第6章

LPM宏模块用法

6.1 调用计数器宏模块示例

6.1.1 计数器LPM模块文本代码的调用

（1）打开LPM宏功能块调用管理器。

图6-1 定制新的宏功能块

6.1 调用计数器宏模块示例

6.1.1 计数器LPM模块文本代码的调用

（1）打开LPM宏功能块调用管理器。

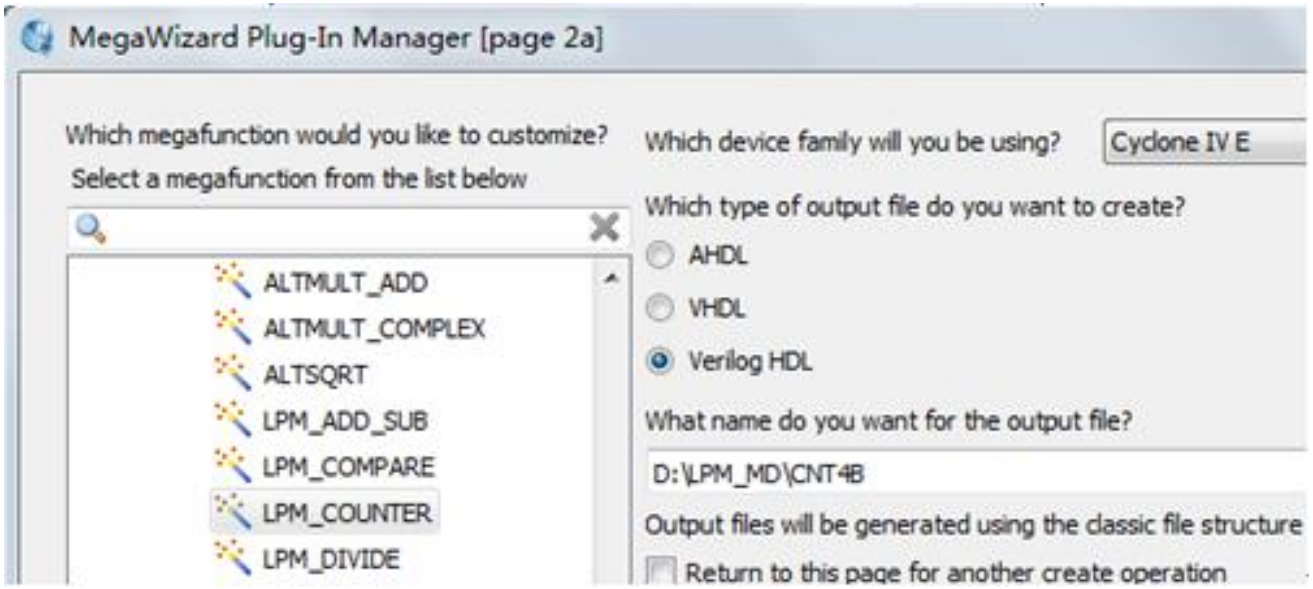

图6-2 LPM宏功能块设定

6.1 调用计数器宏模块示例

6.1.1 计数器LPM模块文本代码的调用

（2）单击Next按钮后打开如 图6-3所示的对话框

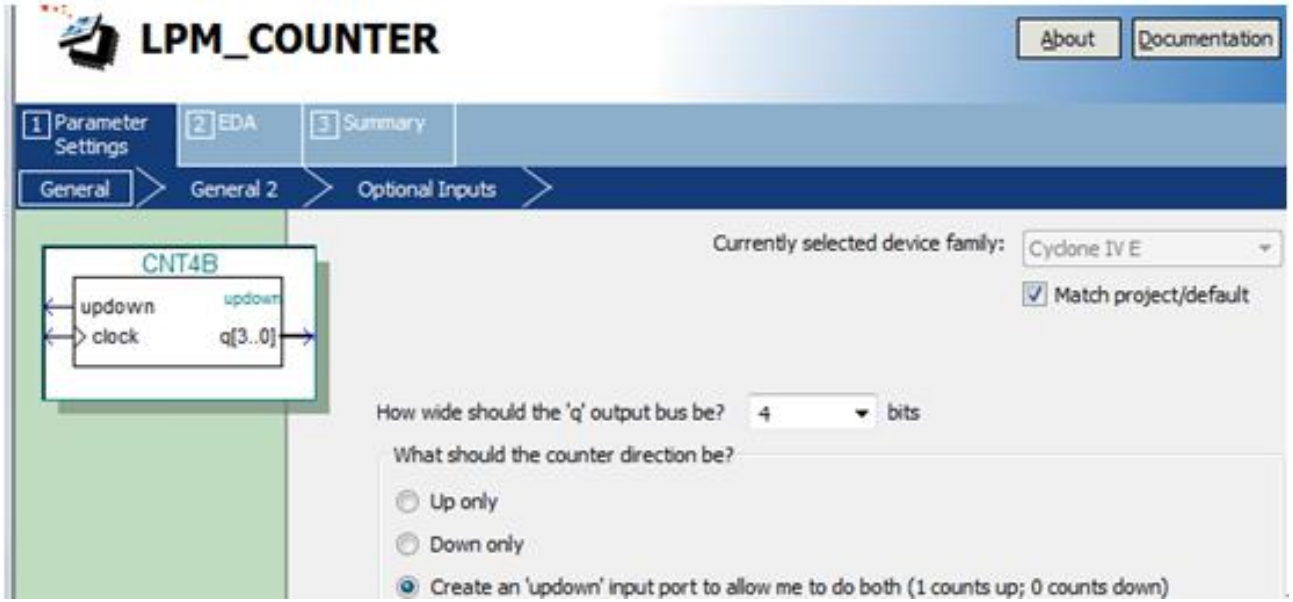

图6-3 设4位可加减计数器

6.1 调用计数器宏模块示例

6.1.1 计数器LPM模块文本代码的调用

（3）再单击Next按钮，打开如图6-4所示的对话框

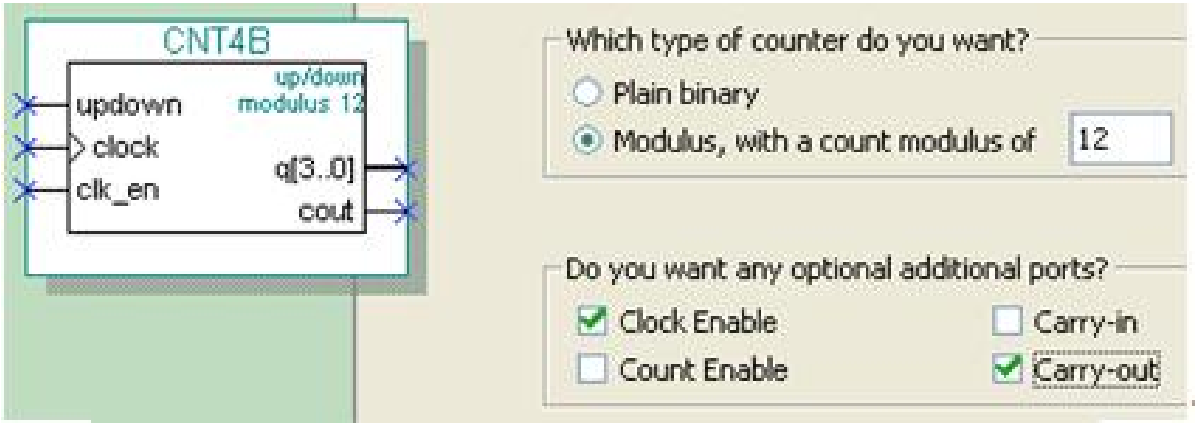

图6-4 设定计数器，含时钟使能和进位输出

6.1 调用计数器宏模块示例

6.1.1 计数器LPM模块文本代码的调用

（4）再单击Next按钮，打开如图6-5所示的对话框

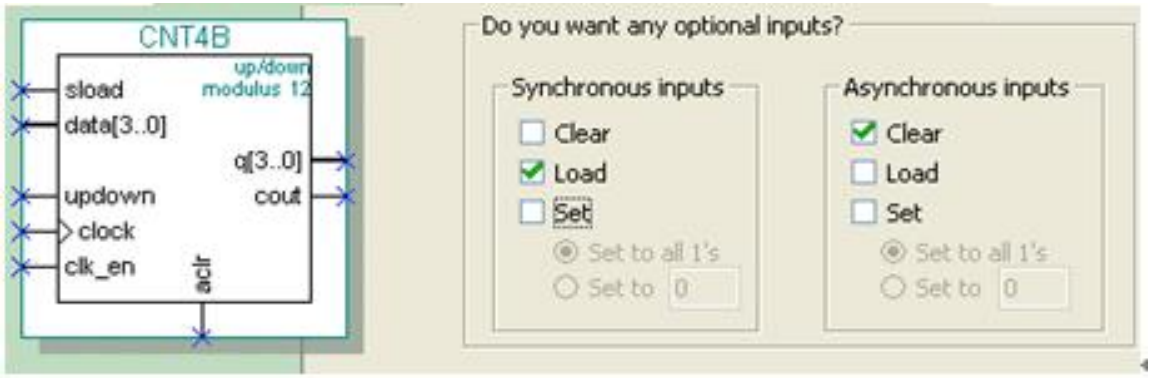

图6-5 加入4位并行数据预置功能

6.1.2 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;                  // 与 assign相同的赋值语句
    wire [3:0] q = sub_wire1[3:0];           // 与 assign 相同的赋值语句
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 调用计数器宏模块示例

