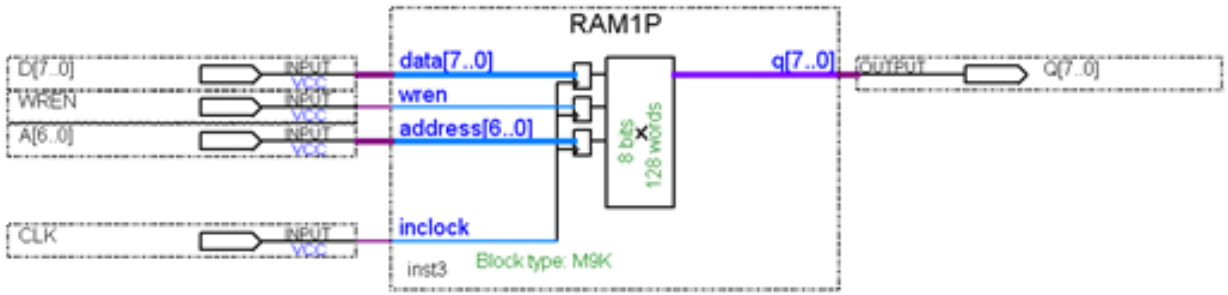

图 7-18 在原理图上连接好的 RAM 模块

7.3 LPM_RAM宏模块的设置与使用

7.3.3 测试LPM_RAM

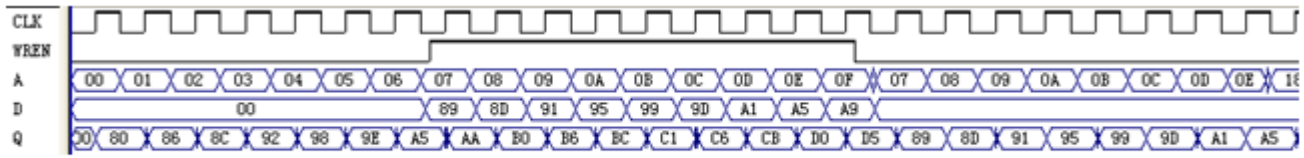

图 7-19 的 RAM 的仿真波形

7.3 LPM_RAM宏模块的设置与使用

7.3.4 存储器的Verilog代码描述

【例 7-6】

```
module RAM78 ( output wire[7:0] Q,    //定义 RAM 的 8 位数据输出端口
               input  wire[7:0] D,    //定义 RAM 的 8 位数据输入端口
               input  wire[6:0] A,    //定义 RAM 的 7 位地址输入端口
               input  wire CLK,WREN ) ;  //定义时钟和写允许控制
   reg[7:0]  mem[127:0]  /* synthesis ram_init_file="DATA7X8.mif" */ ;
   always @(posedge CLK )
     if (WREN) mem[A] <= D;  //在 CLK 上升沿将数据口 D 的数据锁入地址对应单元中
   assign Q = mem[A];        //同时，地址对应单元的数据被输出至端口
endmodule
```

7.3 LPM_RAM宏模块的设置与使用

7.3.4 存储器的Verilog代码描述

1. 存储器端口描述

2. 存储器的Verilog一般描述

```
parameter width=8, msize=1024;
reg[width-1:0] MEM87[msize-1:0];
```

```
reg[7:0] mem87[128:0];
mem87[16]=8'b11001001; // mem87 存储器的第 16 单元被赋值为二进制数 11001001
mem87[122]=76;         // mem87 存储器的第 122 单元被赋值为十进制数 76。
```

```
reg [15:0] A;       // 定义了一个 16 位的寄存器
reg MEM[15:0];      // 定义了一个字长为 1，即 1 位的，容量深度为 16 的存储器
```

```
A[5] = 1'b0;          // 允许对寄存器 A 的第 5 位赋值 0
MEM[7] = 1'b1;        // 允许对存储器 MEM 的第 7 个单元赋值 1
A = 16'hABCD ;        // 允许对寄存器 A 整体赋值
MEM = 16'hABCD;       // 错误！不允许对存储器多个或者所有单元同时赋值
```

7.3 LPM_RAM宏模块的设置与使用

7.3.4 存储器的Verilog代码描述

3. 存储器中初始化文件的调用与配置

```
/* synthesis ram_init_file="DATA7X8.mif" */ ;
```

```
(* ram_init_file = "DATA7X8.mif" *) reg[7:0] mem[127:0]
```

【例 7-7】

```
module RAM78 (output[7:0]Q, input[7:0]D,input[6:0]A, input CLK,WREN);
  reg[7:0]  mem[0:127] ;
  always @(posedge CLK )  if (WREN) mem[A] <= D;
   assign Q = mem[A];
   initial $readmemh("RAM78_DAT.dat", mem );
endmodule
```

7.3 LPM_RAM宏模块的设置与使用

7.3.4 存储器的Verilog代码描述

4. 语句语法说明

```
initial
  begin  语句1;  语句2; ...  end
```

7.3 LPM_RAM宏模块的设置与使用

7.3.5 存储器设计的结构控制

【例 7-8】

```
module RAM78(output reg[7:0] Q, input[7:0] D,input[6:0] A, input CLK,WREN);
    reg[7:0]  mem[127:0]  /*synthesis ram_init_file="DATA7X8.mif" */;
    always  @(posedge CLK )  if (WREN) mem[A] <= D;
    always @(posedge CLK )  Q = mem[A];
  endmodule
```

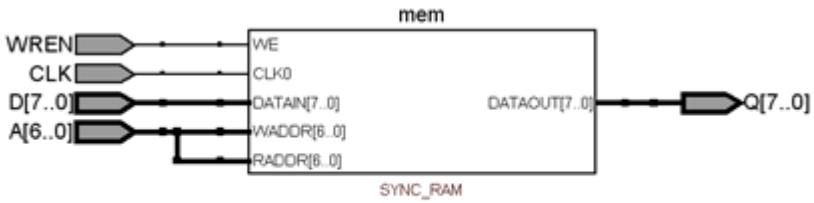

图 7-20 例 7-6 的 RTL 电路模块图

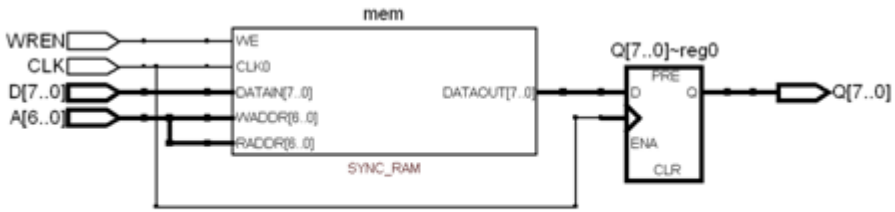

图 7-21 例 7-8 的 RTL 电路模块图

7.3 LPM_RAM宏模块的设置与使用

7.3.5 存储器设计的结构控制

Top-level Entity Name	RAM78
Family	Cyclone III
Device	EP3C55F484C8
Timing Models	Final
Met timing requirements	N/A
Total logic elements	1,497 / 55,856 (3 %)
Total combinational functions	1,340 / 55,856 (2 %)
Dedicated logic registers	1,024 / 55,856 (2 %)
Total registers	1024
Total pins	25 / 328 (8 %)
Total virtual pins	0
Total memory bits	0 / 2,396,160 (0 %)
Embedded Multiplier 9-bit elements	0 / 312 (0 %)
Total PLLs	0 / 4 (0 %)

