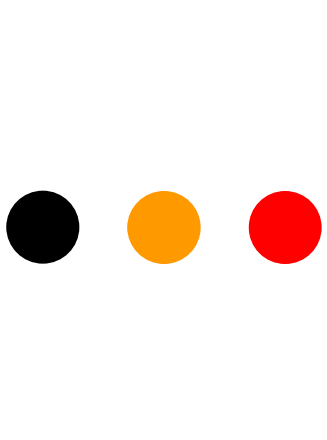

第6章

宏功能模块应用及相关语法

6.1 计数器LPM模块调用示例

6.1.1 计数器模块文本的调用

（1）打开**LPM**宏功能块调用管理器

图 6-1　定制新的宏功能块

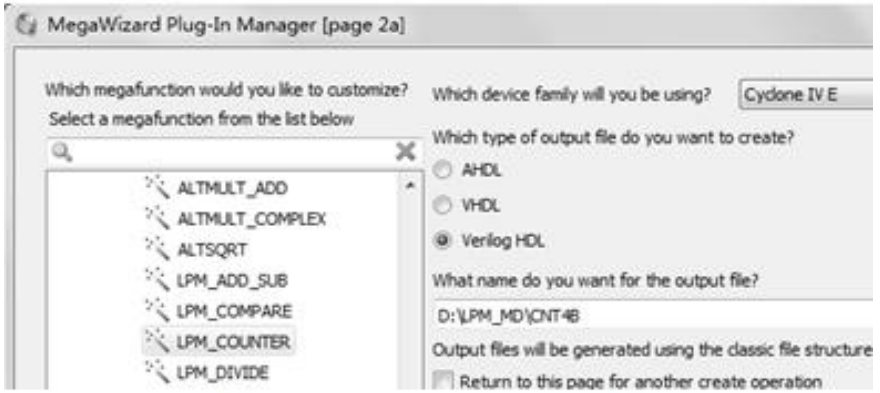

图 6-2　LPM 宏功能块设定

6.1 计数器LPM模块调用示例

6.1.1 计数器模块文本的调用

（2）单击**Next**按钮后打开如图**6-3**所示的对话框

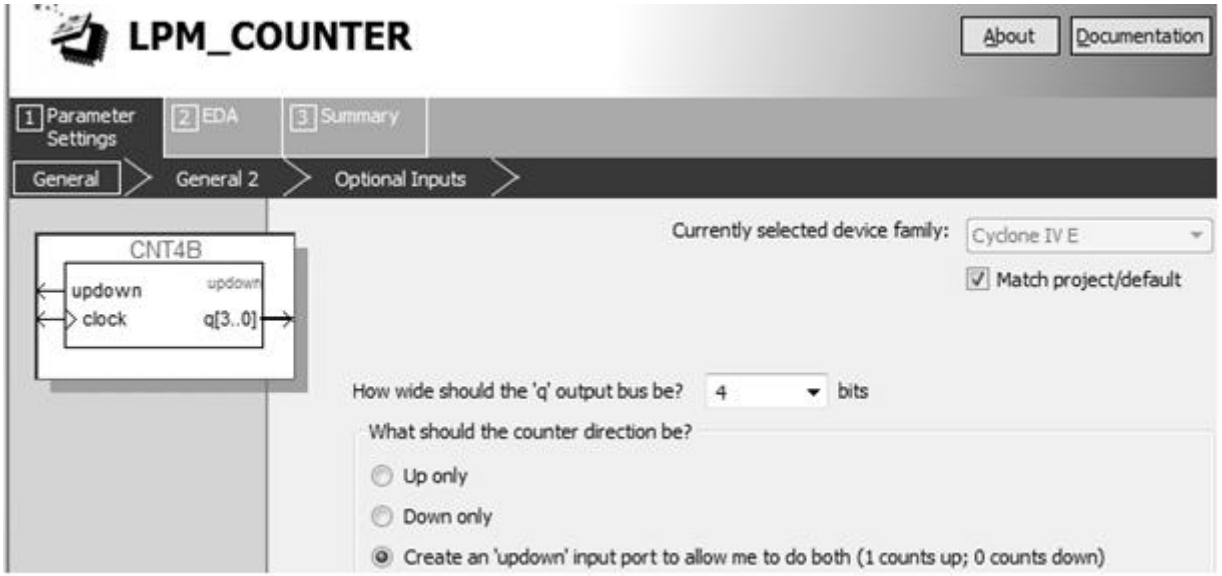

图 6-3　设 4 位可加减计数器

6.1 计数器LPM模块调用示例

6.1.1 计数器模块文本的调用

（3）再单击**Next**按钮，打开如图**6-4**所示的对话框

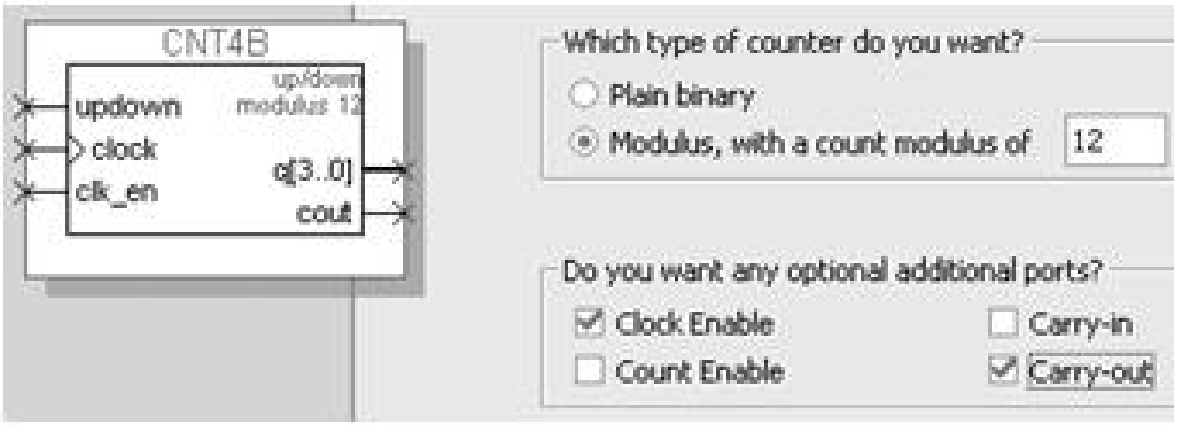

图 6-4　设定计数器，含时钟使能和进位输出

6.1 计数器LPM模块调用示例

6.1.1 计数器模块文本的调用

（4）再单击Next按钮，打开如图6-5所示的对话框

图 6-5　加入 4 位并行数据预置功能

6.1 计数器LPM模块调用示例

6.1.2 LPM计数器代码与参数传递语句应用

【例 6-1】

```
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
```

6.1 计数器LPM模块调用示例

6.1.2 LPM计数器代码与参数传递语句应用

```
defparam <宏模块元件例化名>.<宏模块参数名> = <参数值>
```

【例 6-2】

```
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 6-3】

```
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
```

6.1 计数器LPM模块调用示例

6.1.3 创建工程与仿真测试

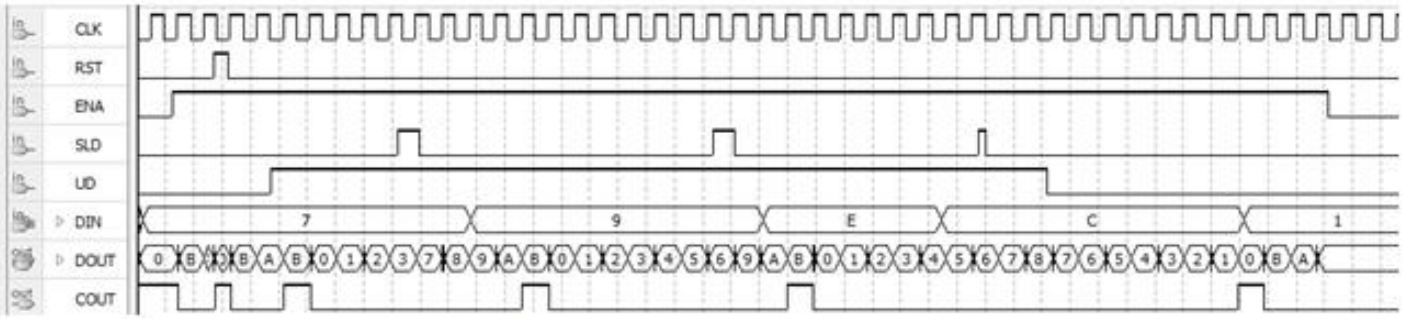

图 6-6　CNT4BIT.v 的仿真波形

6.2 利用属性控制乘法器的构建

【例 6-4】

```
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 = " logic " */;
   assign R1 = A1 * B1;  assign R2 = A2 * B2;
endmodule
```

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

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

6.2 利用属性控制乘法器的构建

Flow Status	Successful - Tue May 23 21:32:59 2017
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Full Version
Revision Name	MULT8
Top-level Entity Name	MULT8
Family	Cyclone IV E
Device	EP4CE55F23C8
Timing Models	Final
Total logic elements	190 / 55,856 (< 1 %)
Total combinational functions	190 / 55,856 (< 1 %)
Dedicated logic registers	0 / 55,856 (0 %)
Total registers	0
Total pins	64 / 325 (20 %)
Total virtual pins	0
Total memory bits	0 / 2,396,160 (0 %)
Embedded Multiplier 9-bit elements	0 / 308 (0 %)
Total PLLs	0 / 4 (0 %)

图 6-7　完全用逻辑宏单元构建乘法器的编译报告

6.2 利用属性控制乘法器的构建

Flow Status	Successful - Tue May 23 21:30:42 2017
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Full Version
Revision Name	MULT8
Top-level Entity Name	MULT8
Family	Cyclone IV E
Device	EP4CE55F23C8
Timing Models	Final
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 / 325 (20 %)
Total virtual pins	0
Total memory bits	0 / 2,396,160 (0 %)
Embedded Multiplier 9-bit elements	2 / 308 (< 1 %)
Total PLLs	0 / 4 (0 %)

图 6-8　调用了 DSP 模块的编译报告

6.3 LPM 随机存储器的设置和调用

6.3.1 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	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 6-9　mif 文件编辑窗

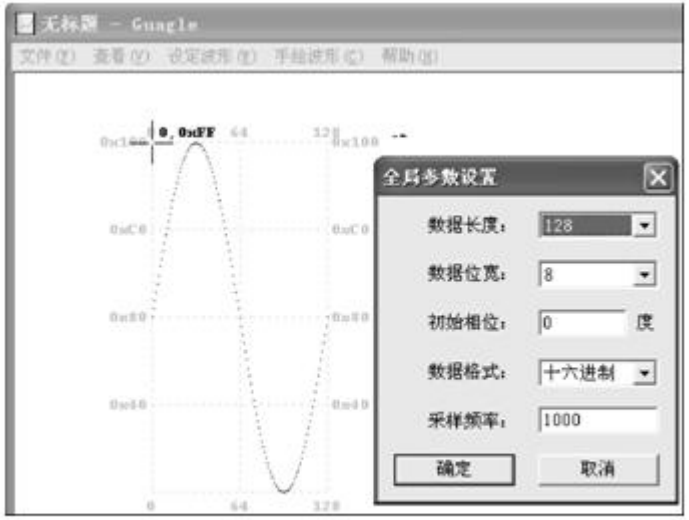

图 6-10　利用 mif 生成器生成 mif 正弦波文件

```
DATA7X8.mif - 记事本
文件(F)  编辑(E)  格式(O)  查
DEPTH = 128;
WIDTH = 8;
ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;
CONTENT  BEGIN
0000 : 0080;
0001 : 0086;
0002 : 008C;
0003 : 0092;
0004 : 0098;
0005 : 009E;
0006 : 00A5;
0007 : 00AA;
0008 : 00B0;
...
007E : 0073;
007F : 0079;
END ;
```

图 6-11　mif 文件

6.3 LPM 随机存储器的设置和调用

6.3.1 存储器初始化文件

1. mif格式文件

【例 6-5】

```
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 表示选择十六进制数据类型
DATA_RADIX = HEX;           ：存储数据类型，HEX 表示选择十六进制数据类型
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;
```

6.3 LPM 随机存储器的设置和调用

6.3.1 存储器初始化文件

2．hex格式文件

3．dat格式文件