6.1.2 LPM计数器代码与参数传递语句应用

【例 6-2】

```
module REG24B (d, clk, q);
    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 调用计数器宏模块示例

6.1.3 创建工程与仿真测试

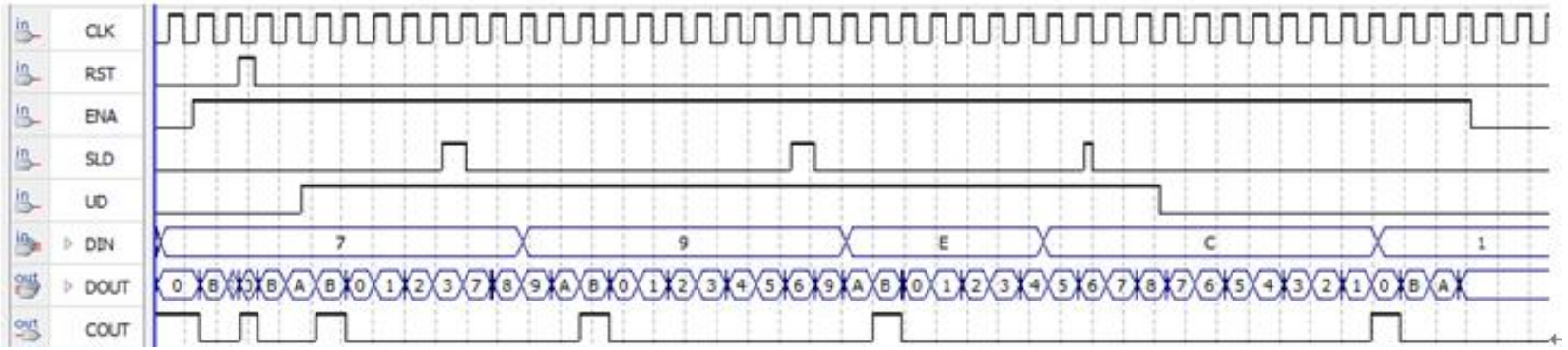

图6-6 CNT4BIT.v的仿真波形

6.2 利用属性控制乘法器构建的示例

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

6.2 利用属性控制乘法器构建的示例

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构建的示例

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

6.2 利用属性控制乘法器构建的示例

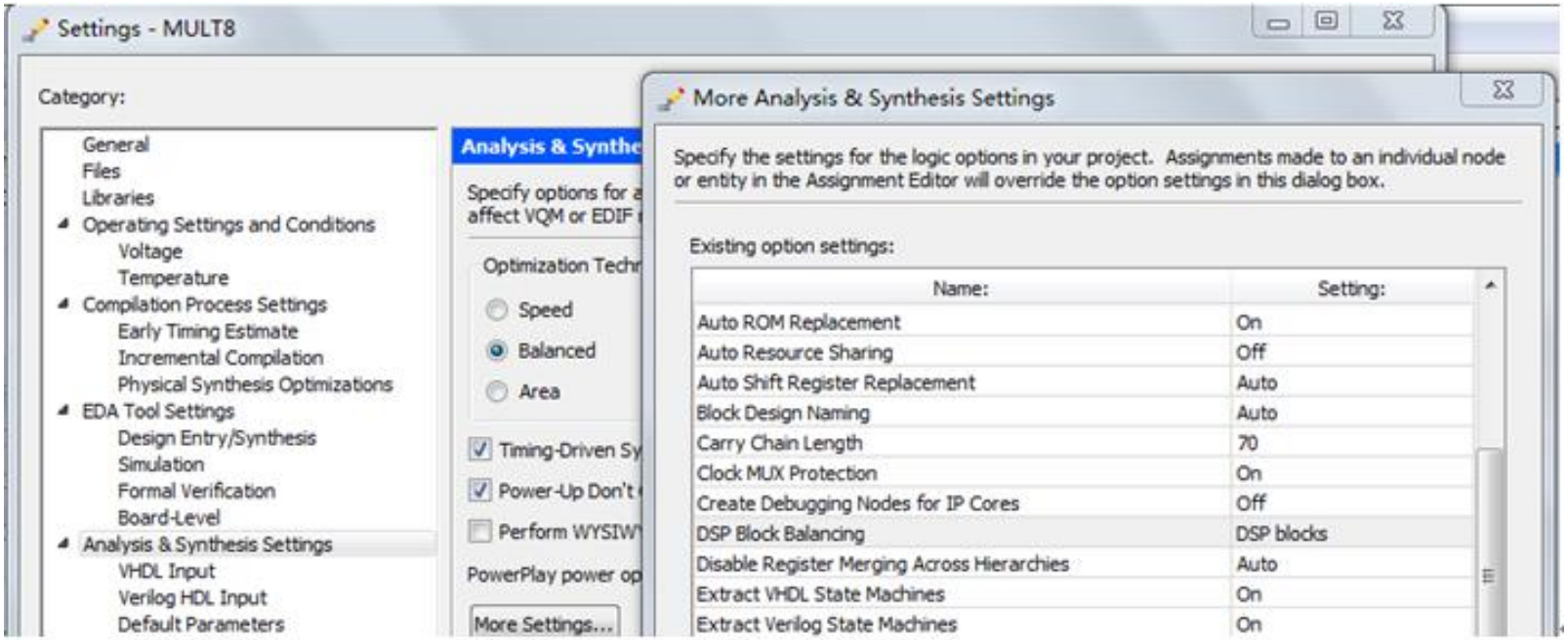

图6-9 选择DSP Block Balancing为DSP blocks

6.3 LPM_RAM宏模块用法

6.3.1 初始化文件及其生成

1. .mif格式文件

（1）直接编辑法。

（2）文件直接编辑法。

（3）高级语言生成。

（4）专用.mif文件生成器。

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 6-10 mif文件编辑窗

6.3 LPM_RAM宏模块用法

6.3.1 初始化文件及其生成

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_RAM宏模块用法

6.3.1 初始化文件及其生成

1. .mif格式文件

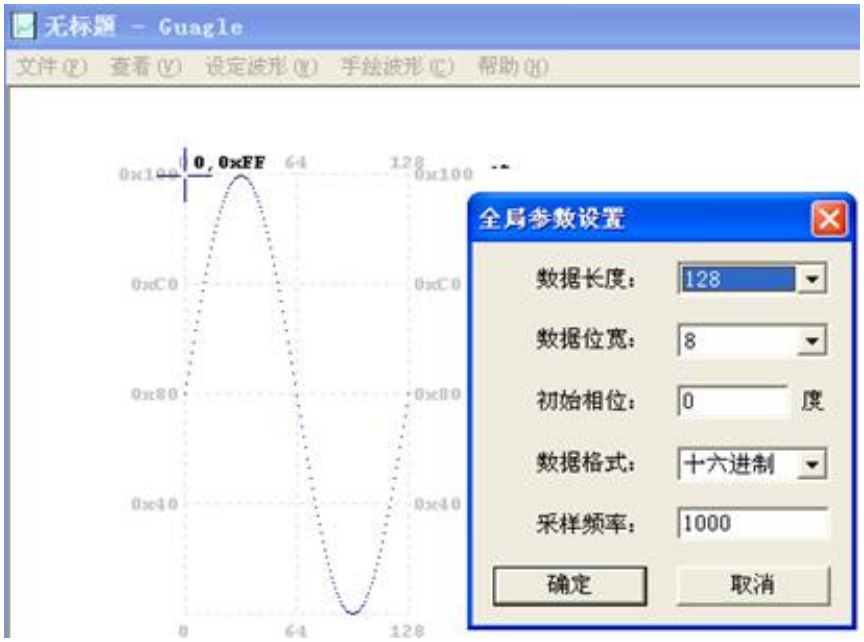

图6-11 利用mif生成器生成.mif正弦波文件

图6-12 打开.mif文件