图 7-22 例 7-6 的编译报告

Top-level Entity Name	RAM78
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	25 / 328 (8 %)
Total virtual pins	0
Total memory bits	1,024 / 2,396,160
Embedded Multiplier 9-bit elements	0 / 312 (0 %)
Total PLLs	0 / 4 (0 %)

图 7-23 例 7-8 的编译报告

7.3 LPM_RAM宏模块的设置与使用

7.3.5 存储器设计的结构控制

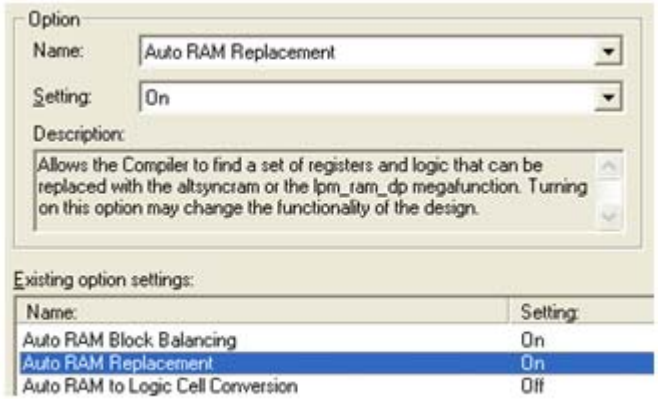

图 7-24 选择 RAM 单元自动替代控制

7.4 LPM_ROM的定制和使用示例

7.4.1 LPM_ROM的调用

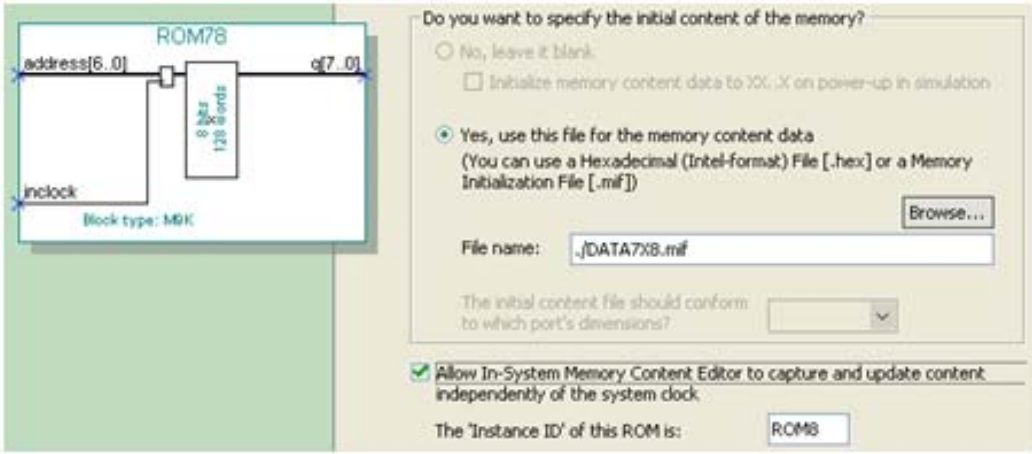

图 7-25 加入初始化配置文件并允许在系统访问 ROM 内容

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

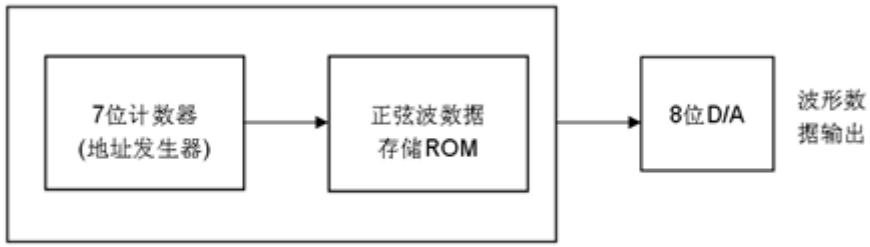

图 7-26 正弦信号发生器结构框图

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

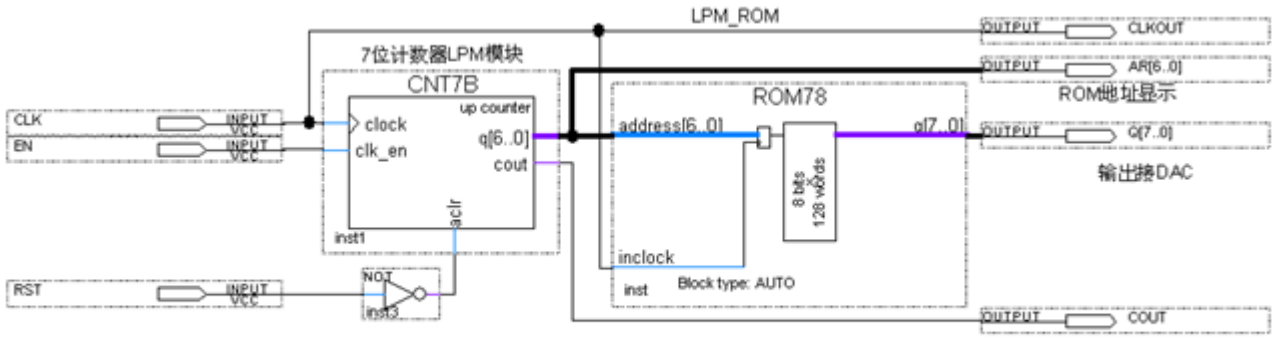

图 7-27 正弦信号发生器电路原理图

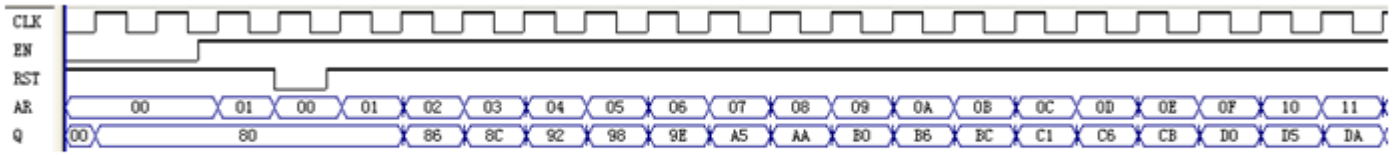

图 7-28 图 7-27 电路仿真波形