```
00  E5  6D ... 34
```

6.3 LPM 随机存储器的设置和调用

6.3.2 LPM_RAM的设置和调用

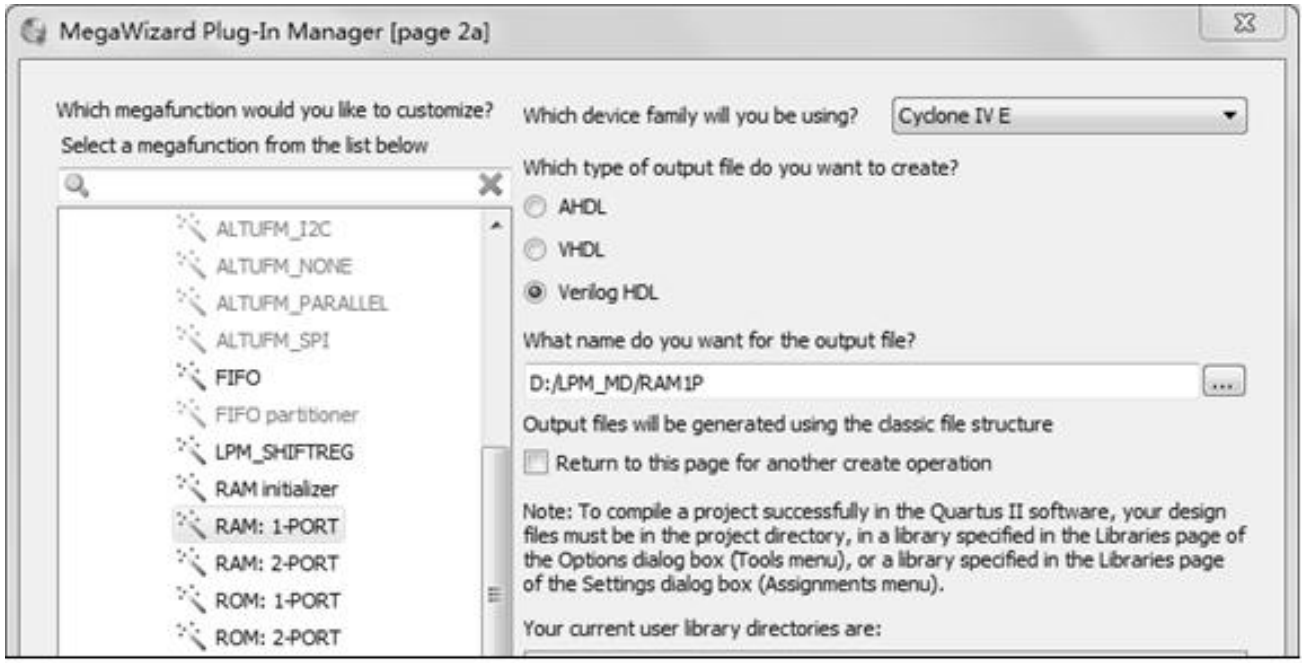

图 6-12　调用单口 LPM RAM

6.3 LPM 随机存储器的设置和调用

6.3.2 LPM_RAM的设置和调用

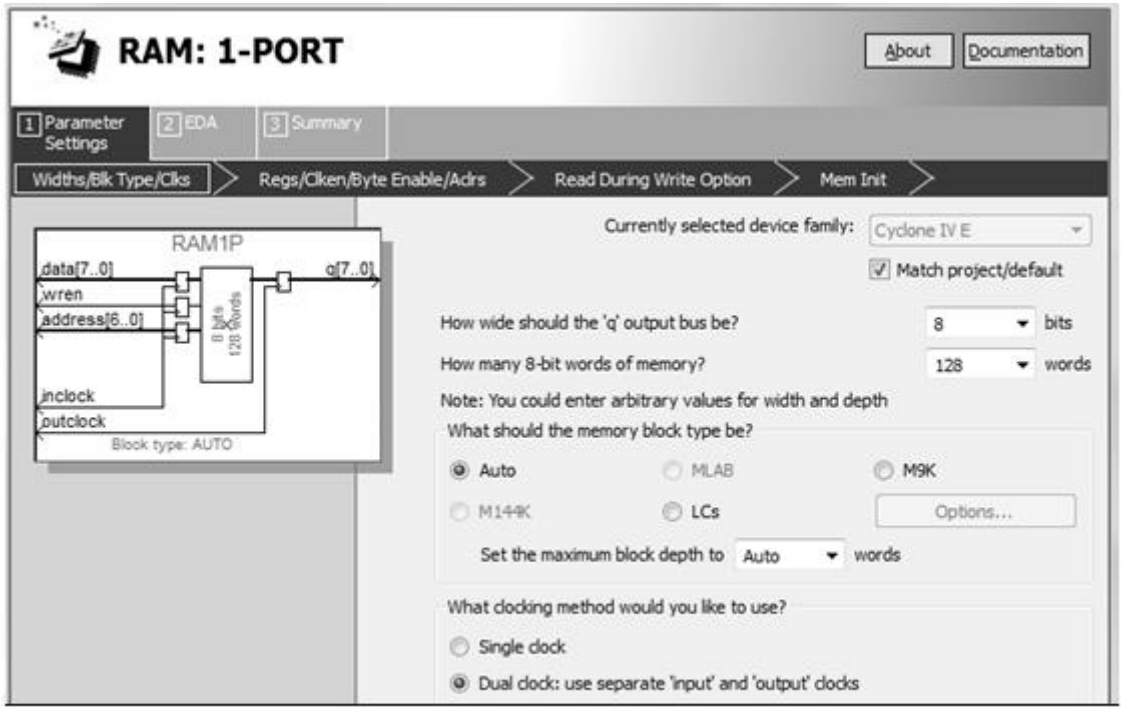

图 6-13　设定 RAM 参数

6.3 LPM 随机存储器的设置和调用

6.3.2 LPM_RAM的设置和调用

图 6-14　设定 RAM 仅输入时钟控制

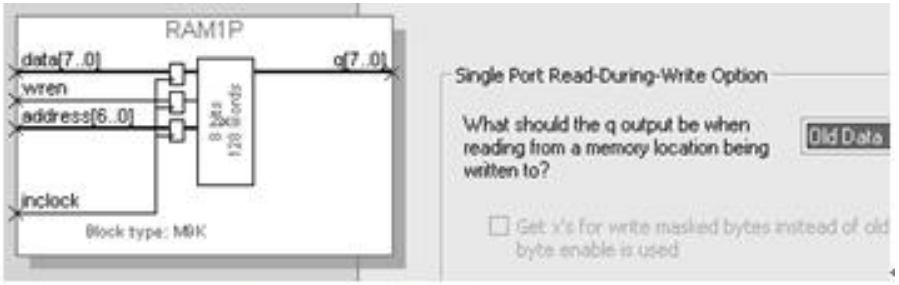

图 6-15　设定在写入同时读出原数据：Old Data

6.3 LPM 随机存储器的设置和调用

6.3.2 LPM_RAM的设置和调用

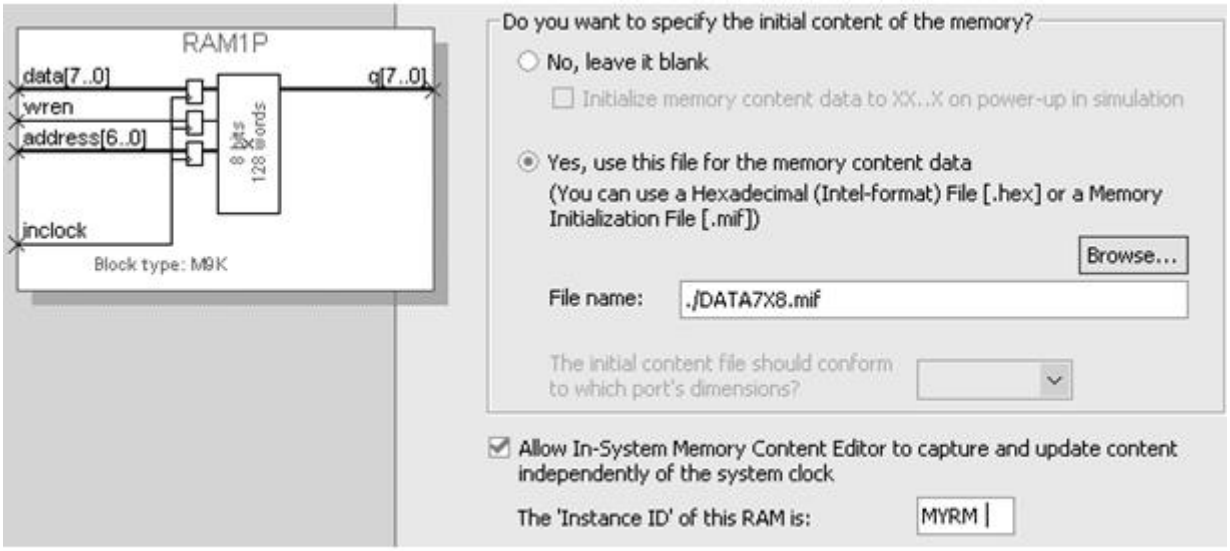

图 6-16　设定初始化文件和允许在系统编辑

6.3 LPM 随机存储器的设置和调用

6.3.2 LPM_RAM的设置和调用

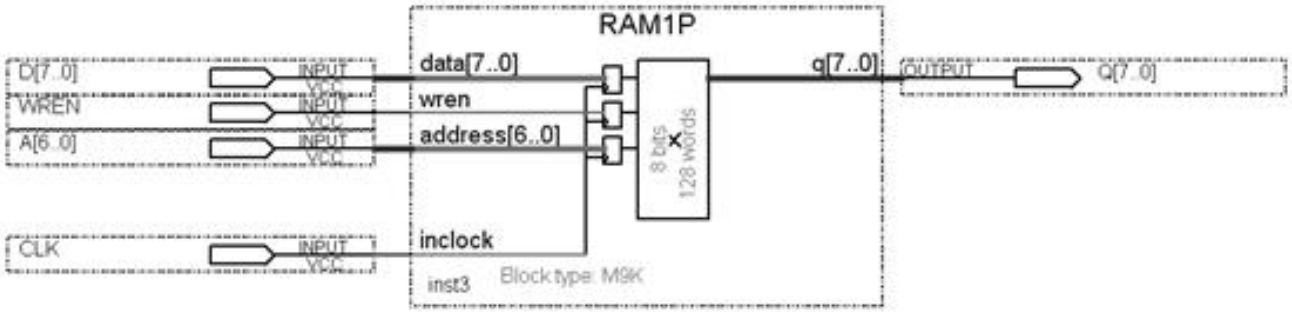

图 6-17　在原理图上连接好的 RAM 模块

6.3 LPM 随机存储器的设置和调用

6.3.3 仿真测试RAM宏模块

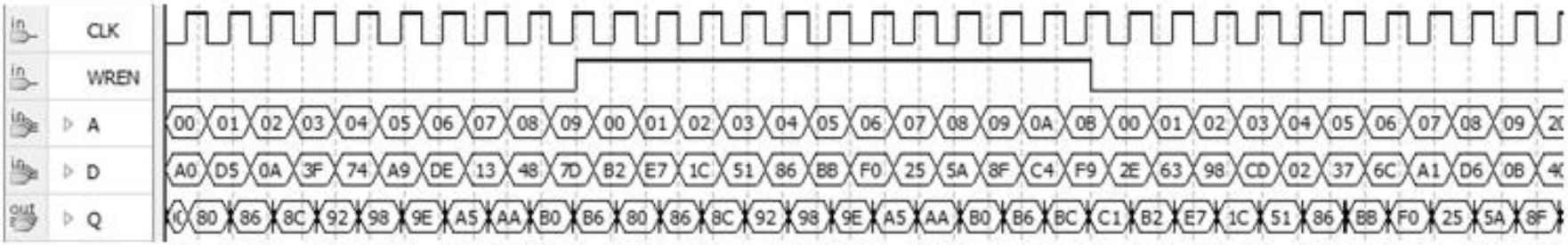

图 6-18　图 6-17 的 RAM 仿真波形

6.3 LPM 随机存储器的设置和调用

6.3.4 存储器的Verilog代码描述

【例 6-6】

```
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
```

【例 6-7】

```
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
```

6.3 LPM 随机存储器的设置和调用

6.3.4 存储器的Verilog代码描述

1．存储器端口描述

```
module RAM78(output[7:0] Q,input[7:0] D,input[6:0] A, input CLK,WREN);
```

2．存储器的Verilog一般描述

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

6.3 LPM 随机存储器的设置和调用

6.3.4 存储器的Verilog代码描述

2. 存储器的Verilog一般描述

```
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
```

6.3 LPM 随机存储器的设置和调用

6.3.4 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 6-8】

```
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
```

6.3 LPM 随机存储器的设置和调用

6.3.4 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
```

6.3 LPM 随机存储器的设置和调用

6.3.5 存储器设计的结构控制

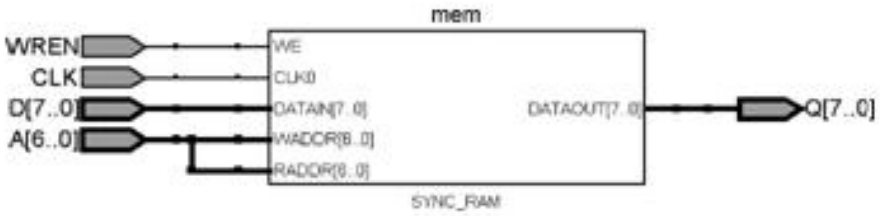

图 6-19　例 6-6 的 RTL 电路模块图

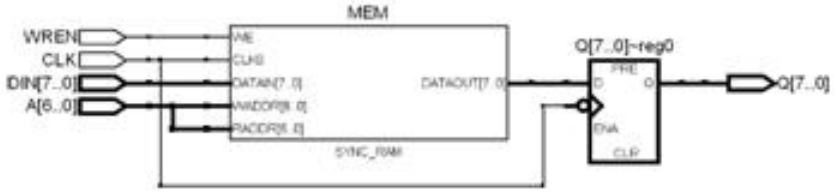

图 6-20　例 6-7 的 RTL 电路模块图

6.3 LPM 随机存储器的设置和调用

6.3.5 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 6-21　例 6-6 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 6-22　例 6-7 的编译报告