6.3 LPM_RAM宏模块用法

6.3.1 初始化文件及其生成

2. .hex格式文件

3. .dat格式文件

6.3 LPM_RAM宏模块用法

6.3.2 以原理图方式对LPM_RAM进行调用

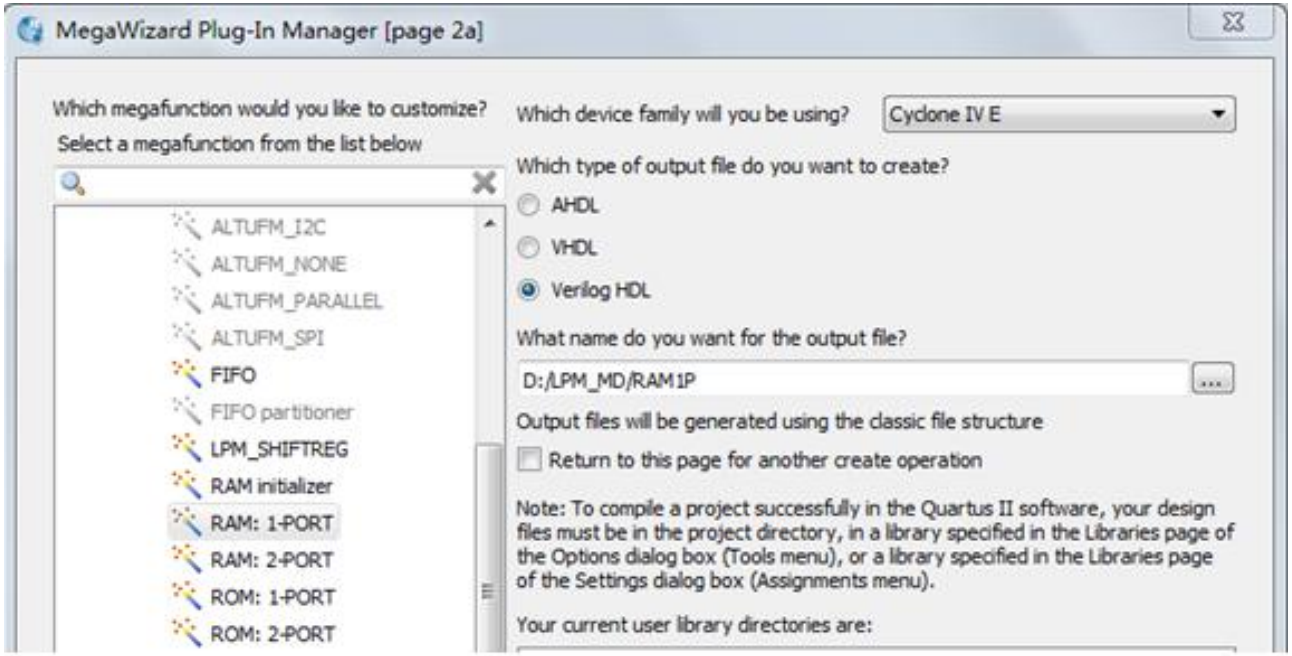

图6-13 调用单口LPM RAM

6.3 LPM_RAM宏模块用法

6.3.2 以原理图方式对LPM_RAM进行调用

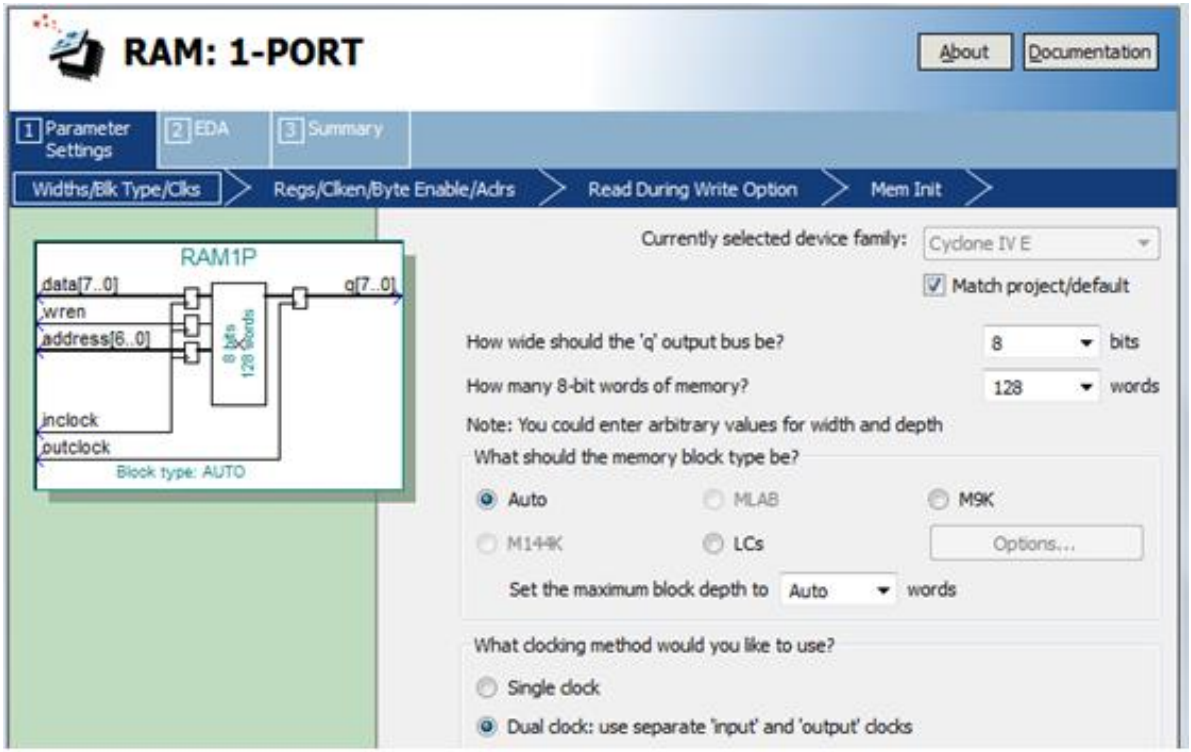

图6-14 设定RAM参数

6.3 LPM_RAM宏模块用法

6.3.2 以原理图方式对LPM_RAM进行调用

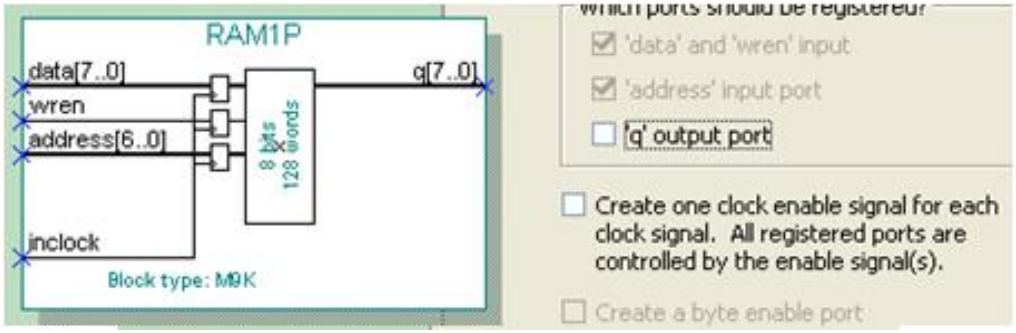

图6-15 设定RAM仅输入时钟控制

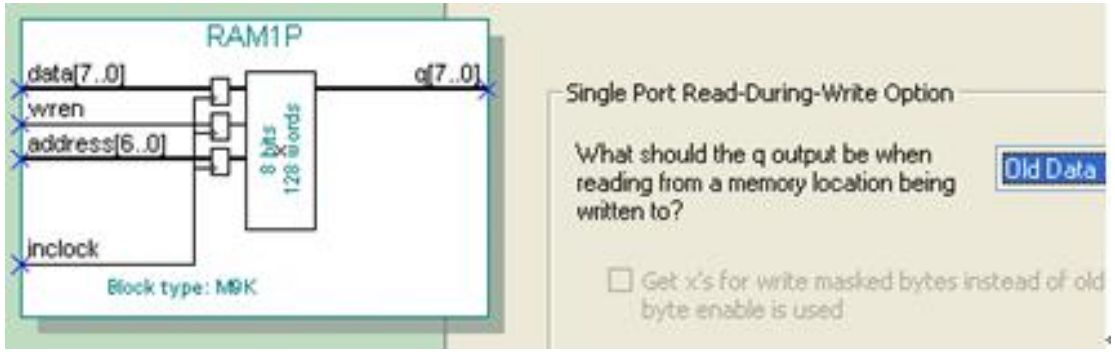

图6-16 设定在写入同时读出原数据：Old Data

6.3 LPM_RAM宏模块用法

6.3.2 以原理图方式对LPM_RAM进行调用

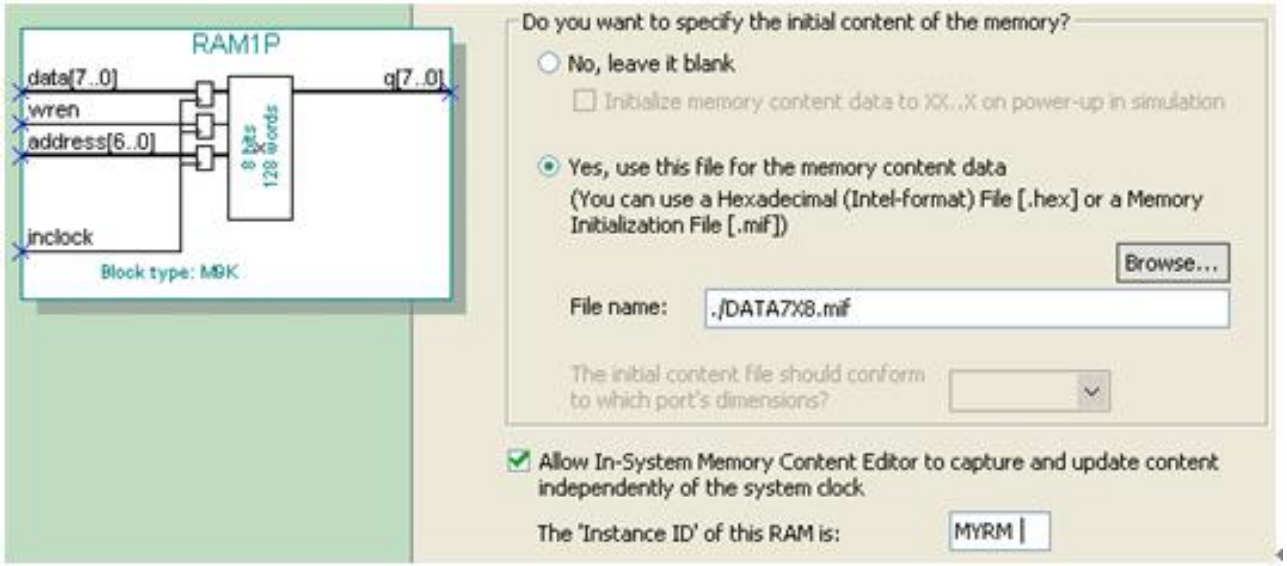

图6-17 设定初始化文件和允许在系统编辑

6.3 LPM_RAM宏模块用法

6.3.2 以原理图方式对LPM_RAM进行调用

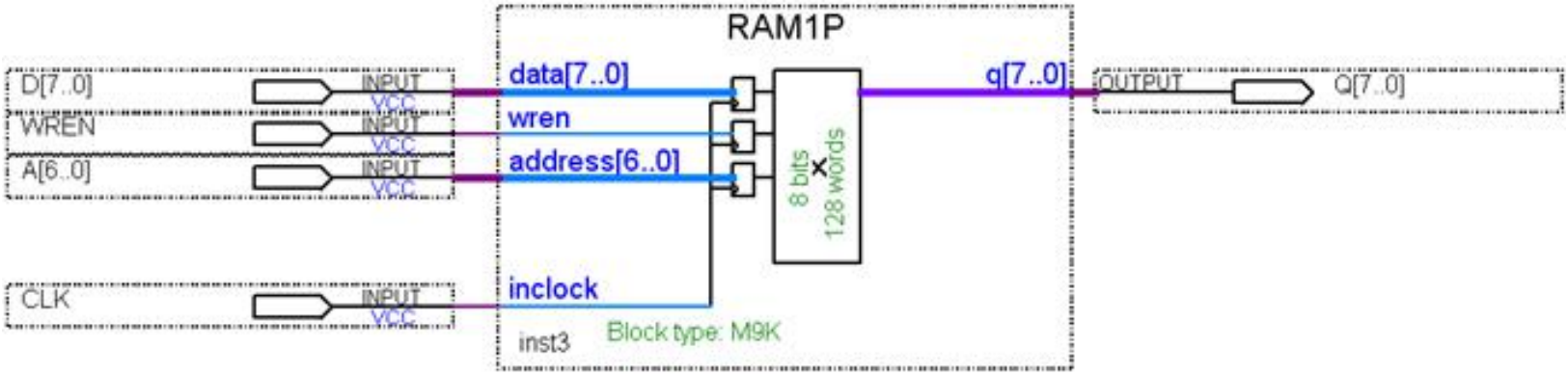

图6-18 在原理图上连接好的RAM模块

6.3 LPM_RAM宏模块用法

6.3.3 测试LPM_RAM

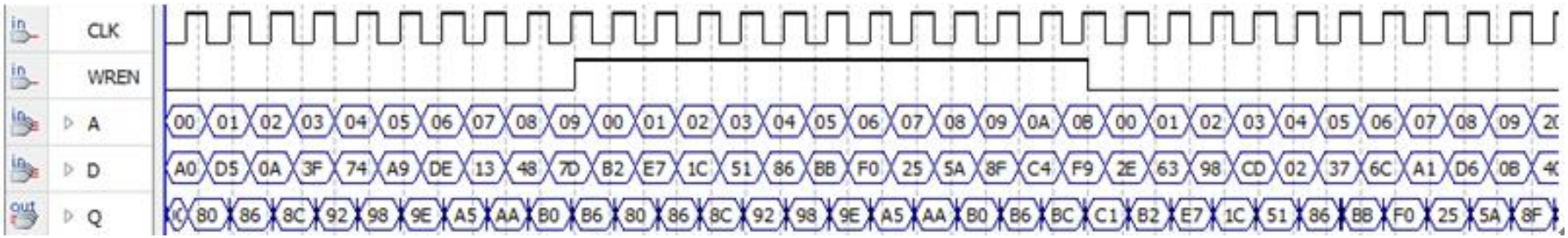

图6-19 图6-18的RAM仿真波形

6.3 LPM_RAM宏模块用法

6.3.4 Verilog代码描述的存储器初始化文件加载表述

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

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

【例 6-6】