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

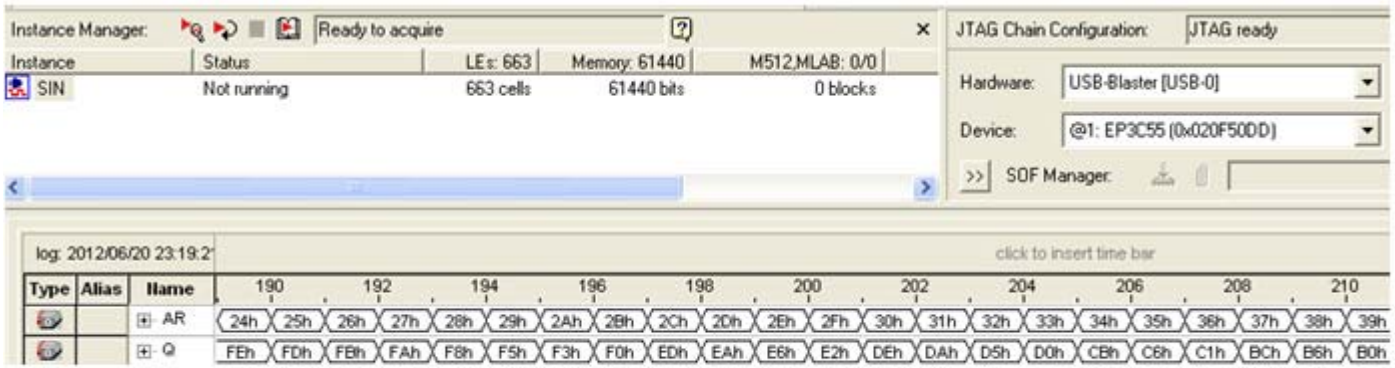

图 7-29 正弦信号发生器数据输出的 SignalTapII 实时测试图

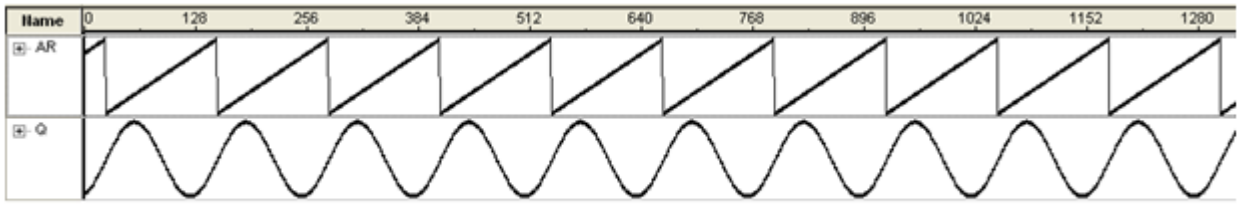

图 7-30 正弦信号发生器的 SignalTapII 的波形显示图

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。

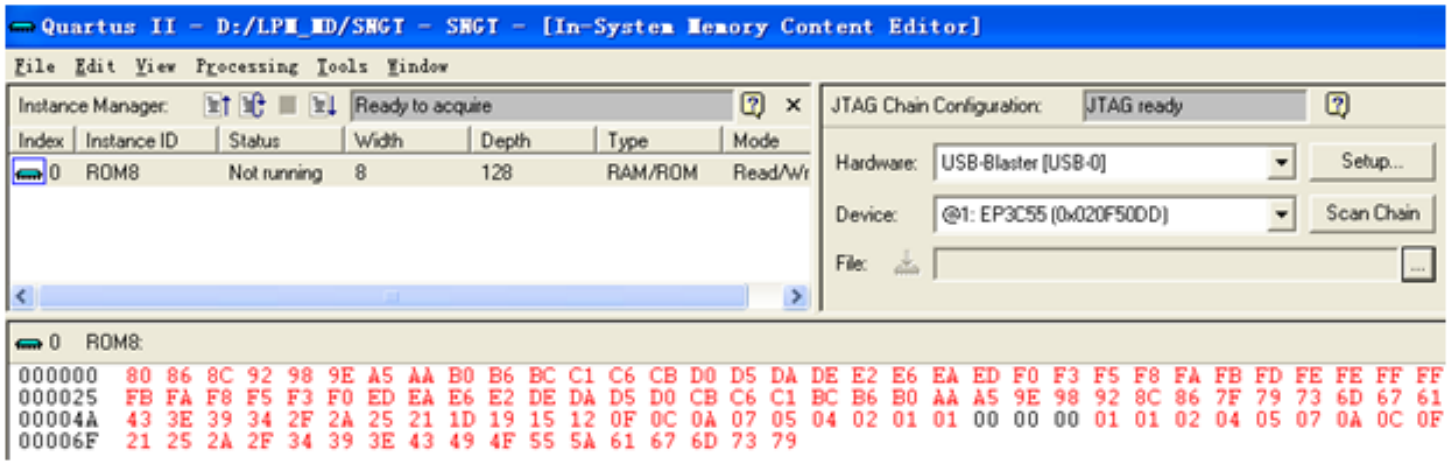

图 7-31 In-System Memory Content Editor 编辑窗

（2）读取ROM中的数据。

7.5 在系统存储器数据读写编辑器应用

（3）写数据。

0 ROM8:

```
000000  11 11 11 11 11 11 11 AA B0 B6 BC C1 C6 CB D0 D5 DA DE E2 E6 EA ED F0 F3 F5 F8 FA FB FD FE FE FF FF FF FE FE FD
000025  FB FA F8 F5 F3 F0 ED EA E6 E2 DE DA D5 D0 CB C6 C1 BC B6 B0 AA A5 9E 98 92 8C 86 7F 79 73 6D 67 61 5A 55 4F 49
00004A  43 3E 39 34 2F 2A 25 21 1D 19 15 12 0F 0C 0A 07 05 04 02 01 01 00 00 00 01 01 02 04 05 07 0A 0C 0F 12 15 19 1D
00006F  21 25 2A 2F 34 39 3E 43 49 4F 55 5A 61 67 6D 73 79
```