6.4 LPM_ROM的定制和使用示例

6.4.1 简易正弦信号发生器设计

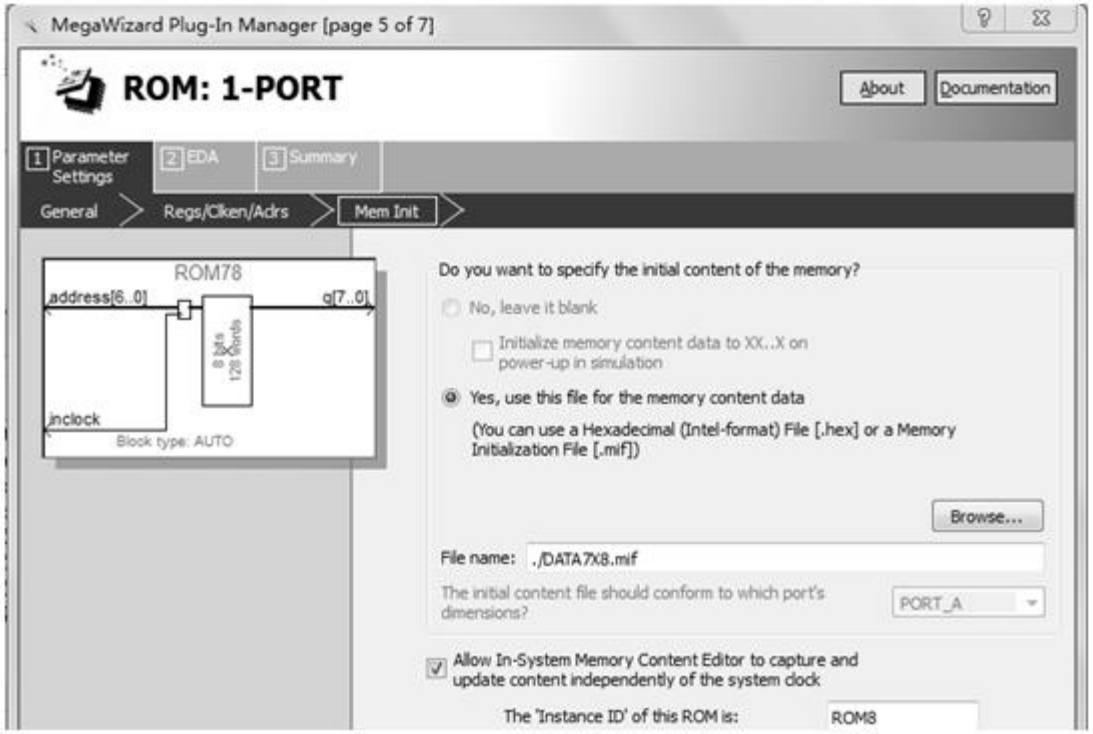

图 6-23　加入初始化配置文件并允许在系统访问 ROM 内容

6.4 LPM_ROM的定制和使用示例

6.4.1 简易正弦信号发生器设计

图 6-24　正弦信号发生器结构框图

6.4 LPM_ROM的定制和使用示例

6.4.1 简易正弦信号发生器设计

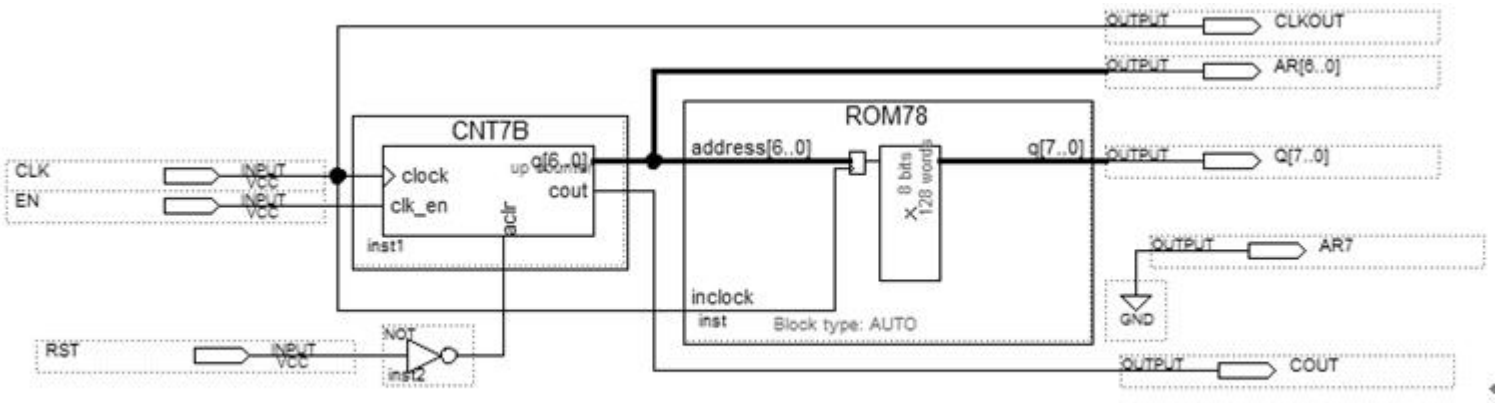

图 6-25　正弦信号发生器电路原理图

6.4 LPM_ROM的定制和使用示例

6.4.1 简易正弦信号发生器设计

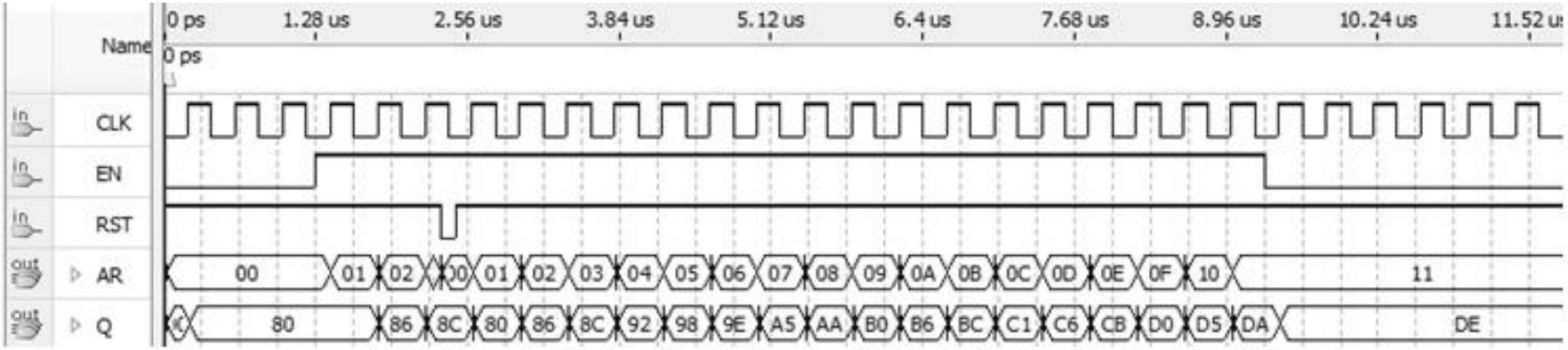

图 6-26　图 6-25 电路仿真波形

6.4 LPM_ROM的定制和使用示例

6.4.2 正弦信号发生器硬件实现和测试

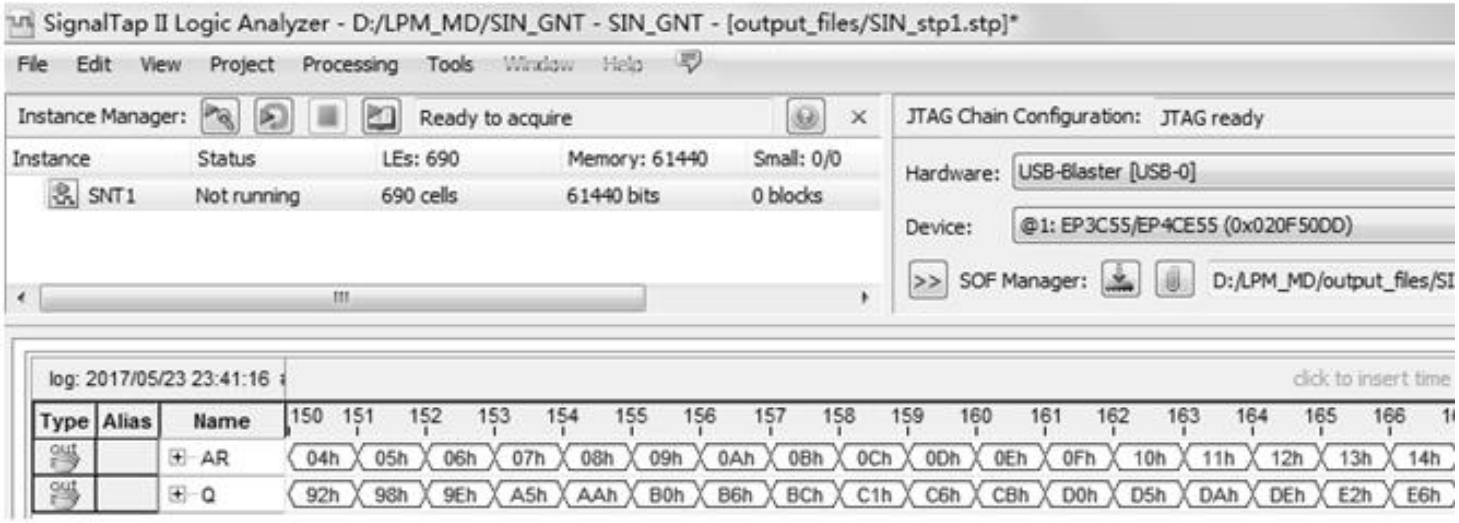

图 6-27　正弦信号发生器数据输出的 SignalTap II 实时测试界面

6.4 LPM_ROM的定制和使用示例

6.4.2 正弦信号发生器硬件实现和测试

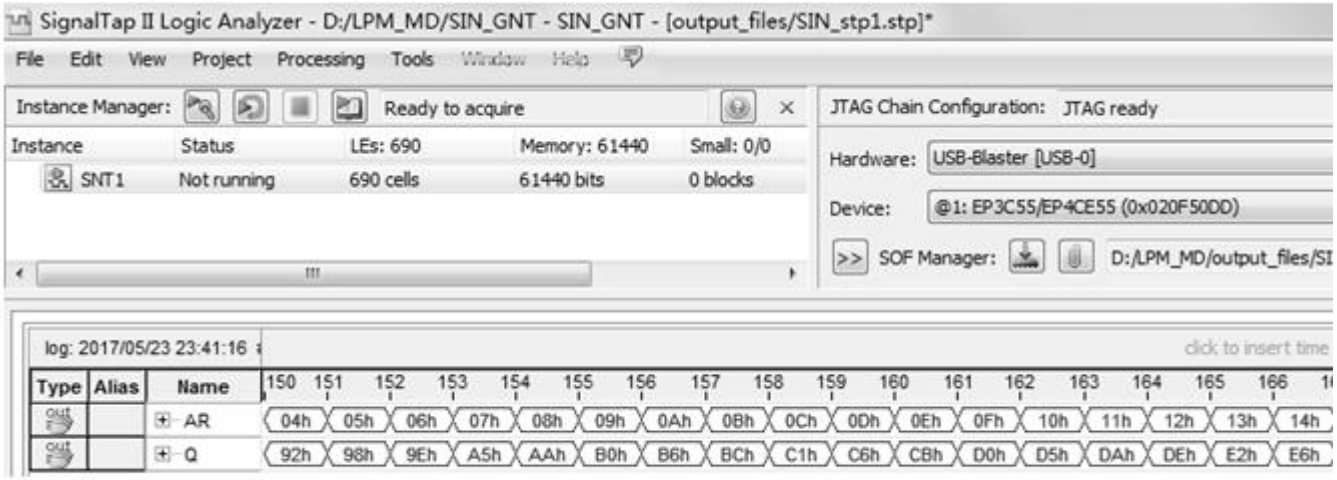

图 6-27　正弦信号发生器数据输出的 SignalTap II 实时测试界面

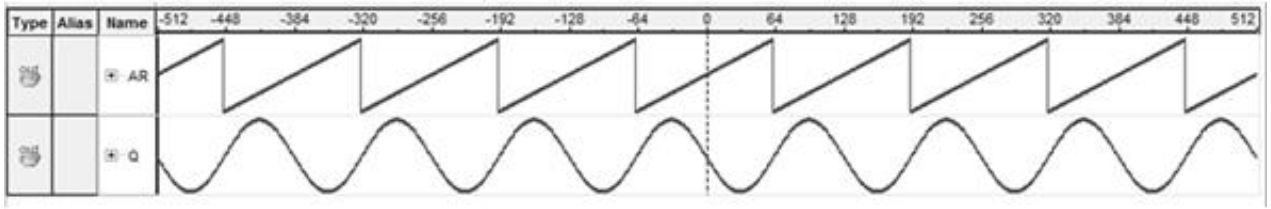

图 6-28　正弦信号发生器的 SignalTap II 的波形显示图

6.5 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

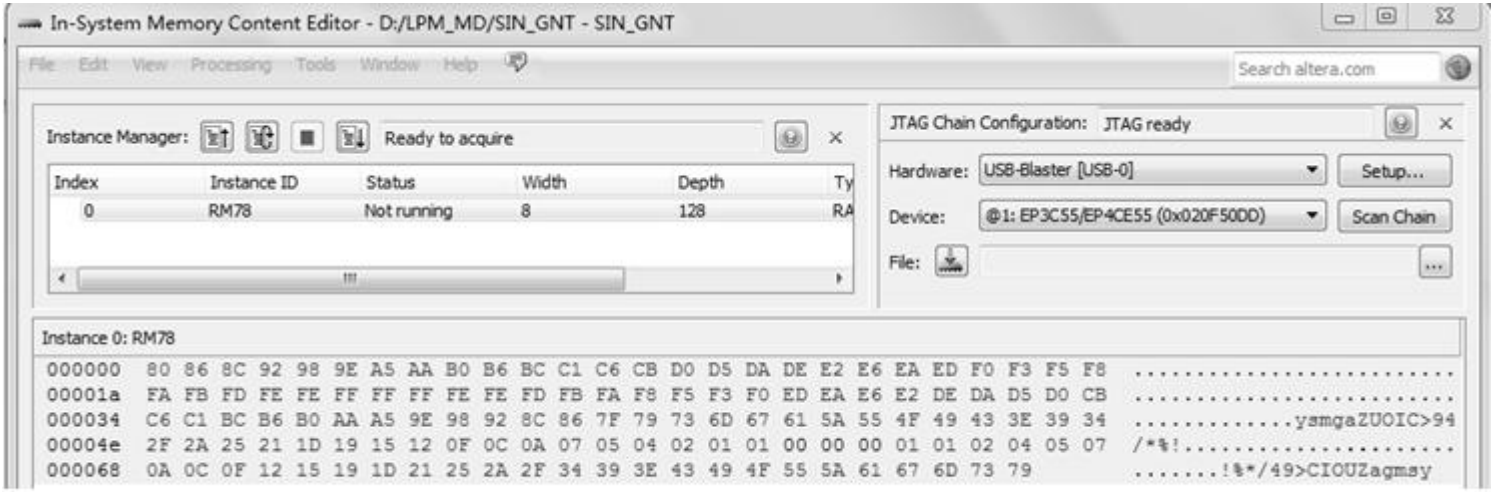

图 6-29 In-System Memory Content Editor 编辑窗，从 FPGA 中的 ROM 读取波形数据

6.5 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

Instance 0: RM78

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
00001a	FA	FB	FD	FE	FE	FF	FF	FF	FE	FE	FD	FB	FA	F8	F5	F3	F0	ED	EA	E6	E2	DE	DA	D5	D0	CB
000034	C6	C1	BC	B6	B0	AA	A5	9E	98	92	8C	86	7F	79	73	6D	67	61	5A	55	4F	49	43	3E	39	34
00004e	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
000068	0A	0C	0F	12	15	19	1D	21	25	2A	2F	34	39	3E	43	49	4F	55	5A	61	67	6D	73	79		