```
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_RAM宏模块用法

6.3.5 存储器设计的结构控制

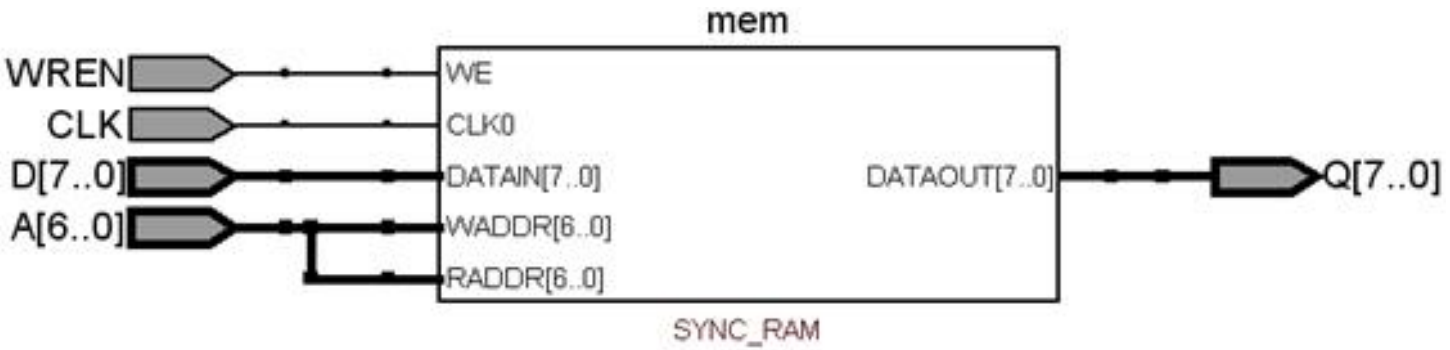

图6-20 例6-6的RTL电路模块图

6.3 LPM_RAM宏模块用法

6.3.5 存储器设计的结构控制

【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
```

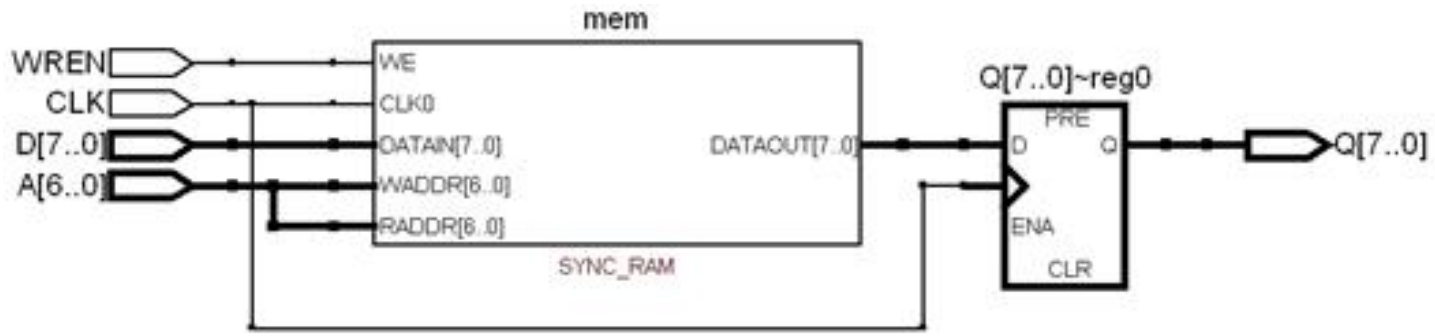

图6-21 例6-7的RTL电路模块图

6.4 LPM_ROM使用示例

6.4.1 简易正弦信号发生器设计

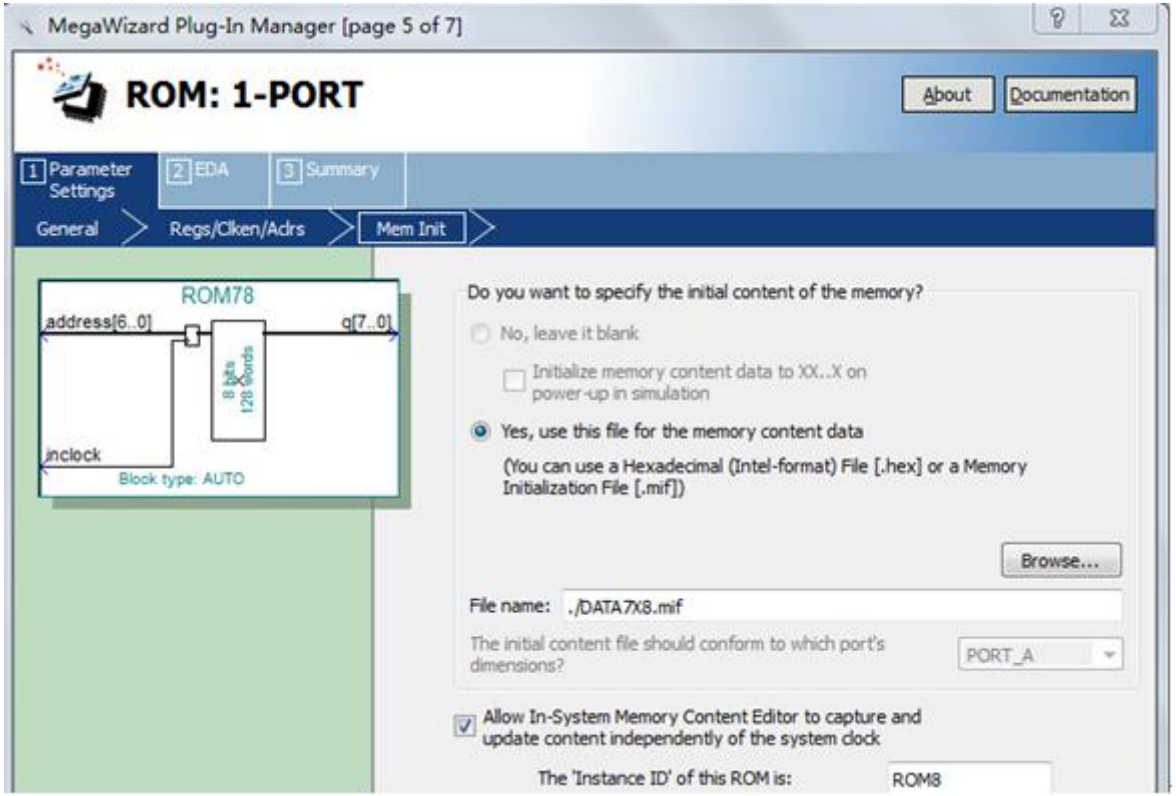

图6-22 加入初始化配置文件并允许在系统访问ROM内容

6.4 LPM_ROM使用示例

6.4.1 简易正弦信号发生器设计

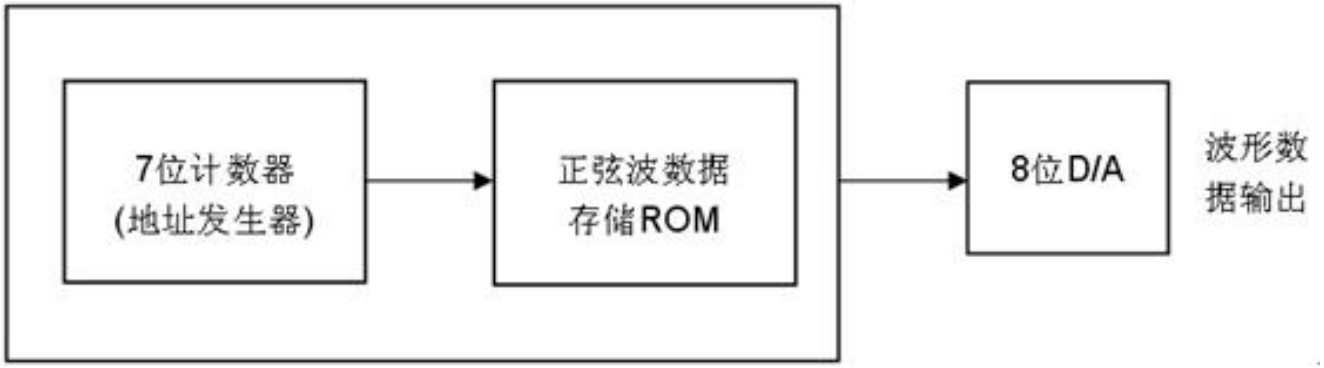

图6-23 正弦信号发生器结构框图

6.4 LPM_ROM使用示例

6.4.1 简易正弦信号发生器设计

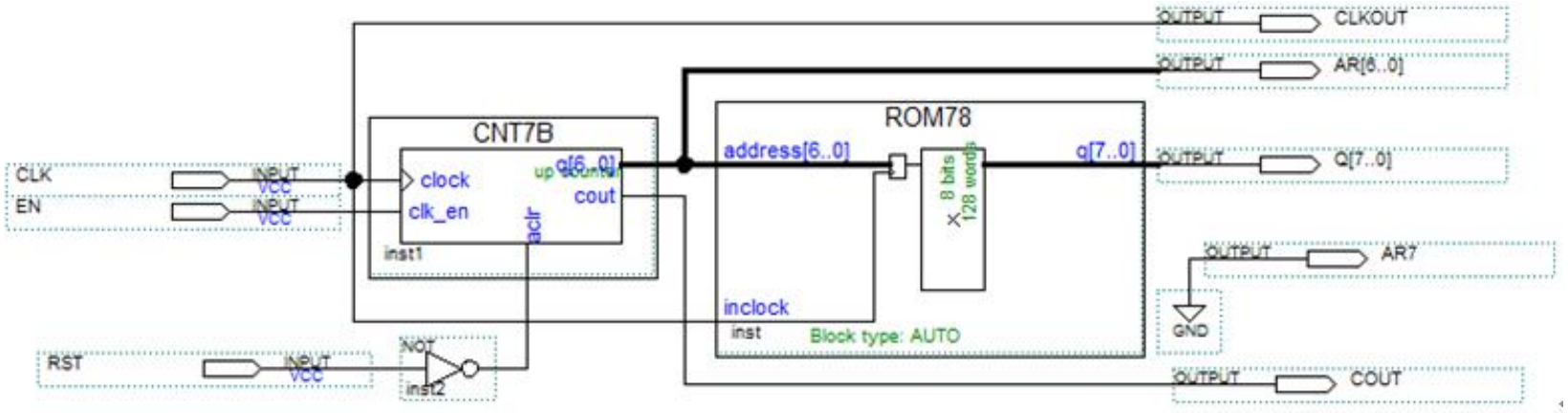

图6-24 正弦信号发生器电路原理图

6.4 LPM_ROM使用示例

6.4.1 简易正弦信号发生器设计