图 7-32 从 FPGA 中的 ROM 读取波形数据并编辑数据

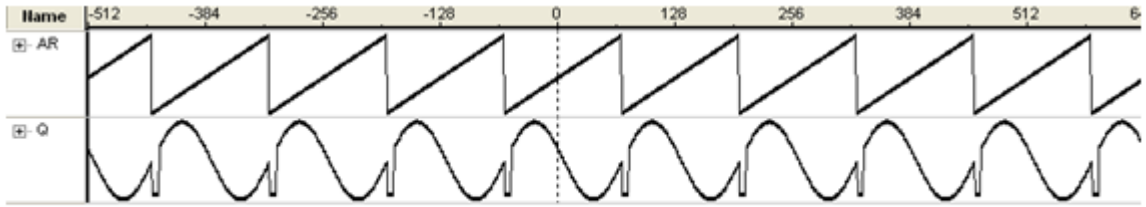

图 7-33 下载编辑数据后的 SignalTap II 采样波形

（4）输入输出数据文件。

7.6 LPM嵌入式锁相环调用

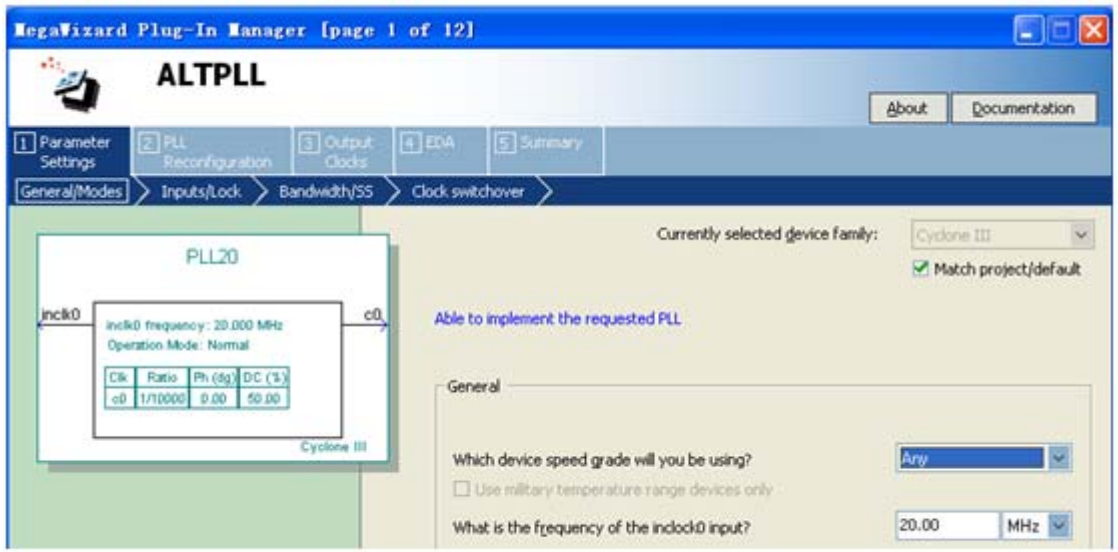

图 7-34 选择输入参考时钟 inclk0 为 20MHz

7.6 LPM嵌入式锁相环调用

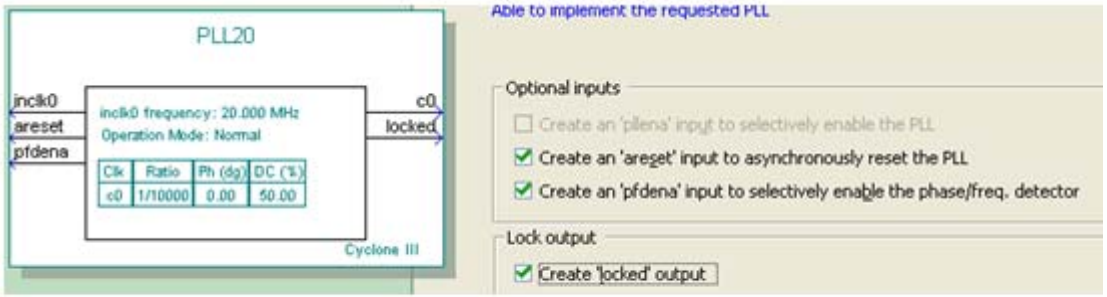

图 7-35 选择控制信号

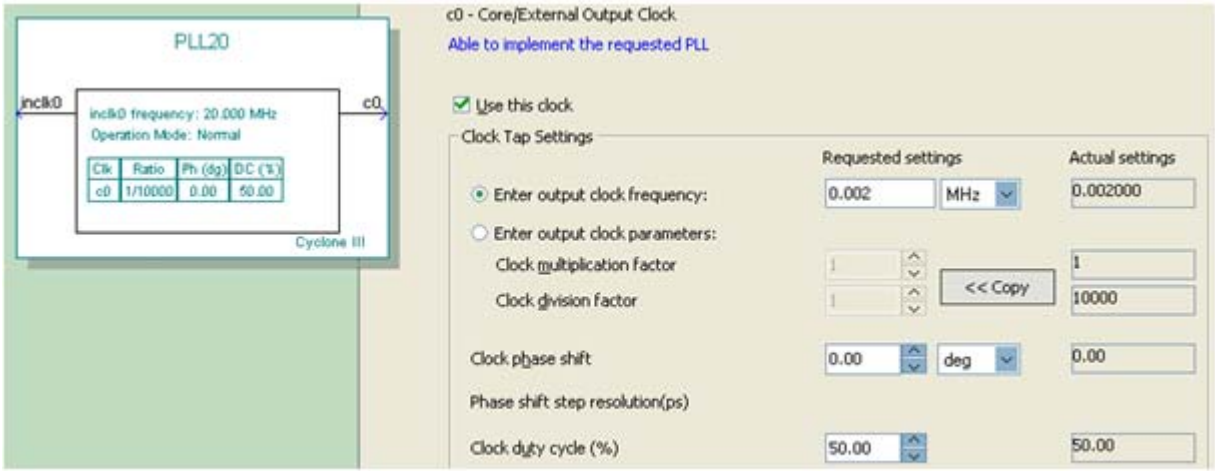

图 7-36 选择 c0 的输出频率为 0.002MHz

7.6 LPM嵌入式锁相环调用

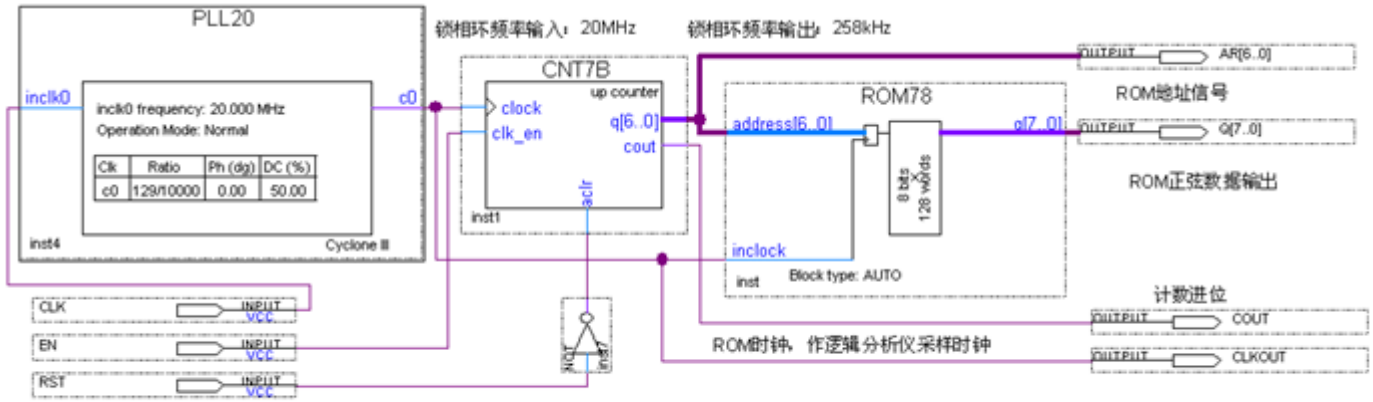

图 7-37 采用嵌入式锁相环作时钟的正弦信号发生器电路图

7.7 In-System Sources and Probes Editor使用方法

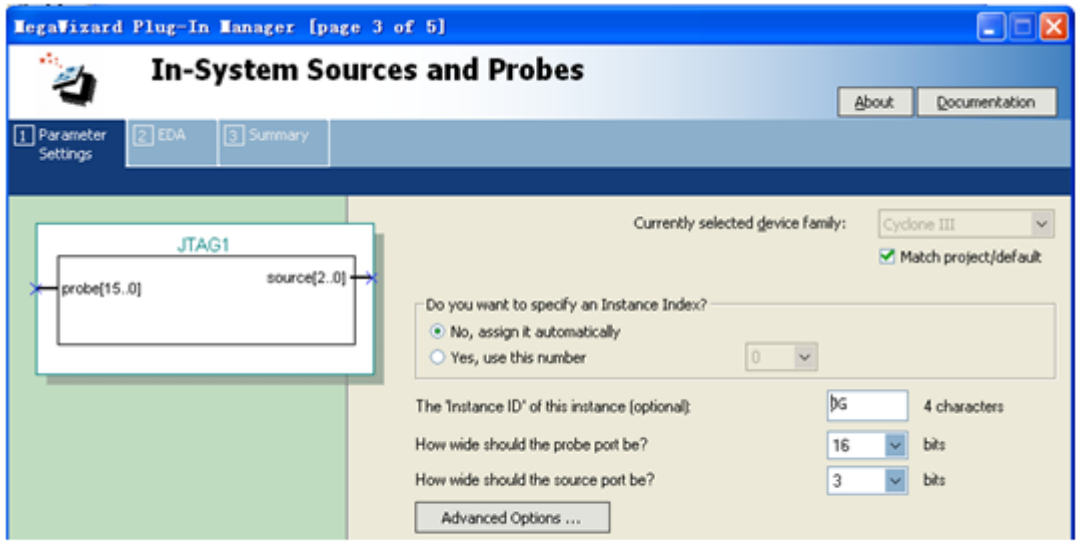

图 7-38 为 In-System Sources and Probes 模块设置参数