图 6-30　在此将编辑好的数据载入 FPGA 中的 ROM 内

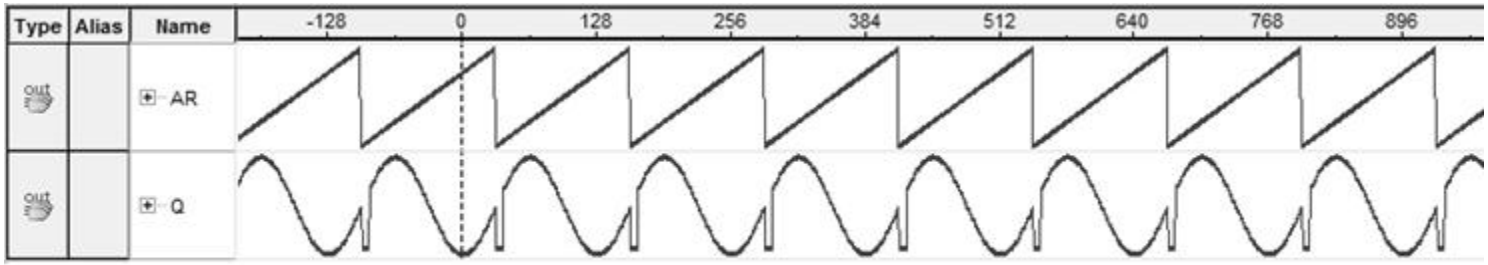

图 6-31　SignalTap II 测得的数据波形

（4）输入输出数据文件

6.6 LPM嵌入式锁相环调用

6.6.1 建立嵌入式锁相环元件

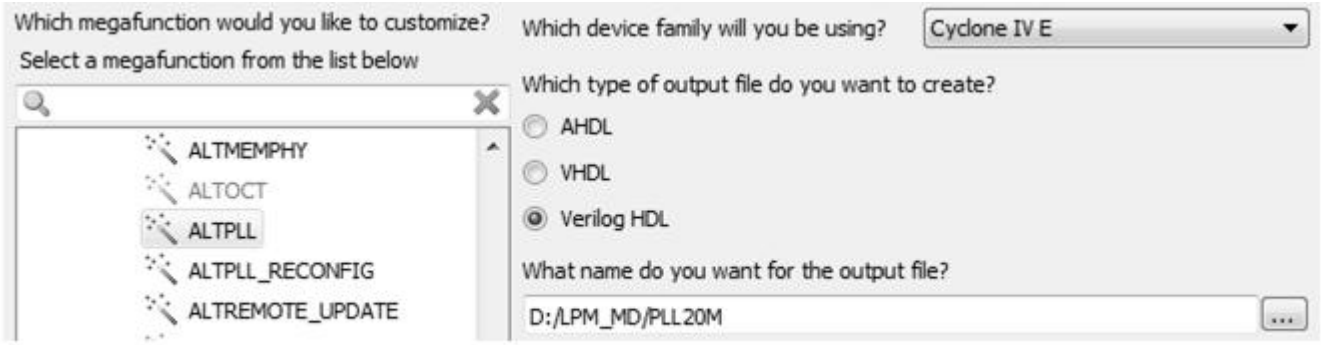

图 6-32　选择锁相环 ALTPLL

6.6 LPM嵌入式锁相环调用

6.6.1 建立嵌入式锁相环元件

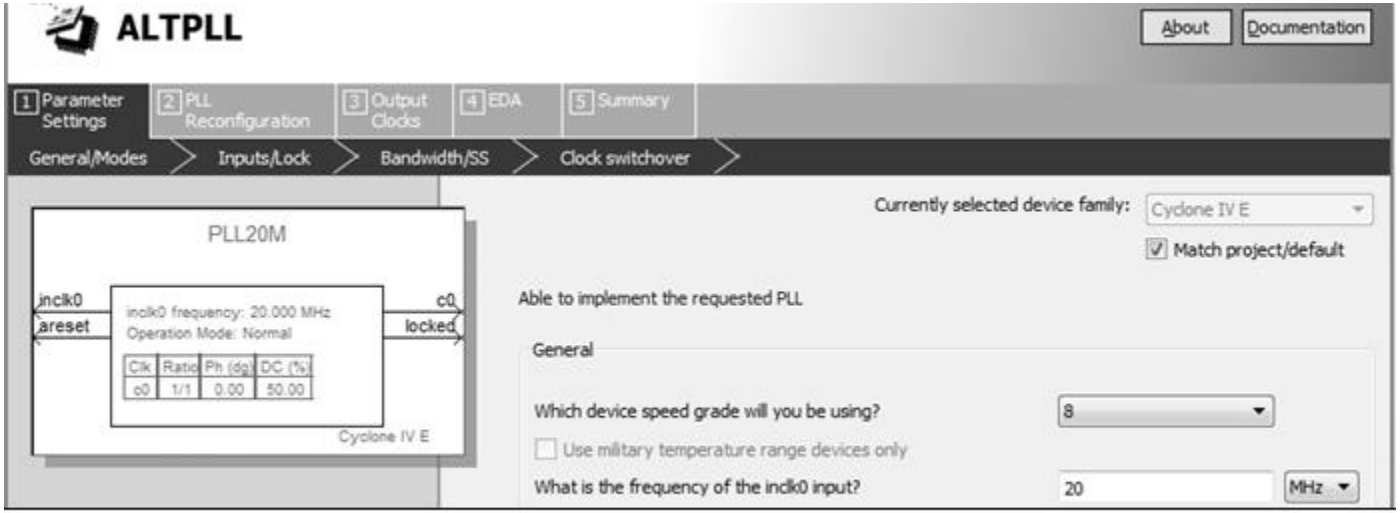

图 6-33　选择输入参考时钟 inclk0 为 20MHz

6.6 LPM嵌入式锁相环调用

6.6.1 建立嵌入式锁相环元件

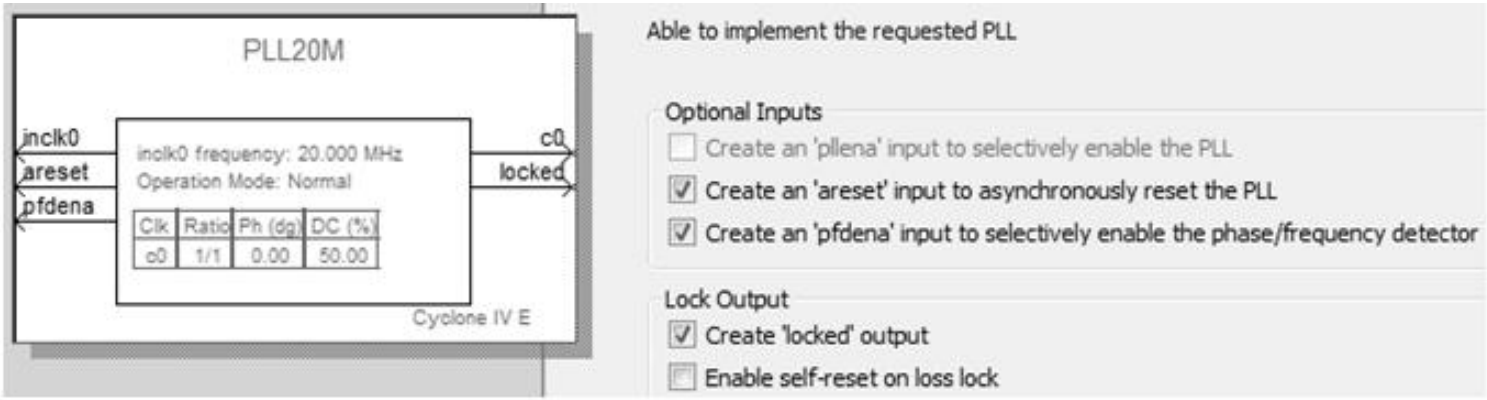

图 6-34　选择锁相环的控制信号

6.6 LPM嵌入式锁相环调用

6.6.1 建立嵌入式锁相环元件

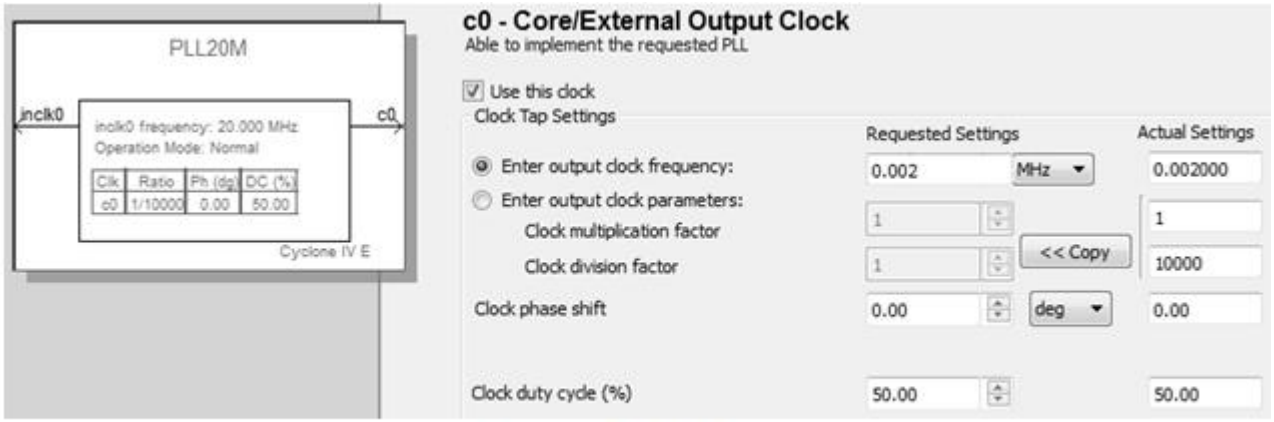

图 6-35　选择 c0 的输出频率为 0.002MHz

6.6 LPM嵌入式锁相环调用

6.6.1 建立嵌入式锁相环元件

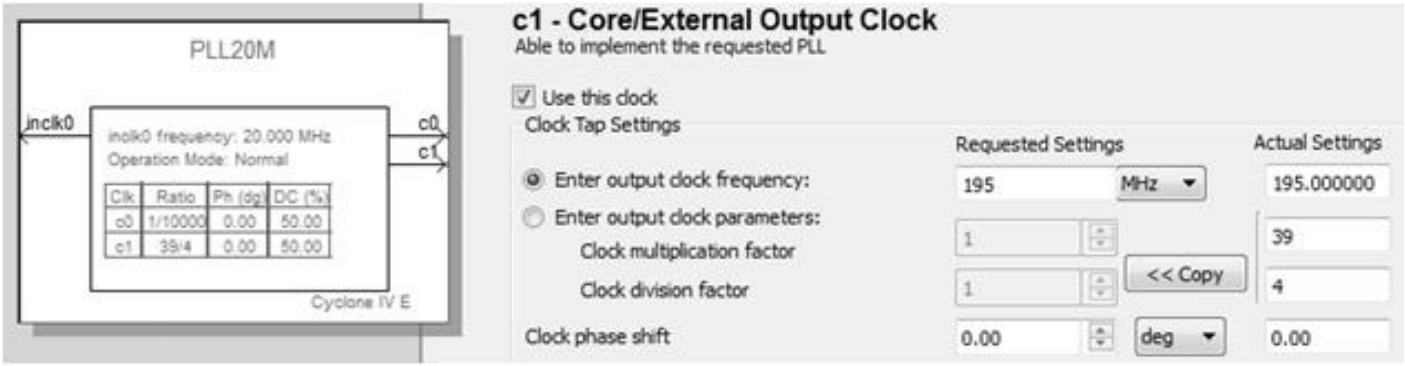

图 6-36　输出第二个时钟信号 c1

6.6 LPM嵌入式锁相环调用

6.6.1 建立嵌入式锁相环元件

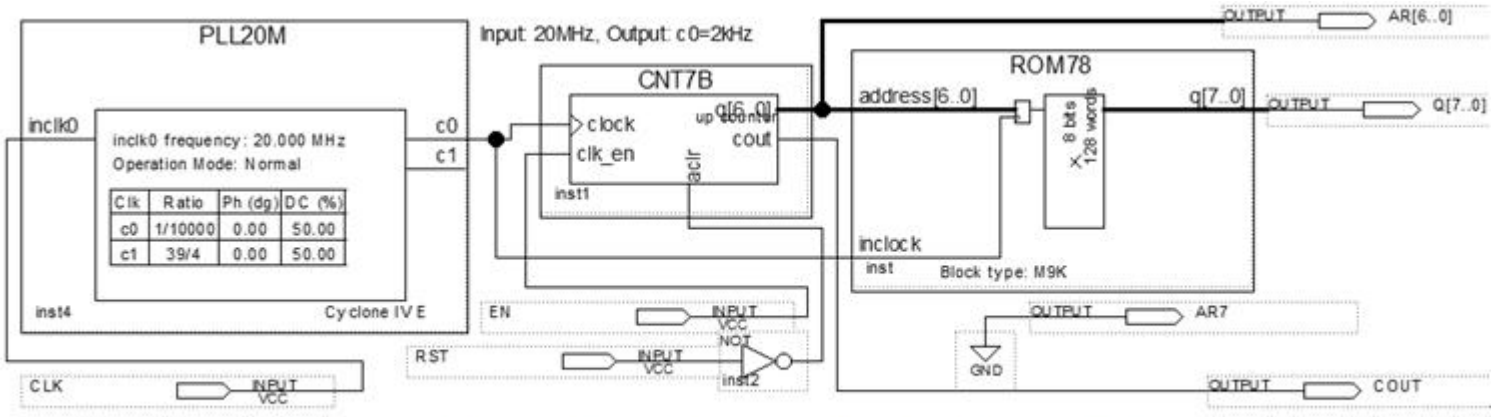

图 6-37　采用了嵌入式锁相环作时钟的正弦信号发生器电路

6.6.2 测试锁相环

6.7 In-System Sources and Probes Editor使用方法

（1）在顶层设计中嵌入In-System Sources and Probes模块

（2）设定参数

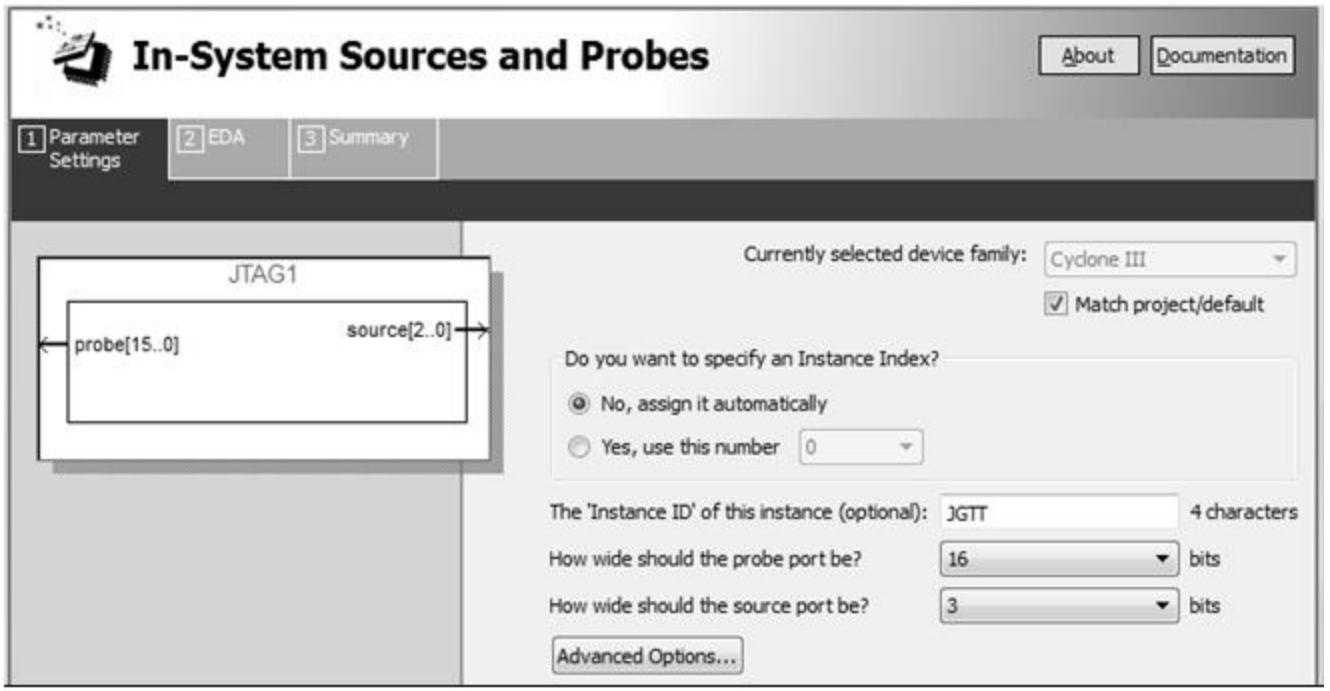