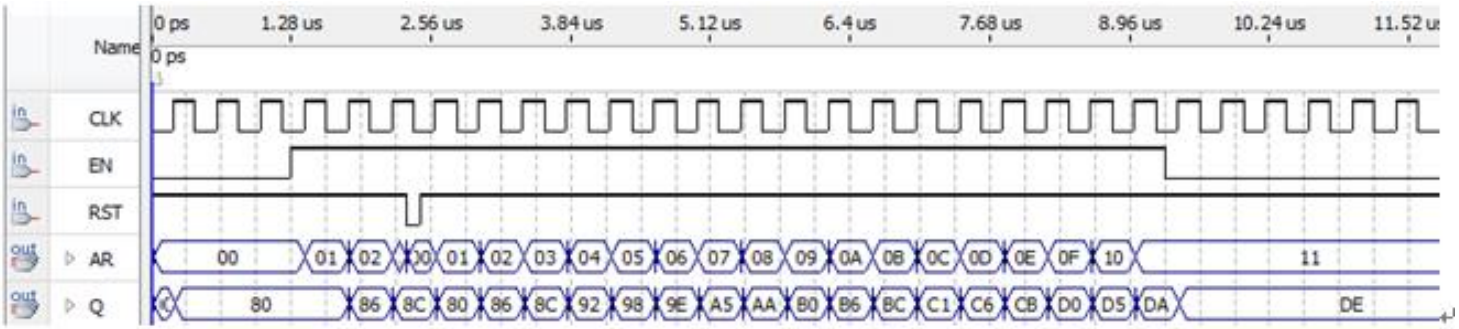

图6-25　图24电路仿真波形

6.4 LPM_ROM使用示例

6.4.2 正弦信号发生器硬件实现和测试

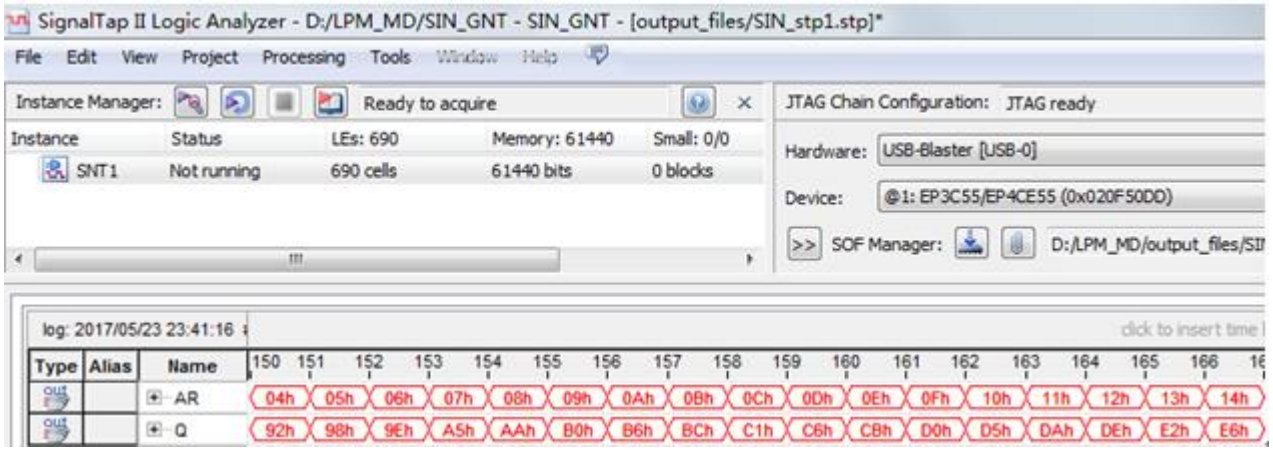

图6-26 正弦信号发生器数据输出的SignalTap II实时测试界面

6.4 LPM_ROM使用示例

6.4.2 正弦信号发生器硬件实现和测试

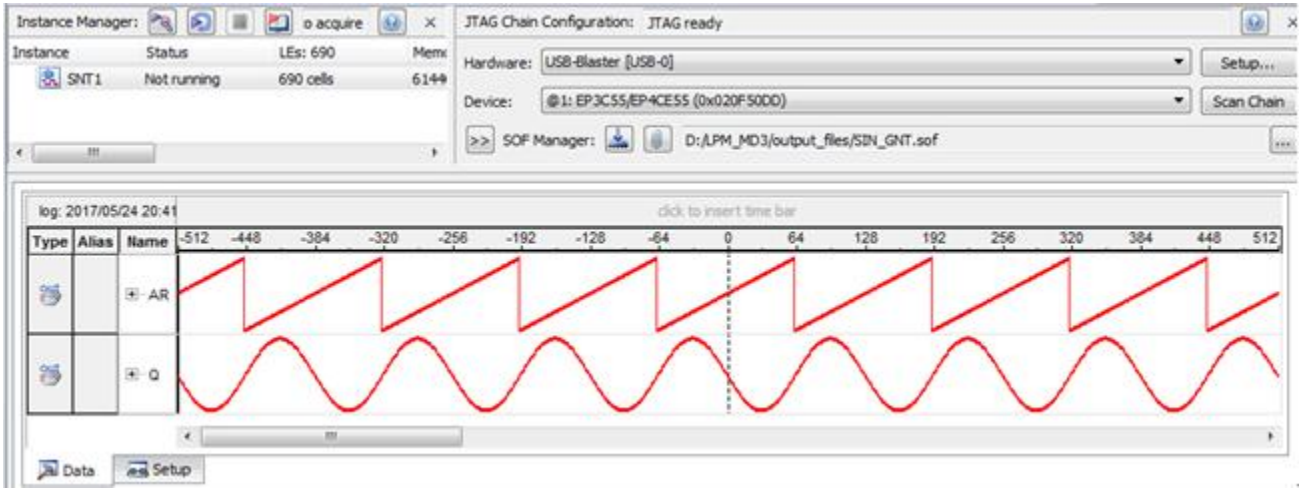

图6-27 正弦信号发生器的SignalTap II的波形显示图面

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。

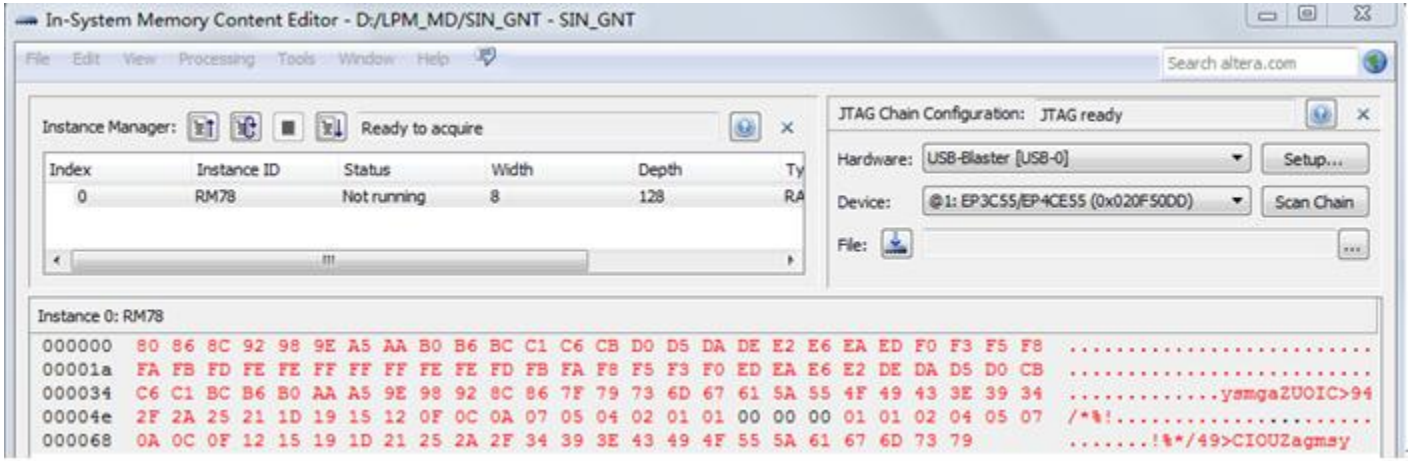

图6-28 In-System Memory Content Editor编辑窗，从FPGA中的ROM读取波形数据

（2）读取ROM中的数据。

6.5 在系统存储器数据读写编辑器应用

（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6-29 在此将编辑好的数据载入FPGA中的ROM内

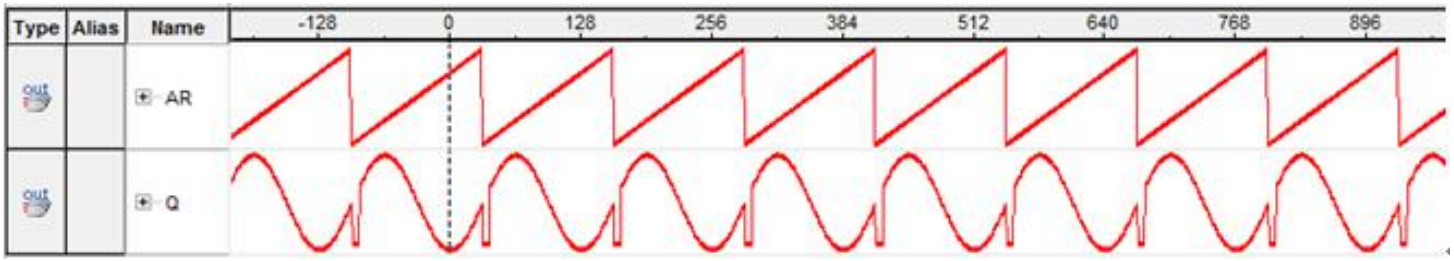

图6-30 SignalTap II测得的数据波形

（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