7.7 In-System Sources and Probes Editor使用方法

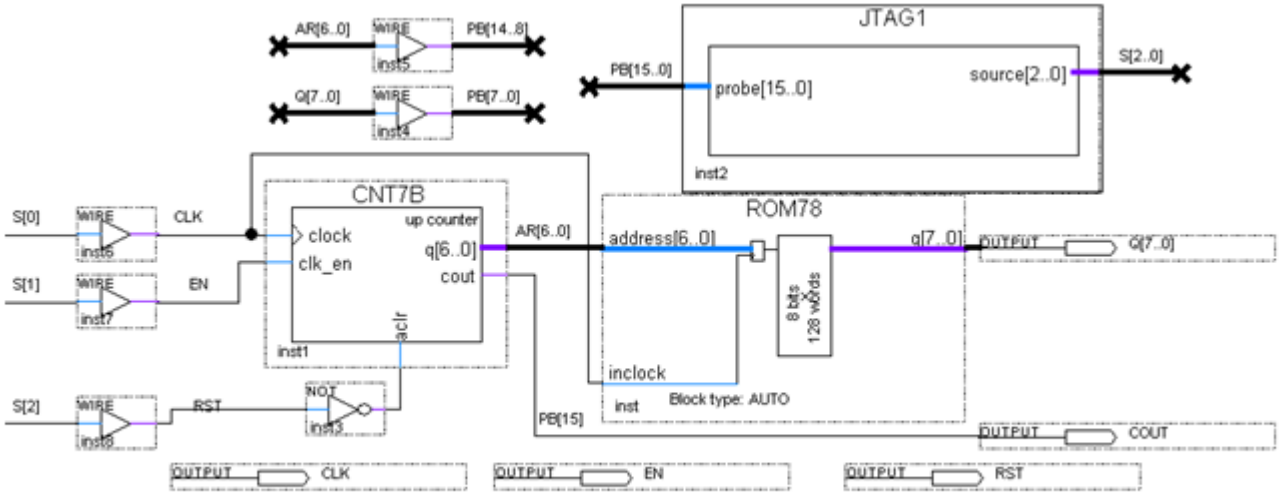

图 7-39 在双十进制计数器设计电路中加入 In-System Sources and Probes 测试模块

7.7 In-System Sources and Probes Editor使用方法

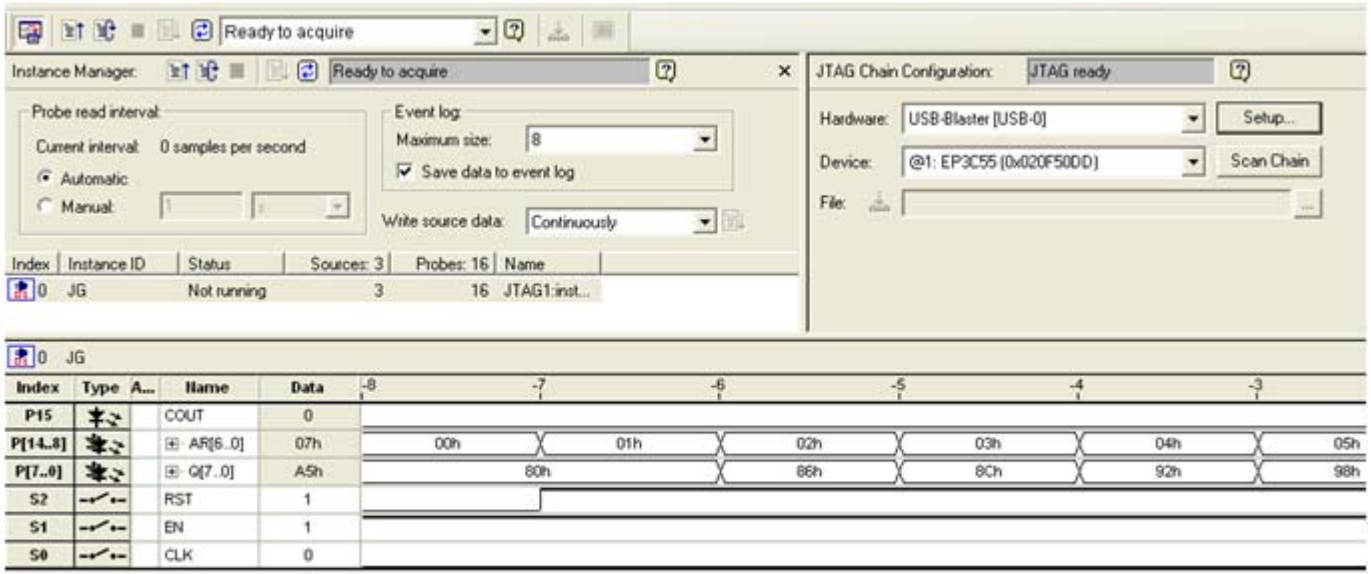

图 7-40 In-System Sources and Probes Editor 的测试情况

7.8 数控振荡器核使用方法

（1）定制NCO。

（2）进入Core文件生成选择窗。

图 7-41 开始进入 Core 文件生成选择窗口

7.8 数控振荡器核使用方法

（3）设置参数。

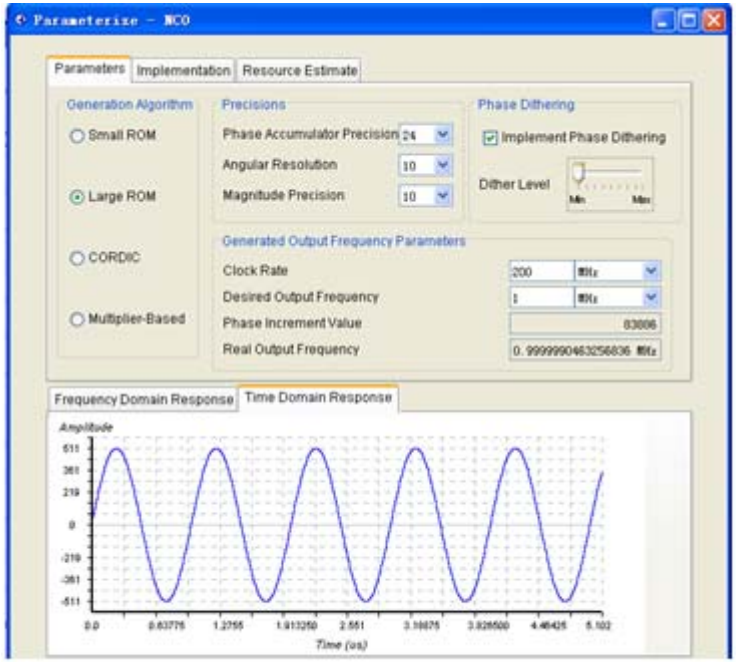

图 7-42 设置 NCO 参数

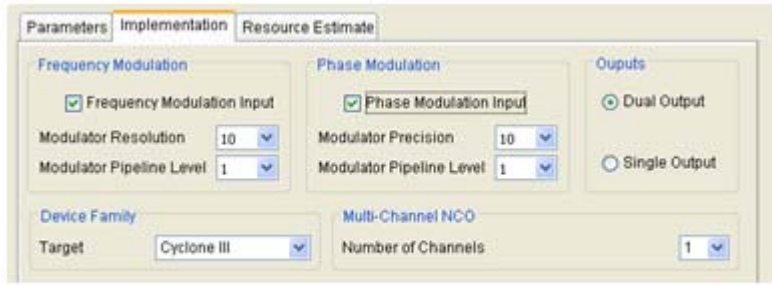

图 7-43 设置 NCO 参数

7.8 数控振荡器核使用方法

（4）生成仿真文件。

（5）加入IP授权文件。