图 6-38　为 In-System Sources and Probes 模块设置参数

6.7 In-System Sources and Probes Editor使用方法

（3）与需要测试的电路系统连接好

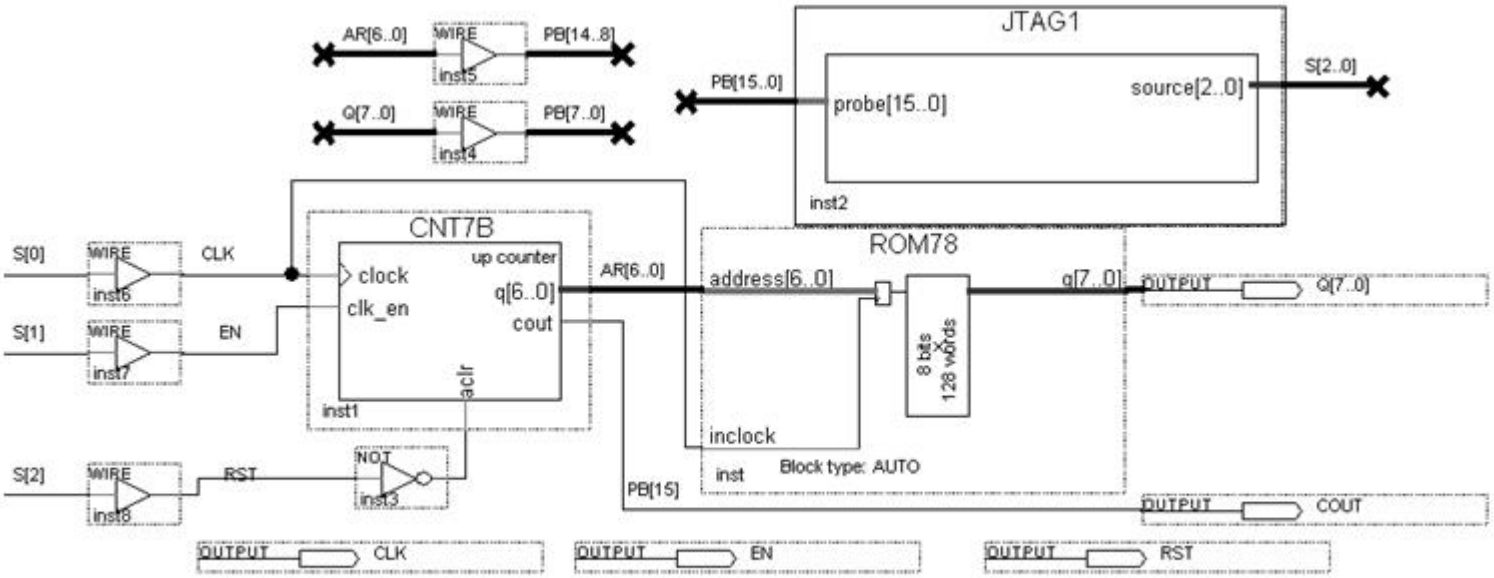

图 6-39　在电路中加入 In-System Sources and Probes 测试模块

6.7 In-System Sources and Probes Editor使用方法

（4）调用In-System Sources and Probes Editor

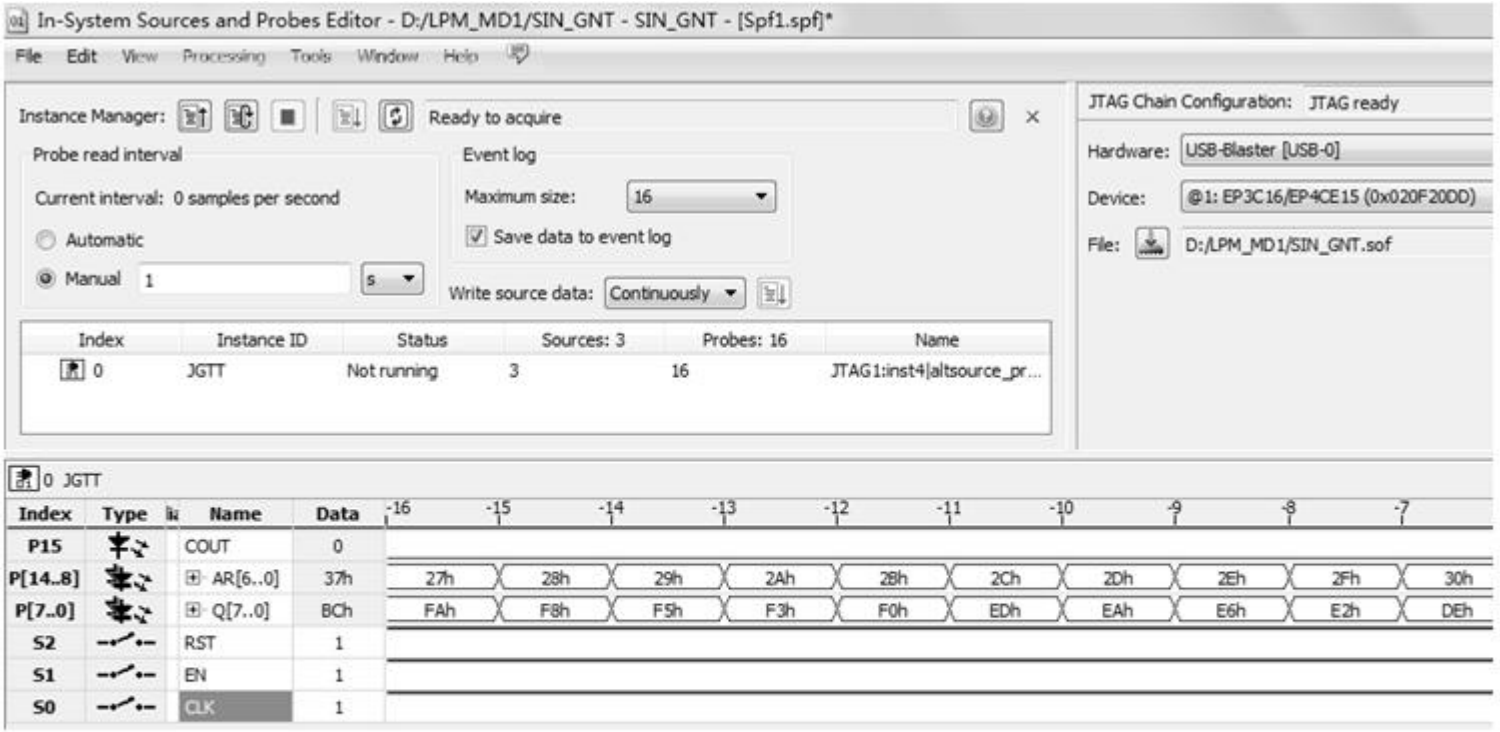

图 6-40　In-System Sources and Probes Editor 的测试情况

进制数以十六进制数格式表达，可右击 Q[7..0]总线表述处，在弹出的下拉菜单中选择 Bus

6.8 NCO核数控振荡器使用方法

（1）定制NCO

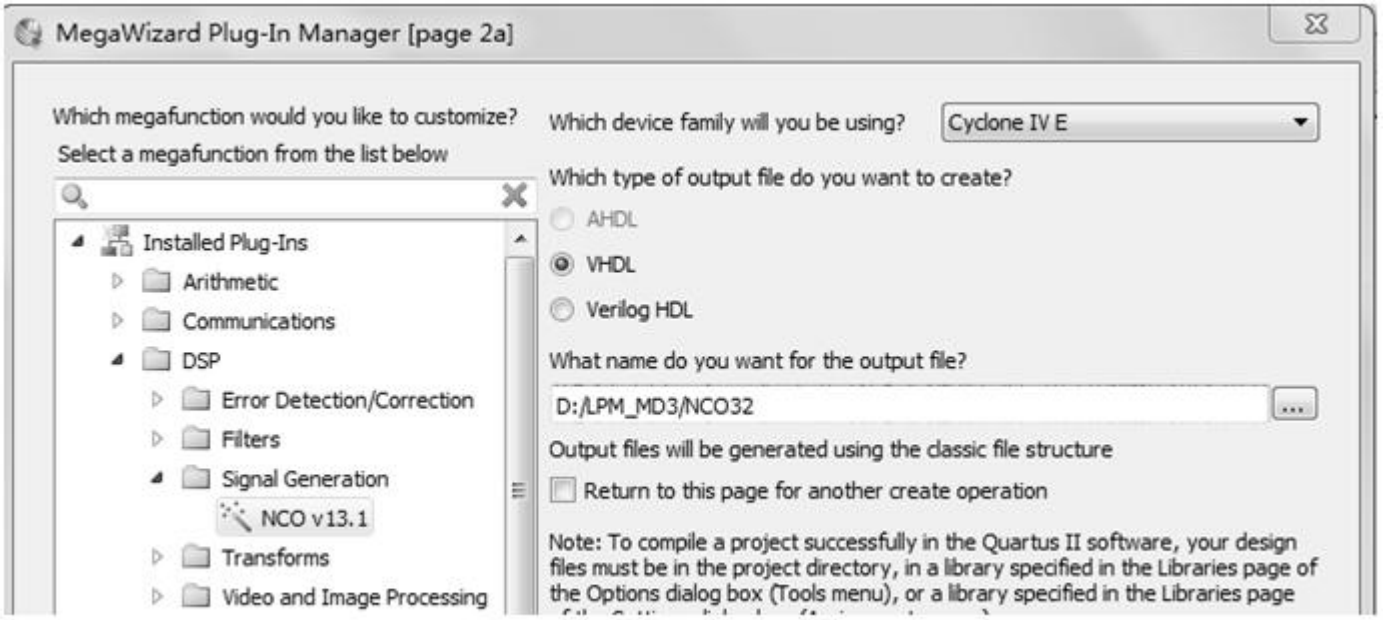

图 6-41　打开 Core 设置管理窗口选择 NCO 核

6.8 NCO核数控振荡器使用方法

（2）进入Core文件生成选择窗

图 6-42 进入 Core 文件生成选择窗口

6.8 NCO核数控振荡器使用方法

（3）设置参数

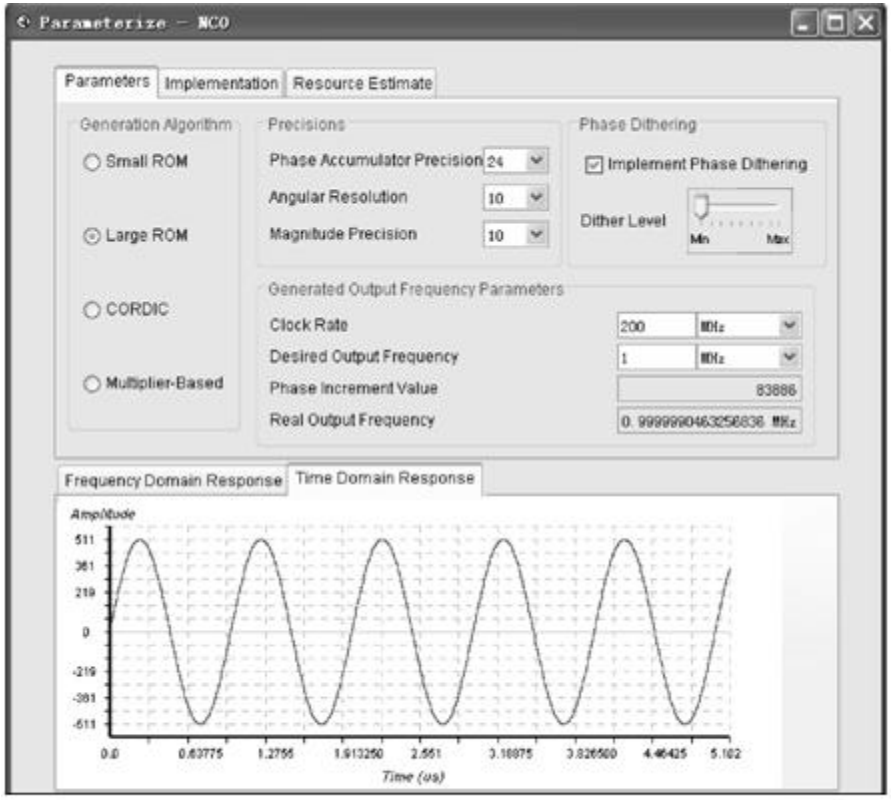

图 6-43　设置 NCO 参数

6.8 NCO核数控振荡器使用方法

（3）设置参数

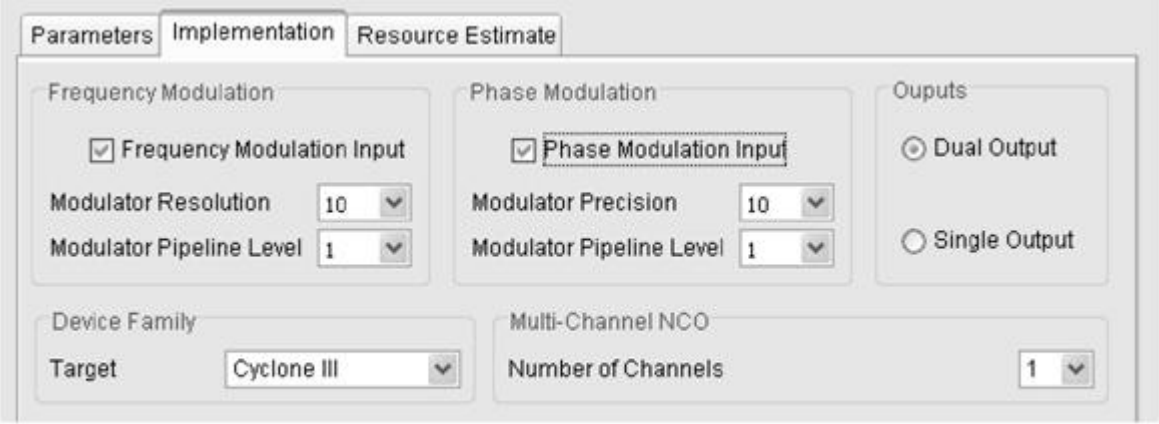

图 6-44　设置 NCO 参数

6.8 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

（6）选择目标器件，然后对生成的模块进行编译及功能检测