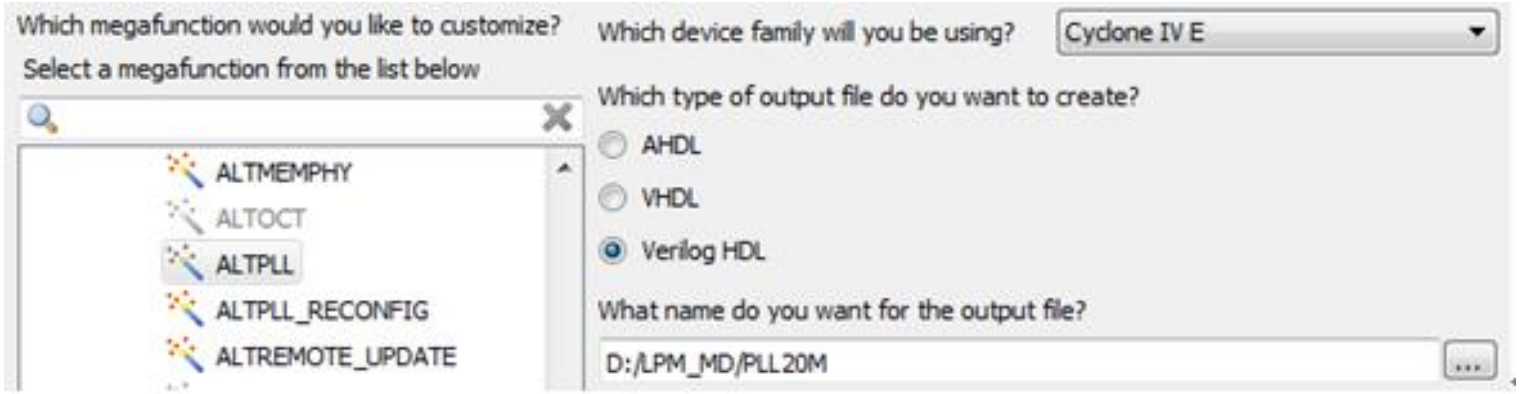

图6-31 选择锁相环ALTPLL

6.6 LPM嵌入式锁相环调用

6.6.1 建立嵌入式锁相环元件

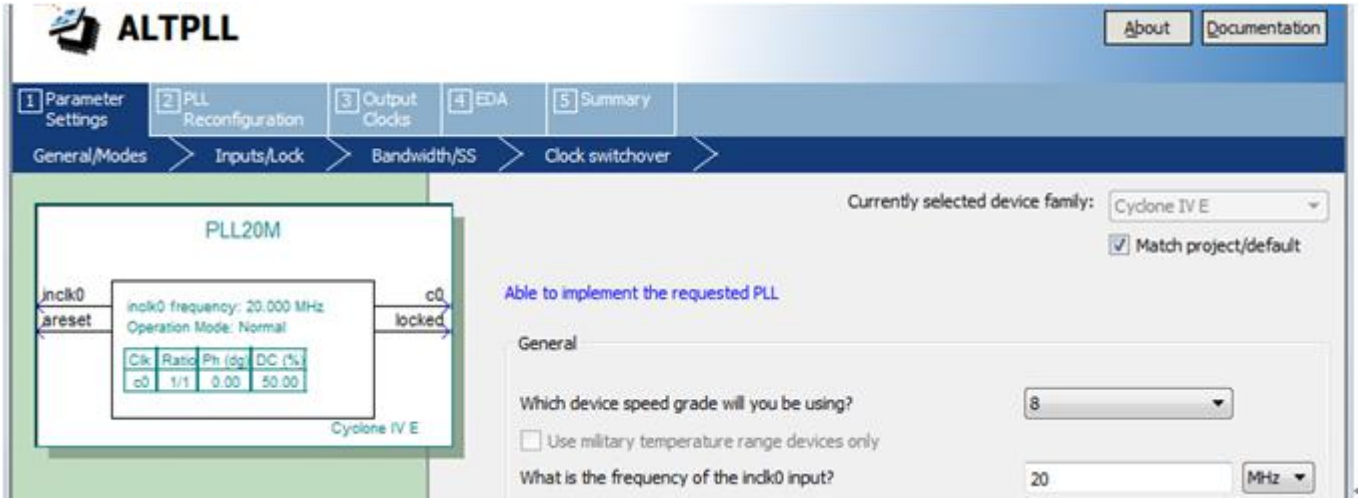

图6-32 选择输入参考时钟inclk0为20MHz

6.6 LPM嵌入式锁相环调用

6.6.1 建立嵌入式锁相环元件

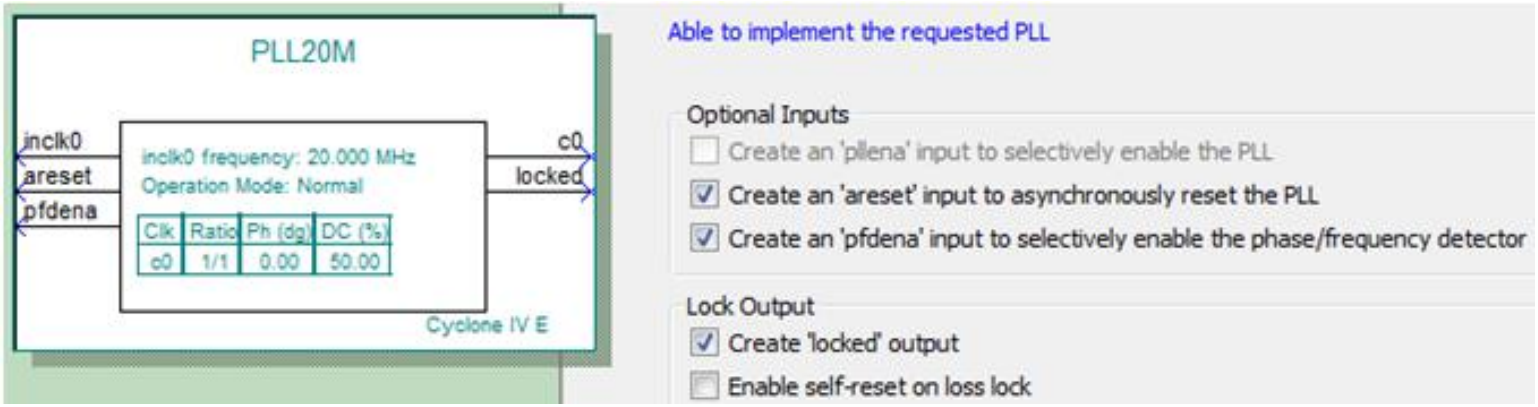

图 6-33 选择→锁相环的控制信号

6.6 LPM嵌入式锁相环调用

6.6.1 建立嵌入式锁相环元件

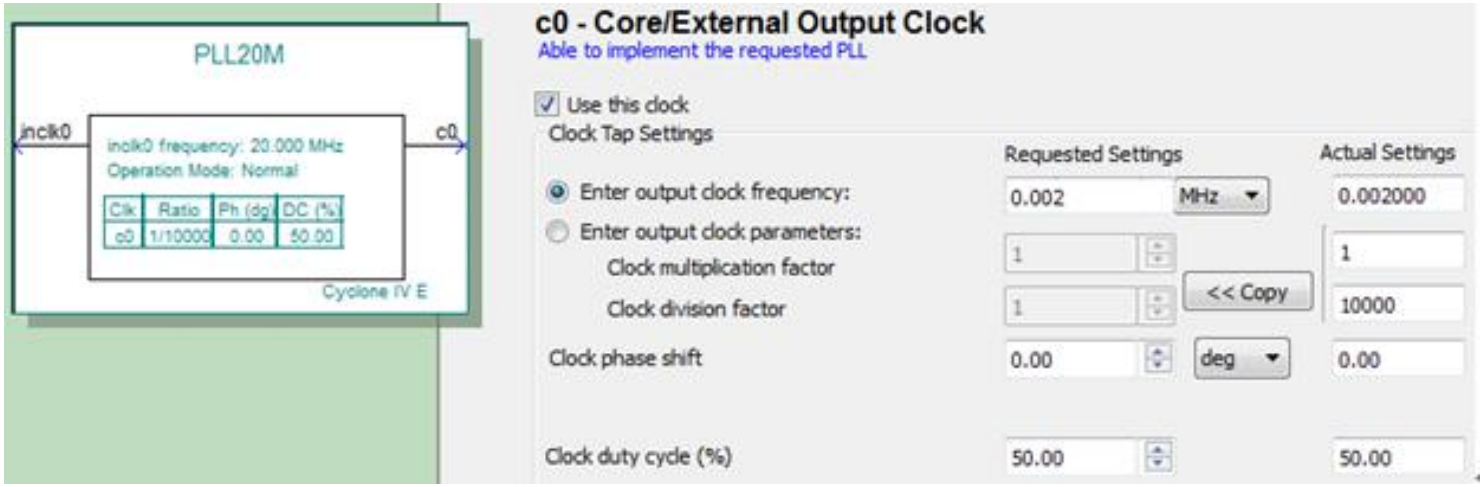

图6-34 选择c0的输出频率为0.002MHz

6.6 LPM嵌入式锁相环调用

6.6.1 建立嵌入式锁相环元件

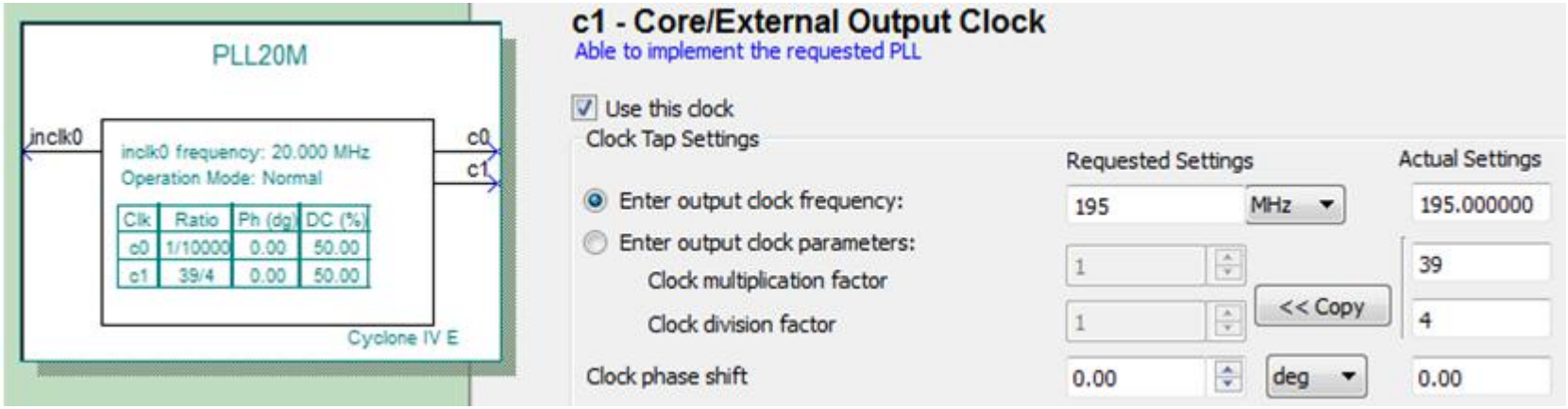

图6-35 输出第二个时钟信号c1

6.6 LPM嵌入式锁相环调用

6.6.1 建立嵌入式锁相环元件

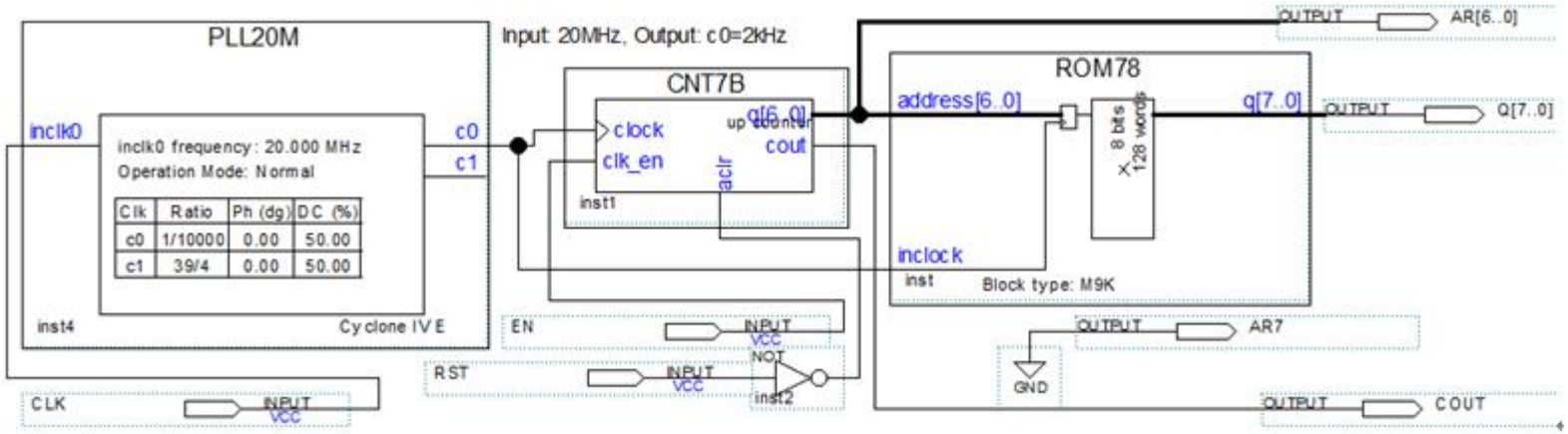

图6-36 采用了嵌入式锁相环作时钟的正弦信号发生器电路

6.6.2 测试锁相环

6.7 In-System Sources and Probes Editor用法

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