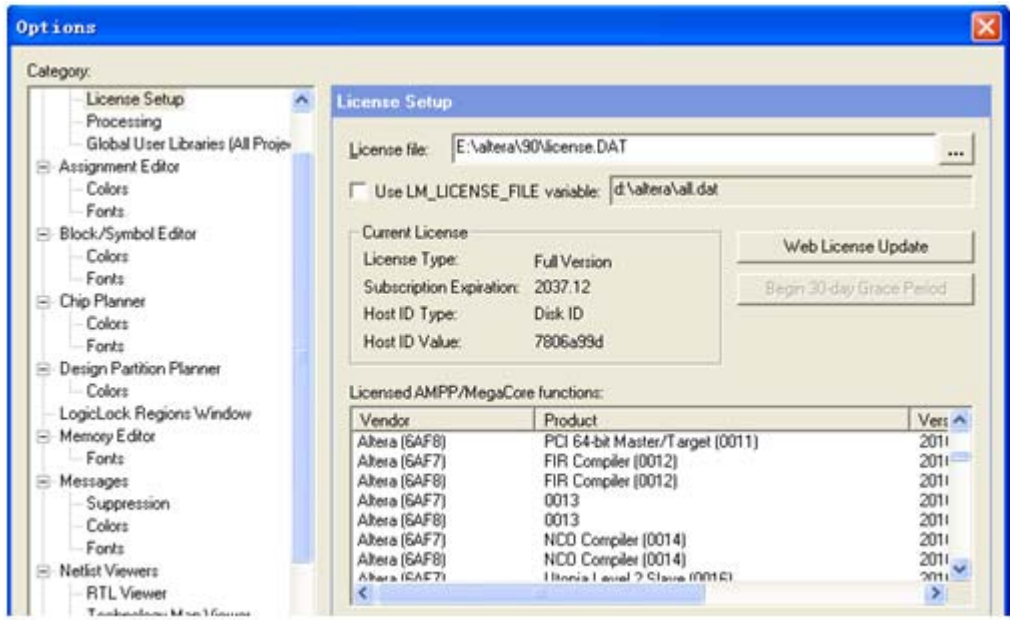

图 7-44 加入含有 NCO 等 IP 的授权文件

7.8 数控振荡器核使用方法

（6）选择目标器件，然后对生成的模块进行编译及功能检测。

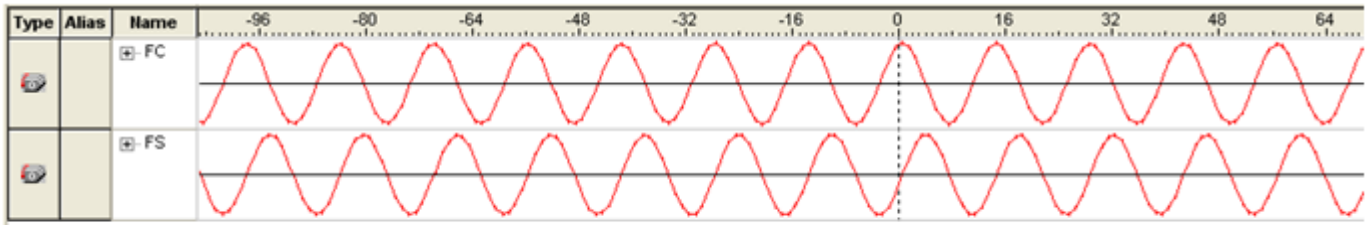

图 7-45 当前 NCO 的逻辑分析仪测试波形

7.9 FIR核使用方法

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

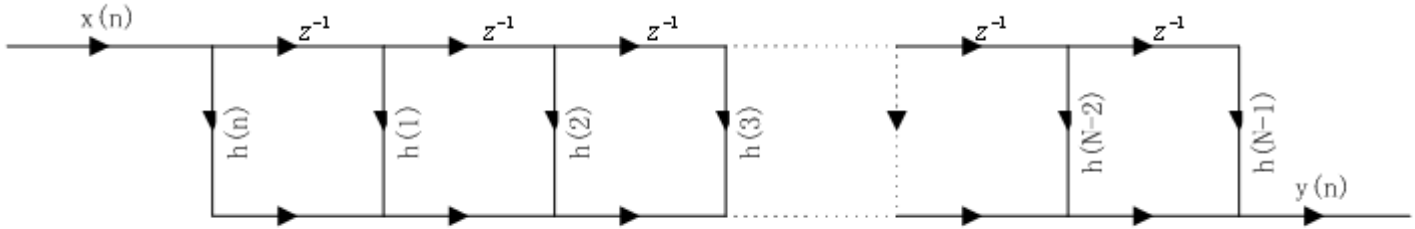

图 7-46 直接型 FIR 滤波器结构

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

7.9 FIR核使用方法

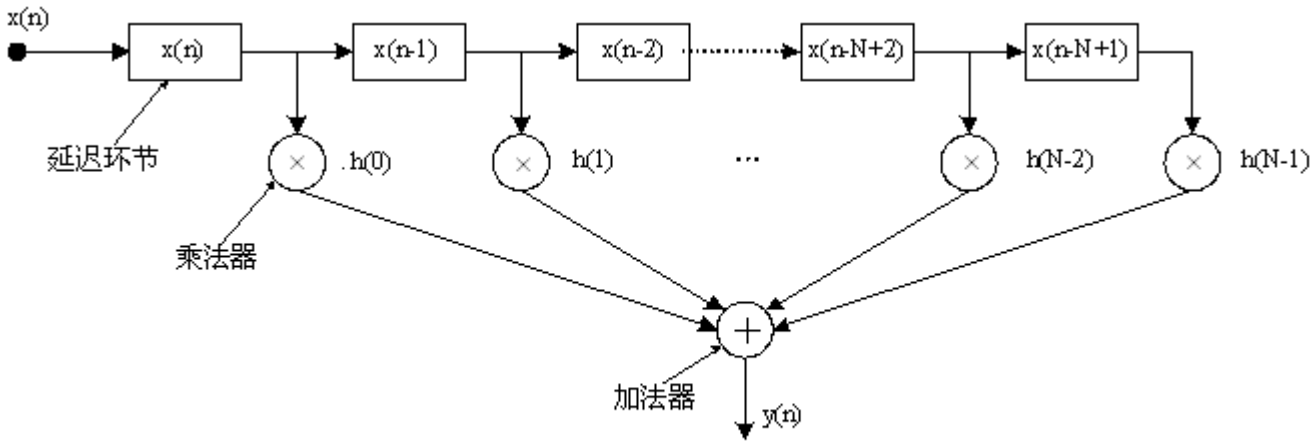

图 7-47 直接型 FIR 实现结构

图 7-48 FIR 滤波器设计示意

7.10 DDS实现原理与应用

7.10.1 DDS原理

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