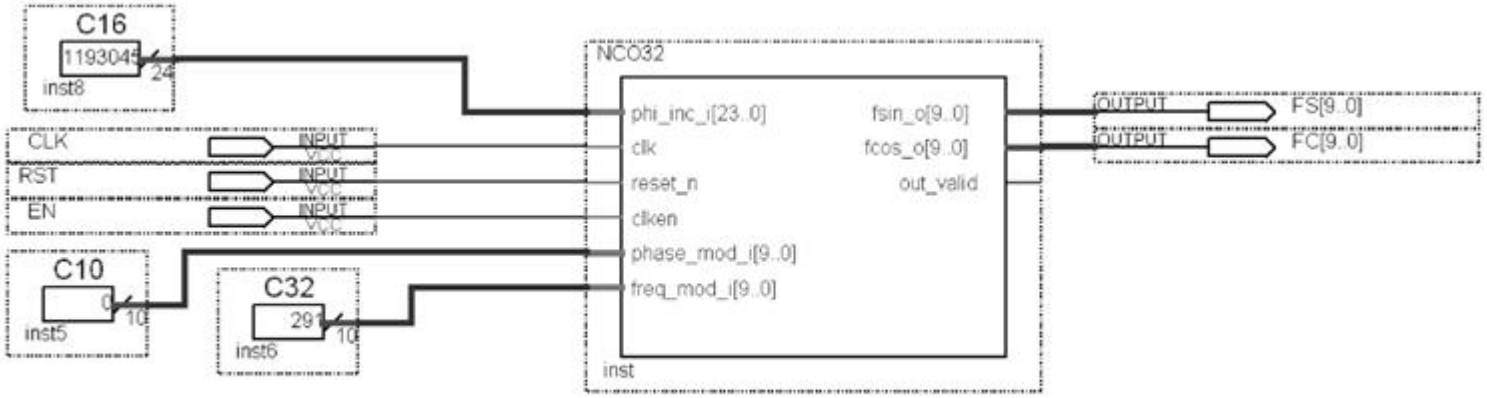

图 6-45　测试 NCO 的电路

6.8 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

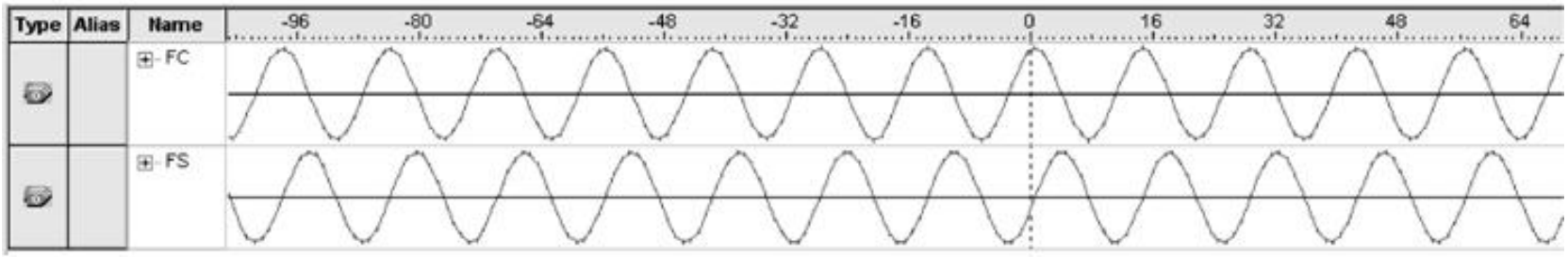

图 6-46　NCO 的逻辑分析仪测试波形

6.9 FIR核使用方法

$$H(z)=\sum_{n=0}^{N-1}h(n)z^{-n} \qquad (6\text{-}1)$$

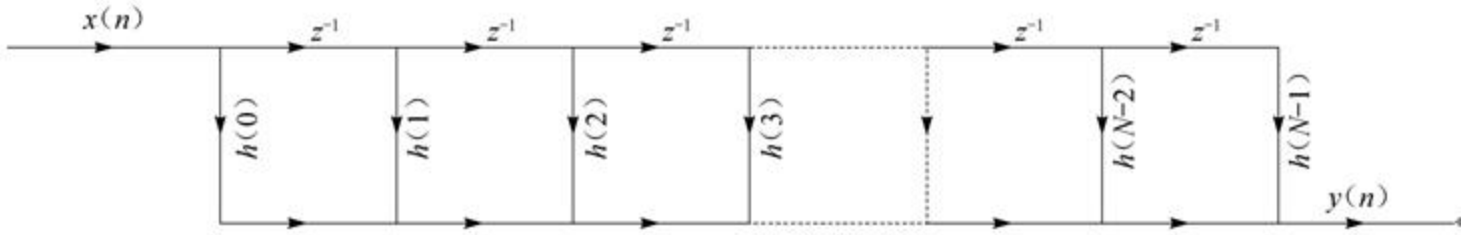

图 6-47　直接型 FIR 滤波器结构

6.9 FIR核使用方法

$$Y(n)=\sum_{m=0}^{N-1}h(m)x(n-m) \qquad (6\text{-}2)$$

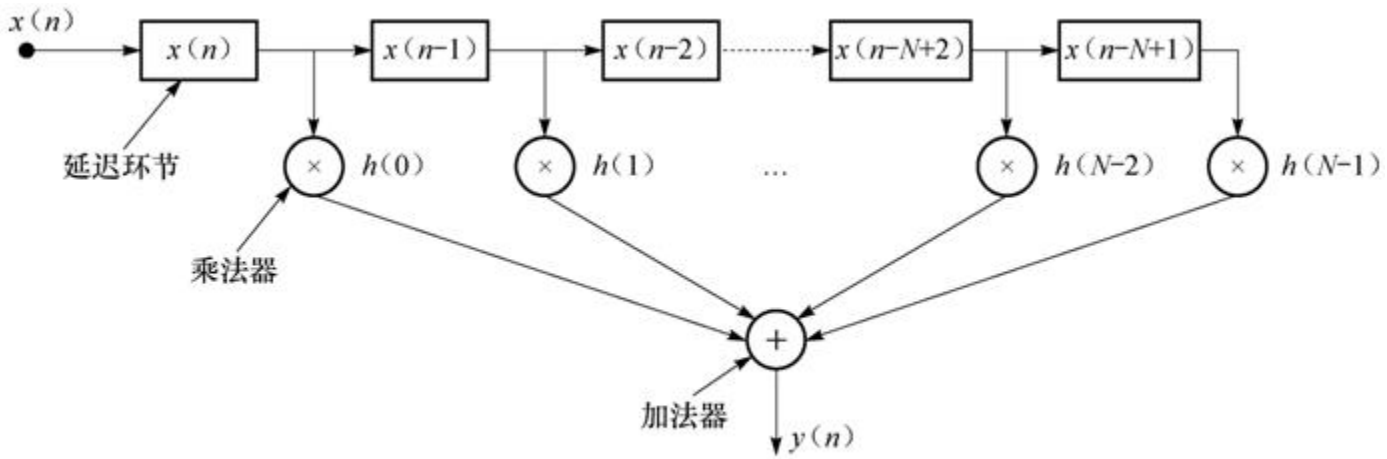

图 6-48　直接型 FIR 实现结构

6.9 FIR核使用方法

图 6-49　FIR 滤波器设计示意

6.10 DDS实现原理与应用

6.10.1 DDS原理

$$S_{out} = A\sin\omega t = A\sin(2\pi f_{out} t) \qquad (6\text{-}3)$$

$$\theta = 2\pi f_{out} t \qquad (6\text{-}4)$$

$$\Delta\theta = 2\pi f_{out} T_{clk} = \frac{2\pi f_{out}}{f_{clk}} \qquad (6\text{-}5)$$

$$\frac{B_{\Delta\theta}}{2^N} = \frac{f_{out}}{f_{clk}} \quad , \quad B_{\Delta\theta} = 2^N \cdot \frac{f_{out}}{f_{clk}} \qquad (6\text{-}6)$$