（2）设定参数。

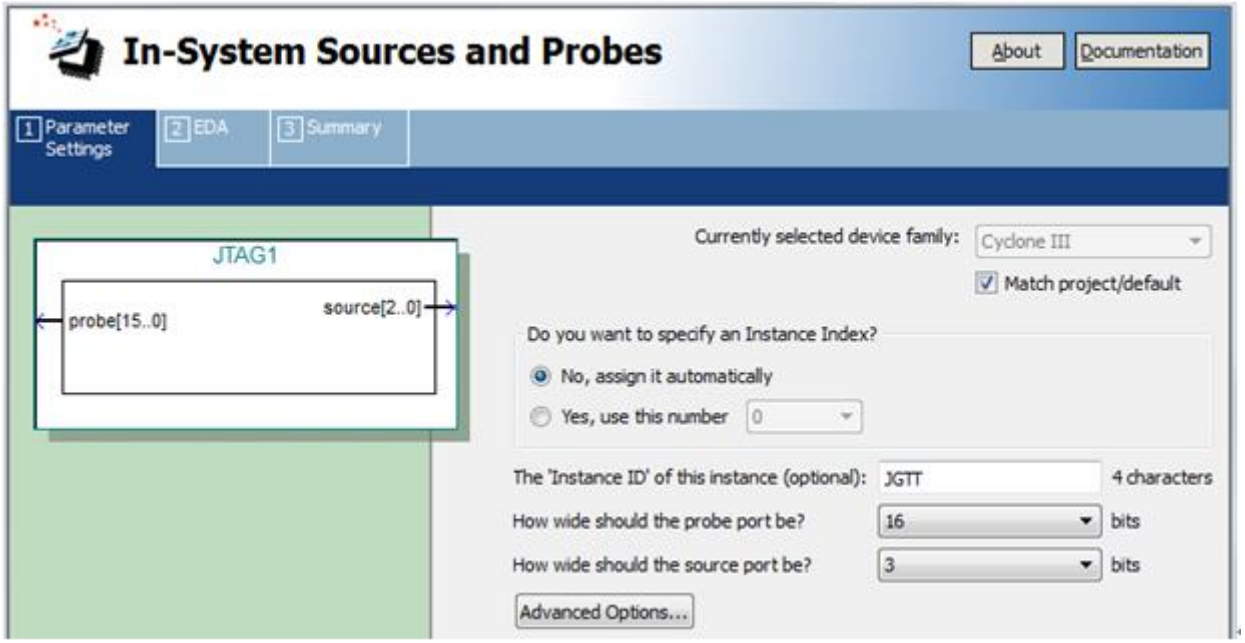

图6-37 为In-System Sources and Probes模块设置参数

6.7 In-System Sources and Probes Editor用法

（3）与需要测试的电路系统连接好。

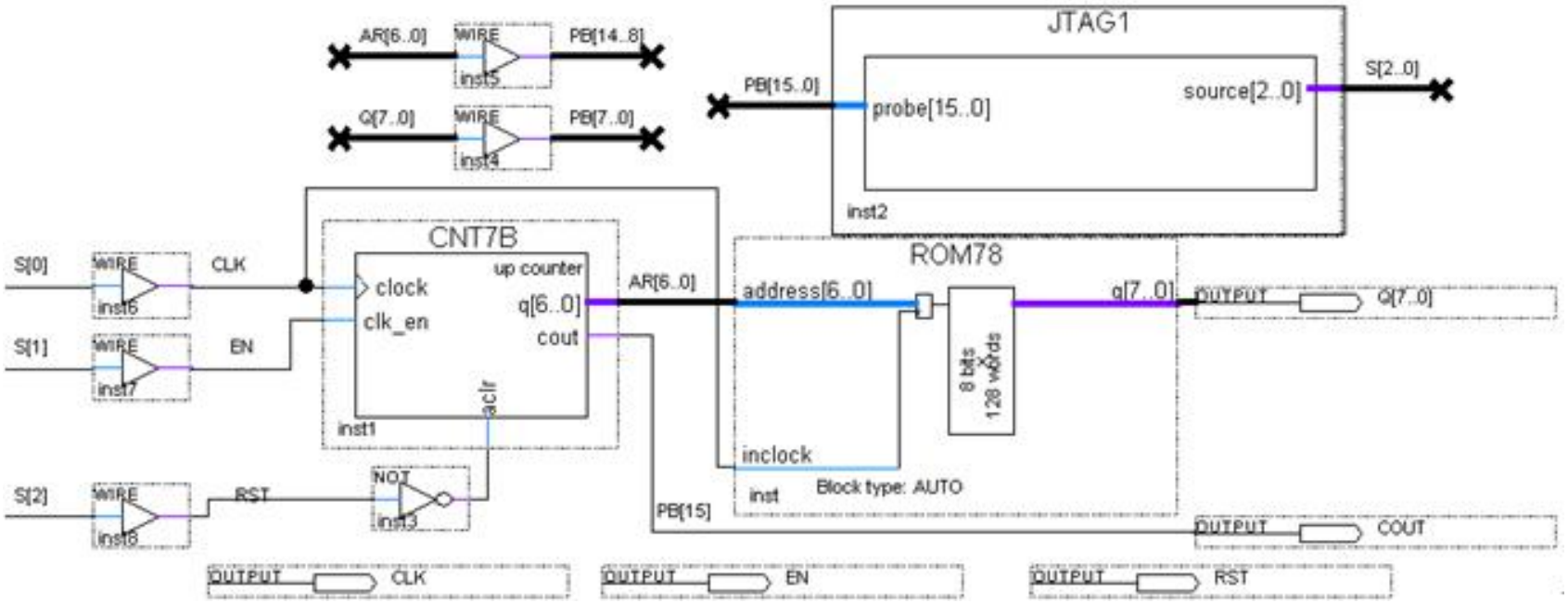

图6-38 在电路中加入In-System Sources and Probes测试模块

6.7 In-System Sources and Probes Editor用法

（4）调用In-System Sources and Probes Editor。

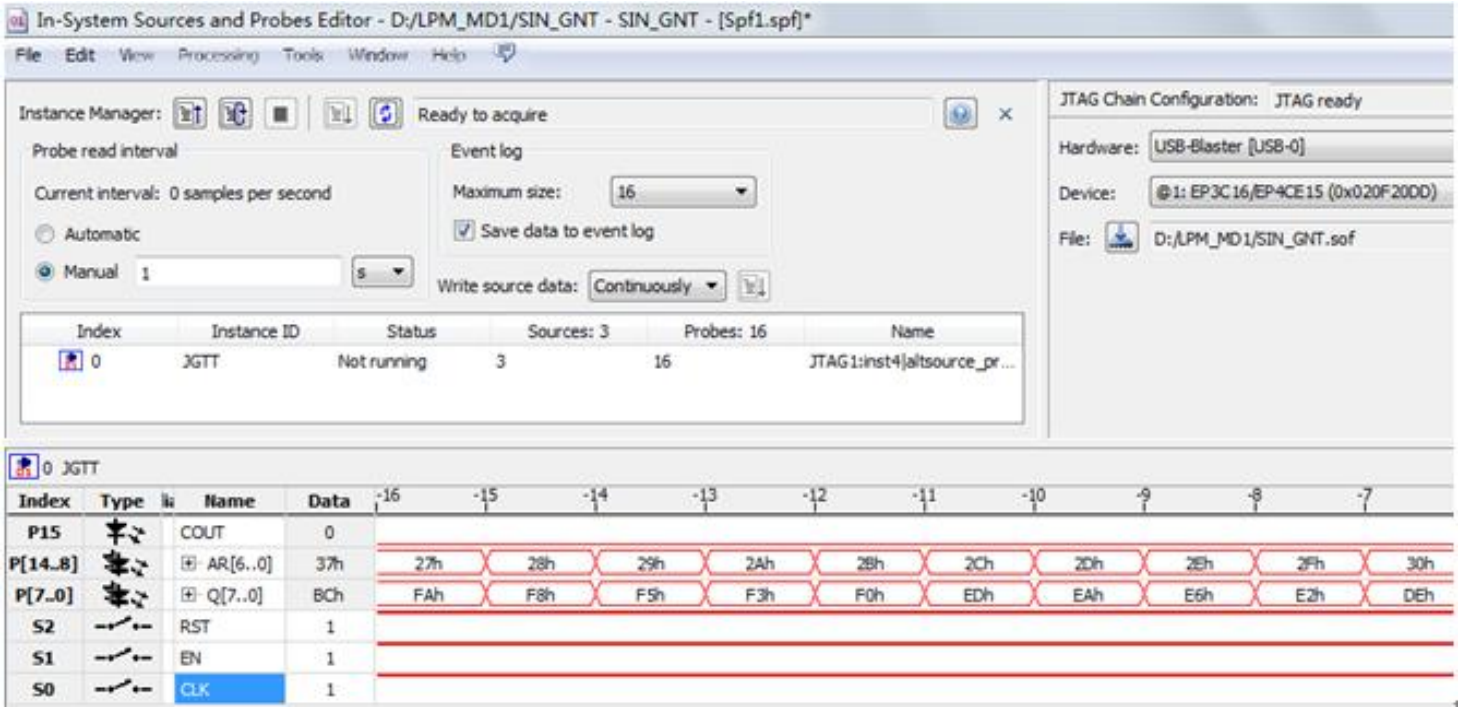

图6-39 In-System Sources and Probes Editor的测试情况

6.8 DDS实现原理与应用

6.8.1 DDS原理

$$S_{out} = A\sin\omega t = A\sin(2\pi f_{out} t) \qquad (6\text{-}1)$$

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

$$\Delta\theta = 2\pi f_{out} T_{clk} = \frac{2\pi f_{out}}{f_{clk}} \qquad (6\text{-}3)$$