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

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

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

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

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

7.10 DDS实现原理与应用

7.10.1 DDS原理

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

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

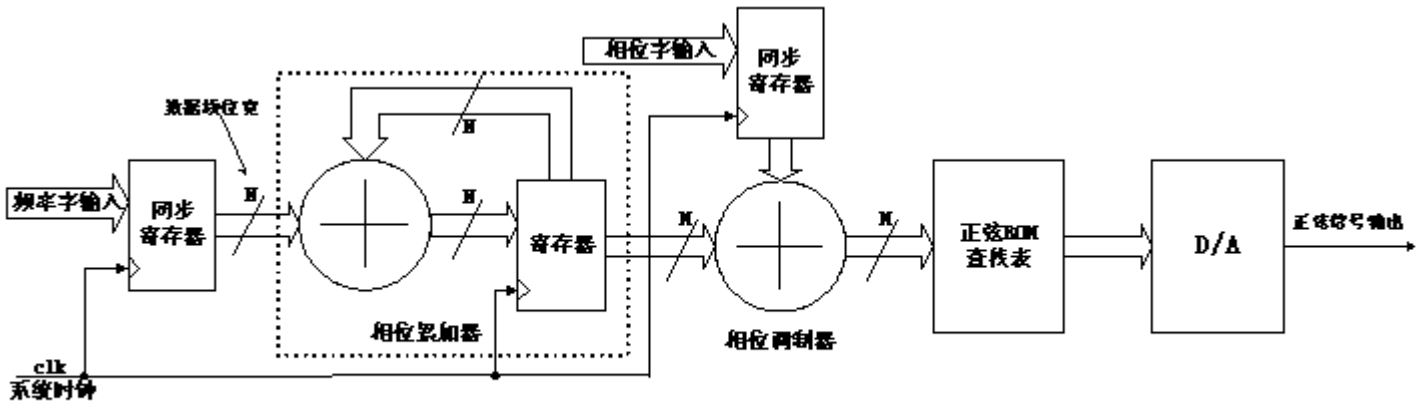

图 7-49 基本 DDS 结构

7.10 DDS实现原理与应用

7.10.2 DDS信号发生器设计示例

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

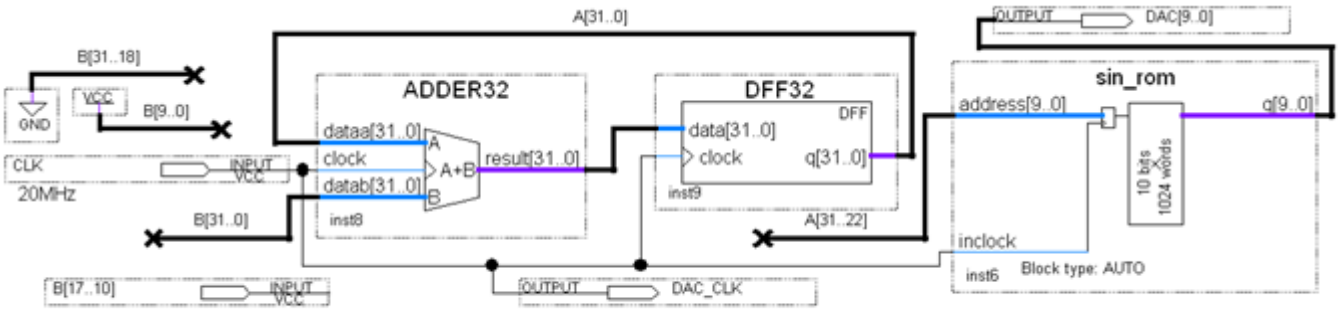