$$S_{out} = A\sin(\theta_{k-1} + \Delta\theta) = A\sin\left[\frac{2\pi}{2^N} \cdot (B_{\theta_{k-1}} + B_{\Delta\theta})\right] = Af_{\sin}(B_{\theta_{k-1}} + B_{\Delta\theta}) \qquad (6\text{-}7)$$

$$B_{\theta_{k-1}} \approx \frac{\theta_{k-1}}{2\pi} \cdot 2^N \qquad (6\text{-}8)$$

6.10 DDS实现原理与应用

6.10.1 DDS原理

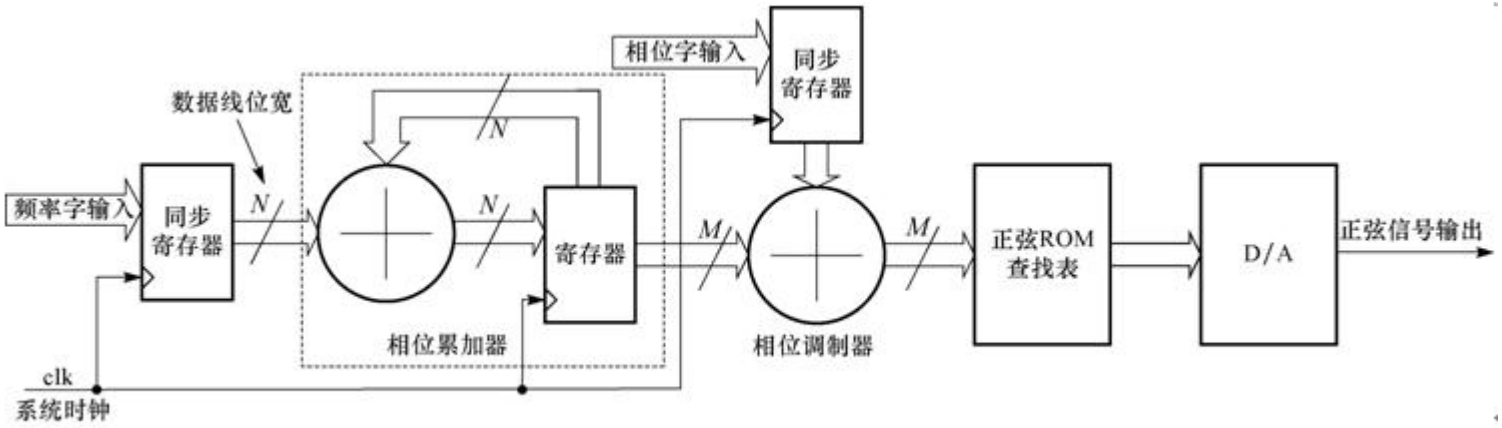

图 6-50　基本 DDS 结构

6.10 DDS实现原理与应用

6.10.1 DDS原理

$$f_{out} = \frac{B_{\Delta\theta}}{2^N} \cdot f_{clk} \qquad (6\text{-}9)$$

$$f_{out} = \frac{f_{clk}}{2^N} \qquad (6\text{-}10)$$

6.10 DDS实现原理与应用

6.10.2 DDS信号发生器设计示例

（1）32位加法器ADDER32

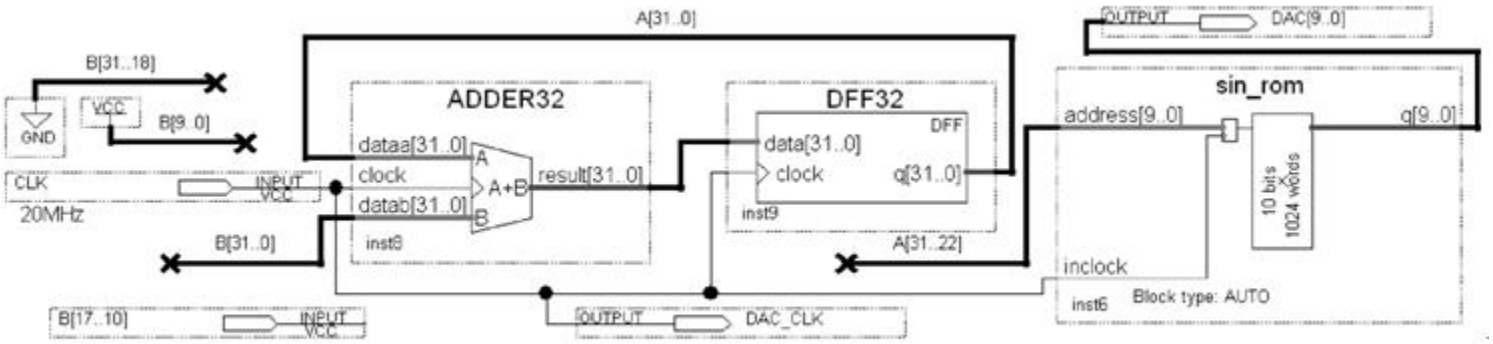

图 6-51　DDS 信号发生器电路顶层原理图

6.10 DDS实现原理与应用

6.10.2 DDS信号发生器设计示例

（2）32位寄存器DFF32

（3）正弦波形数据ROM

（4）频率控制字输入B[17..10]

$$f_{out} = \frac{B[31..0]}{2^{32}} \cdot f_{clk} \qquad (6\text{-}11)$$

（5）DAC驱动数据口DAC[9..0]