图 7-50　DDS 信号发生器电路顶层原理图

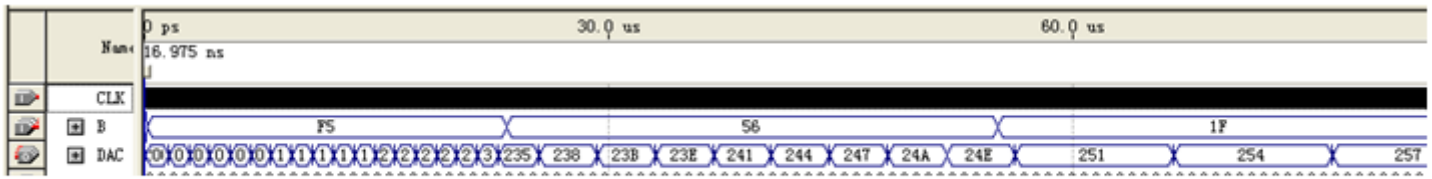

图 7-51 图 7-50 的仿真波形

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．简易逻辑分析仪设计

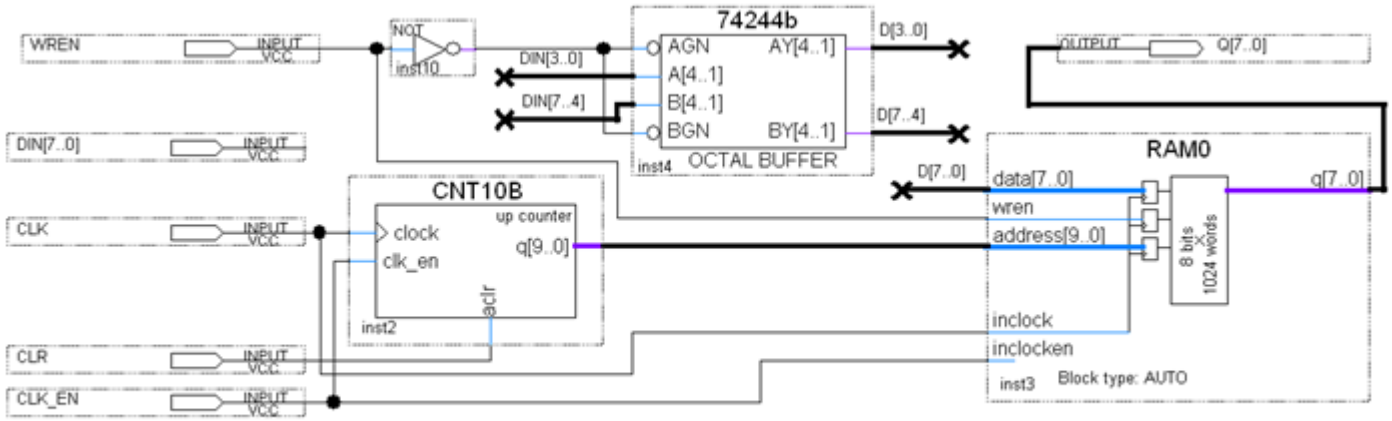

图 7-52 逻辑数据采样电路顶层设计

实验与设计

7-3. 简易逻辑分析仪设计

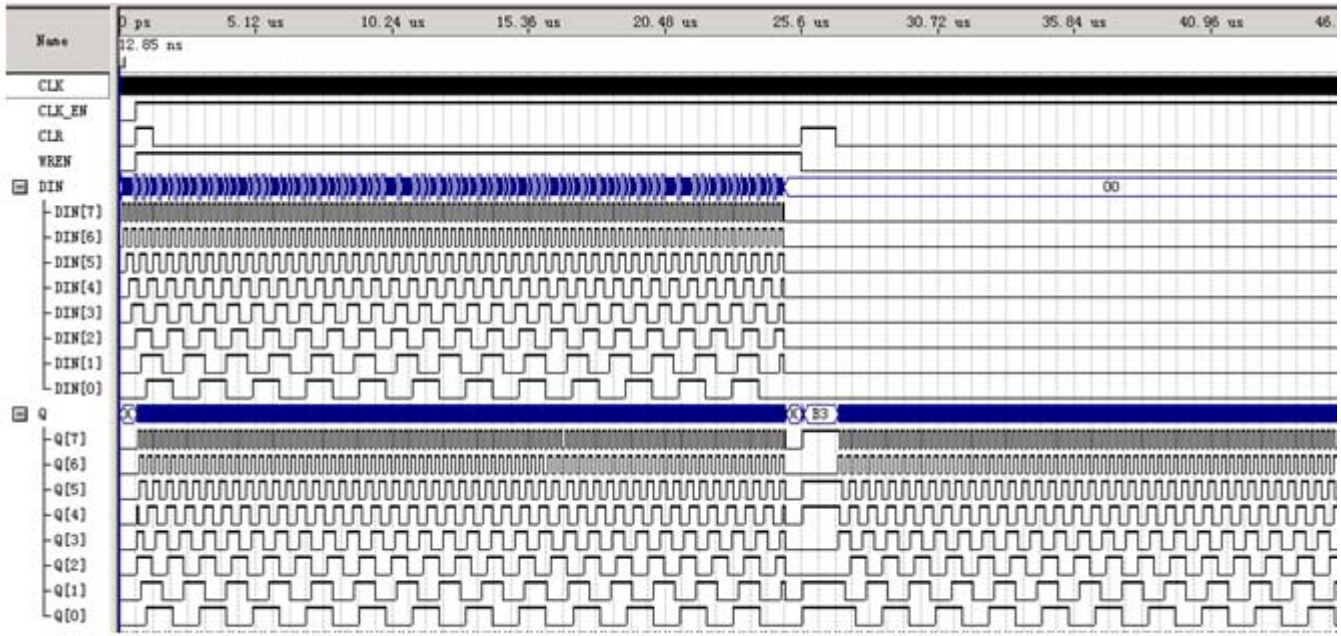

图 7-53 逻辑数据采样电路时序仿真波形

实验与设计

7-4 DDS正弦信号发生器设计

7-5 移相信号发生器设计

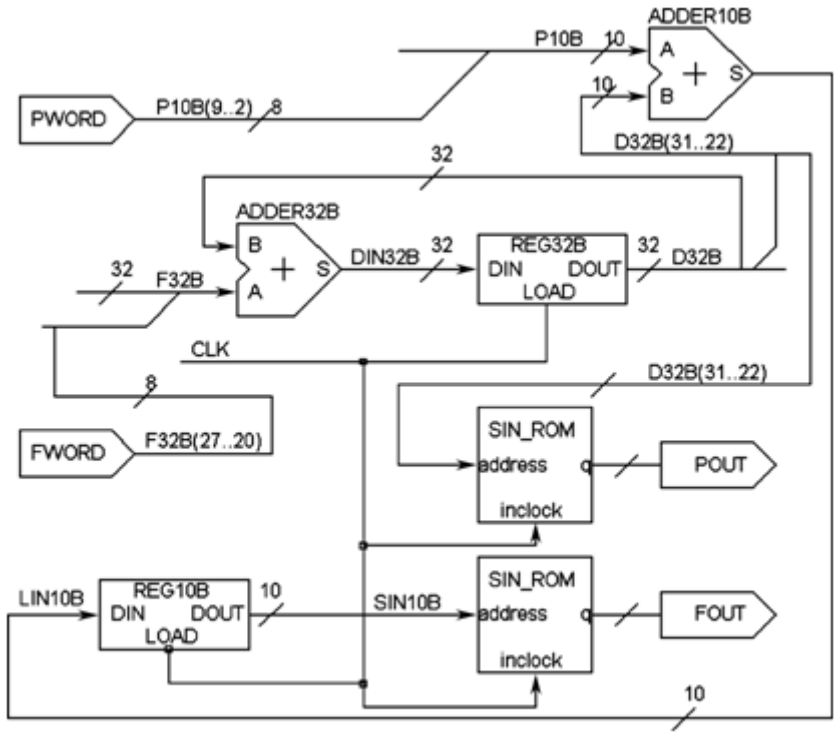

图 7-54 全数字移相信号发生器电路模型图

实验与设计

7-6 16位×16位高速硬件乘法器设计

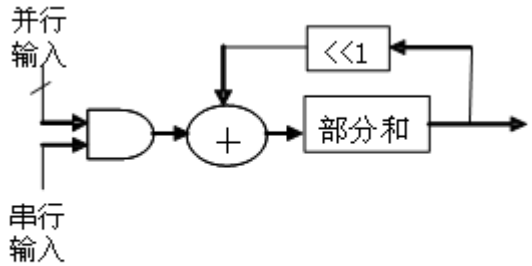

图7-55 最基本的硬件乘法器

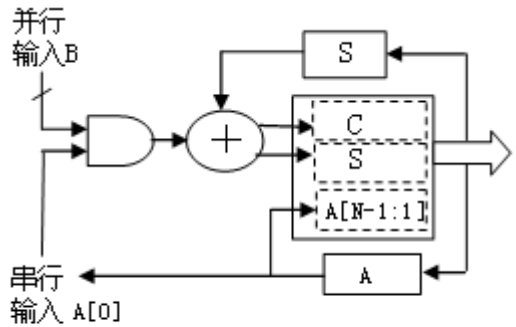

图7-56 改进后的硬件乘法器