实验与设计

6-4 简易数据采集系统设计

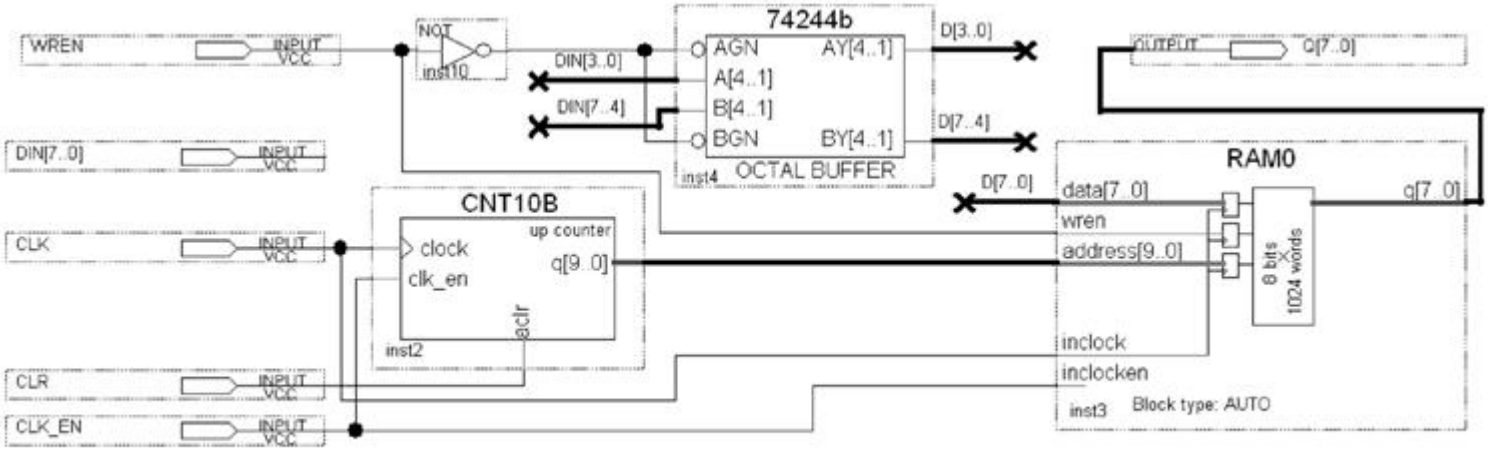

图 6-52　逻辑数据采样电路顶层设计

实验与设计

6-5 移相信号发生器设计

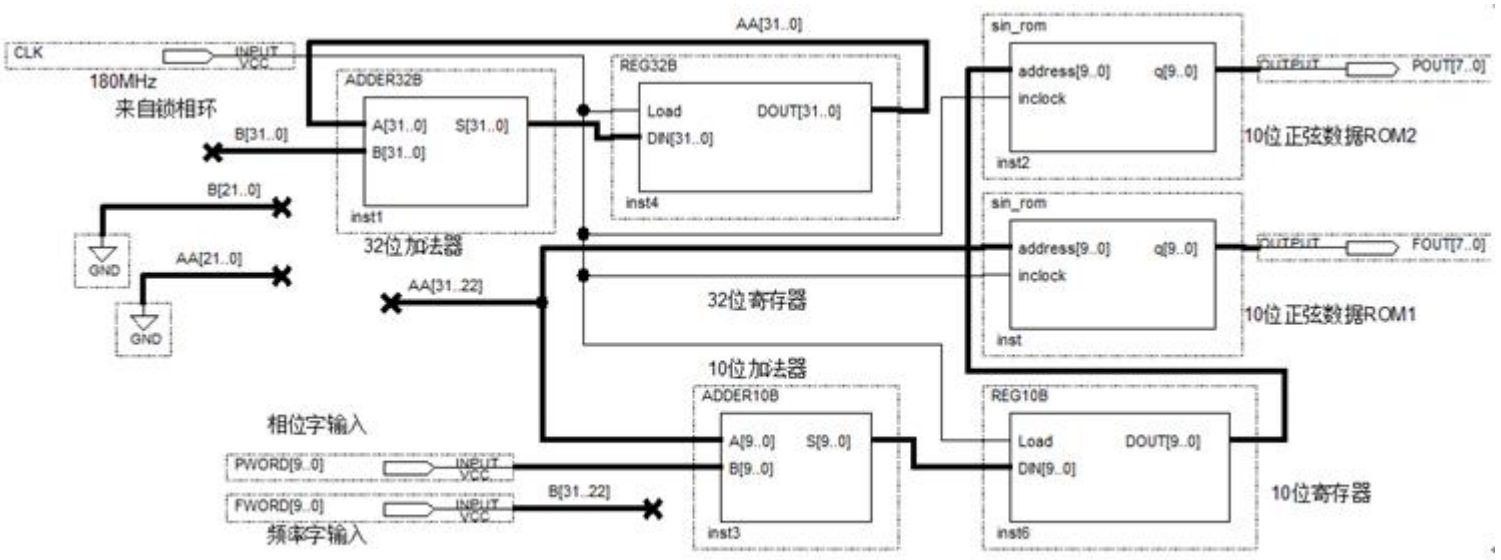

图 6-53　全数字移相信号发生器电路原理图

实验与设计

6-6 16位×16位高速硬件乘法器设计

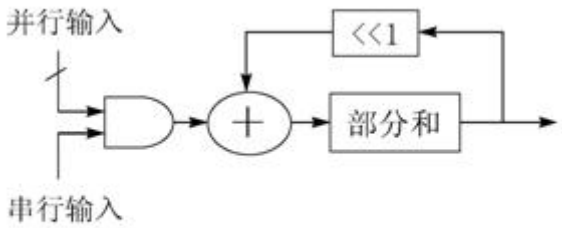

图 6-54 最基本的硬件乘法器

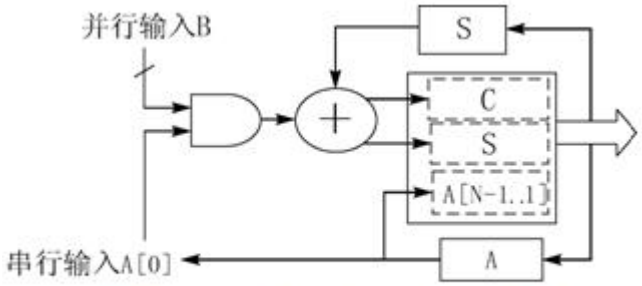

图 6-55 改进后的硬件乘法器

实验与设计

6-7 乐曲硬件演奏电路设计

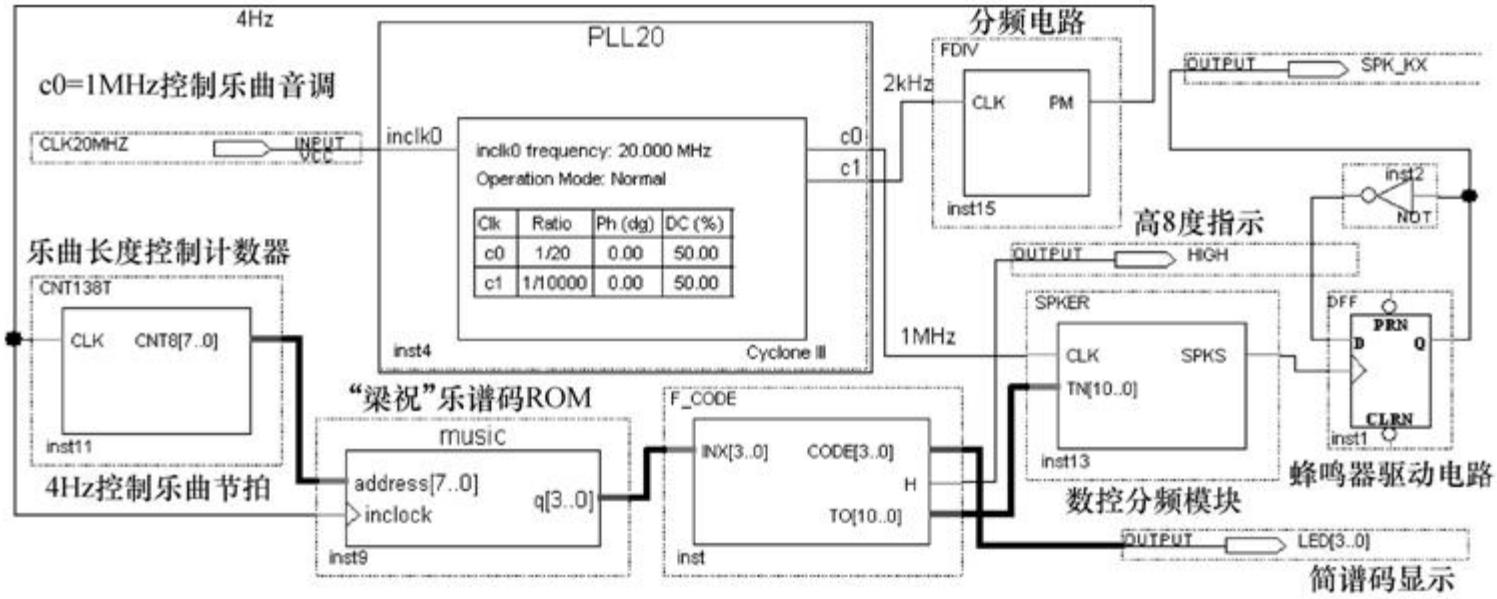

图 6-56 乐曲演奏电路顶层设计

实验与设计

6-7 乐曲硬件演奏电路设计

【例 6-9】

```
module SPKER (CLK, TN, SPKS);
 input CLK;  input[10:0] TN;  output SPKS;
 reg SPKS;  reg[10:0] CNT11;
 always @(posedge CLK)   begin : CNT11B_LOAD //11位可预置计数器
     if (CNT11==11'h7FF)  begin  CNT11=TN;  SPKS<=1'b1;   end
      else  begin   CNT11=CNT11+1;  SPKS<=1'b0;    end    end
endmodule
```

实验与设计

6-7 乐曲硬件演奏电路设计

【例 6-10】

```
module F_CODE (INX, CODE, H, TO);
  input[3:0] INX; output[3:0] CODE; output H; output[10:0] TO;
  reg[10:0] TO;   reg[3:0] CODE;   reg H;
  always @(INX)   begin
  case (INX)          // 译码电路，查表方式，控制音调的预置
   0 : begin TO <= 11'H7FF; CODE<=0; H<=0; end
   1 : begin TO <= 11'H305; CODE<=1; H<=0; end
   2 : begin TO <= 11'H390; CODE<=2; H<=0; end
   3 : begin TO <= 11'H40C; CODE<=3; H<=0; end
   4 : begin TO <= 11'H45C; CODE<=4; H<=0; end
   5 : begin TO <= 11'H4AD; CODE<=5; H<=0; end
   6 : begin TO <= 11'H50A; CODE<=6; H<=0; end
   7 : begin TO <= 11'H55C; CODE<=7; H<=0; end
   8 : begin TO <= 11'H582; CODE<=1; H<=1; end
   9 : begin TO <= 11'H5C8; CODE<=2; H<=1; end
  10 : begin TO <= 11'H606; CODE<=3; H<=1; end
  11 : begin TO <= 11'H640; CODE<=4; H<=1; end
  12 : begin TO <= 11'H656; CODE<=5; H<=1; end
  13 : begin TO <= 11'H684; CODE<=6; H<=1; end
  14 : begin TO <= 11'H69A; CODE<=7; H<=1; end
  15 : begin TO <= 11'H6C0; CODE<=1; H<=1; end
   default : begin TO <= 11'H6C0; CODE<=1; H<=1;    end
  endcase    end
endmodule
```

实验与设计

6-7 乐曲硬件演奏电路设计

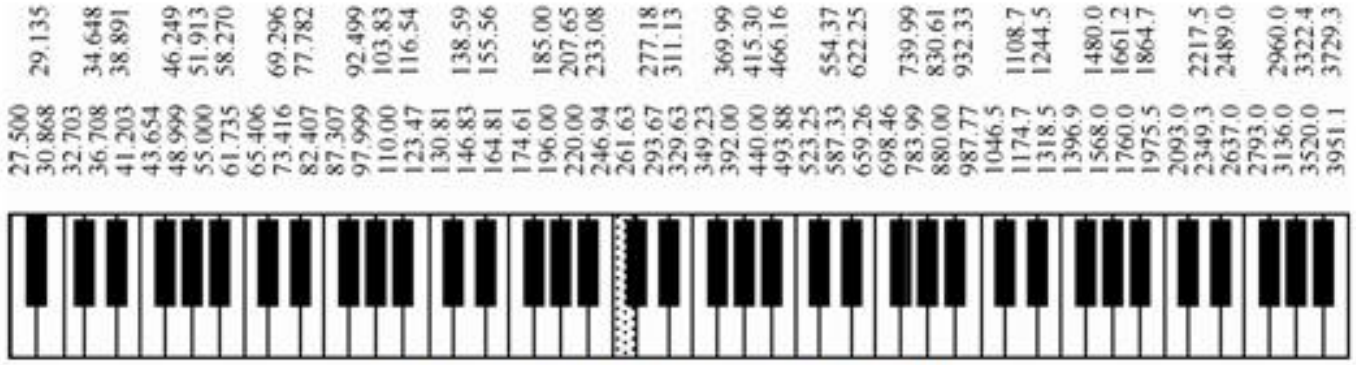

图 6-57 电子琴音阶基频对照图（单位 Hz）

实验与设计

6-7 乐曲硬件演奏电路设计

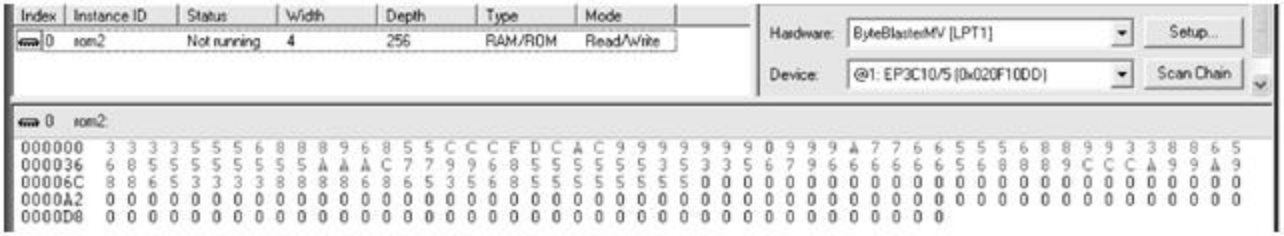

图 6-58 In-System Memory Content Editor 对 MUSIC 模块的数据读取

实验与设计

6-7 乐曲硬件演奏电路设计

【例 6-11】

```
module FDIV  (CLK,PM);
  input CLK;  output PM;  reg[8:0] Q1;  reg FULL;  wire RST;
    always @(posedge CLK or posedge RST) begin
       if (RST) begin Q1<=0; FULL<=1; end
         else begin Q1 <= Q1+1; FULL<=0;   end   end
    assign RST = (Q1==499); assign PM = FULL;
    assign DOUT = Q1;
endmodule
```

【例 6-12】

```
module CNT138T (CLK, CNT8);
    input CLK;    output[7:0] CNT8;   reg[7:0] CNT;   wire LD;
    always @(posedge CLK or posedge LD)   begin
      if (LD) CNT <= 8'b00000000;  else    CNT<=CNT+1;   end
    assign CNT8=CNT;   assign LD=(CNT==138);
endmodule
```

实验与设计

6-7 乐曲硬件演奏电路设计

【例 6-13】

```
WIDTH = 4;                  //"梁祝"乐曲演奏数据
DEPTH = 256 ;               //实际深度 139
ADDRESS_RADIX = DEC ; //地址数据类是十进制
DATA_RADIX = DEC ;      //输出数据的类型也是十进制
CONTENT  BEGIN              //注意实用文件中要展开以下数据，每一组占一行
00: 3; 01: 3; 02: 3; 03: 3; 04: 5;   05: 5; 06: 5; 07: 6; 08: 8; 09: 8;
10: 8; 11: 9; 12: 6; 13: 8; 14: 5;   15: 5; 16:12; 17: 12;18: 12;19:15;
20:13; 21:12; 22:10; 23:12; 24: 9;   25: 9; 26: 9; 27: 9; 28: 9; 29: 9;
30: 9; 31: 0; 32: 9; 33: 9; 34: 9;   35:10; 36: 7; 37: 7; 38: 6; 39: 6;
40: 5; 41: 5; 42: 5; 43: 6; 44: 8;   45: 8; 46: 9; 47: 9; 48: 3; 49: 3;
50: 8; 51: 8; 52: 6; 53: 5; 54: 6;   55: 8; 56: 5; 57: 5; 58: 5; 59: 5;
60: 5; 61: 5; 62: 5; 63: 5; 64:10;   65:10; 66:10; 67:12; 68: 7; 69: 7;
70: 9; 71: 9; 72: 6; 73: 8; 74: 5;   75: 5; 76: 5; 77: 5; 78: 5; 79: 5;
80: 3; 81: 5; 82: 3; 83: 3; 84: 5;   85: 6; 86: 7; 87: 9; 88: 6; 89: 6;
90: 6; 91: 6; 92: 6; 93: 6; 94: 5;   95: 6; 96: 8; 97: 8; 98: 8; 99: 9;
100:12;101:12;102:12;103:10;104:9;  105: 9;106:10;107: 9;108: 8;109: 8;
110:6; 111:5; 112: 3;113:3; 114:3;  115: 3;116: 8;117: 8;118: 8;119: 8;
120:6; 121:8; 122: 6;123:5; 124:3;  125: 5;126: 6;127: 8;128: 5;129: 5;
130:5; 131:5; 132: 5;133:5; 134:5;  135: 5;136: 0;137: 0;138: 0;
END;
```