$$\frac{B_{\Delta\theta}}{2^N} = \frac{f_{out}}{f_{clk}}, \quad B_{\Delta\theta} = 2^N \cdot \frac{f_{out}}{f_{clk}} \qquad (6\text{-}4)$$

6.8 DDS实现原理与应用

6.8.1 DDS原理

$$S_{out} = A\sin(\theta_{k-1} + \Delta\theta) = A\sin\left[\frac{2\pi}{2^N}\cdot\left(B_{\theta_{k-1}} + B_{\Delta\theta}\right)\right] = Af_{\sin}\left(B_{\theta_{k-1}} + B_{\Delta\theta}\right) \quad (6\text{-}5)$$

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

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

$$f_{out} = \frac{f_{clk}}{2^N} \quad (6\text{-}8)$$

6.8 DDS实现原理与应用

6.8.1 DDS原理

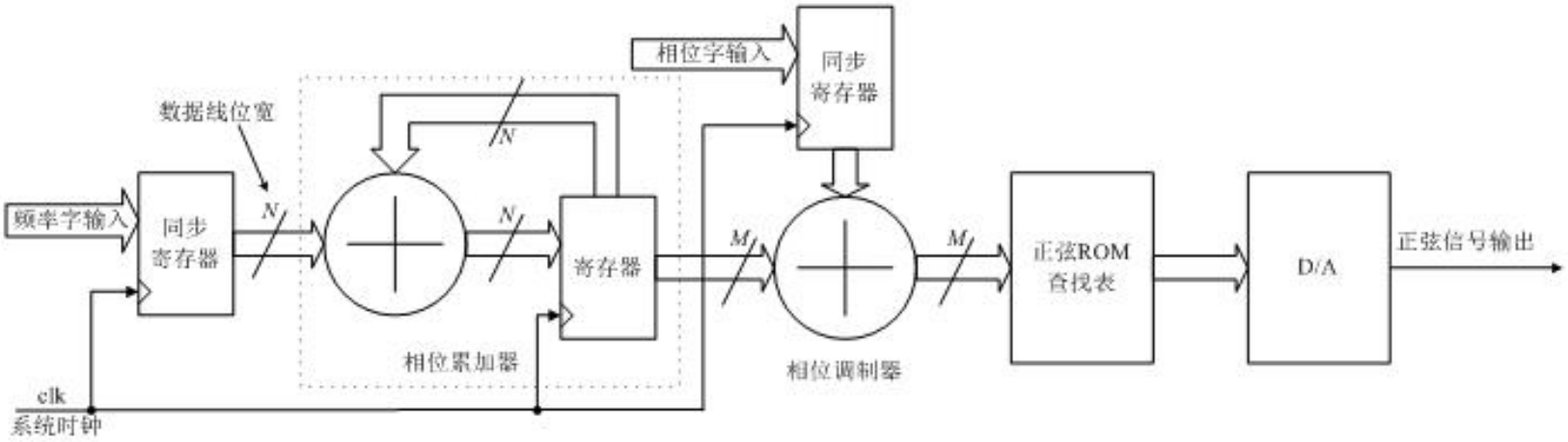

图6-40 基本DDS结构

6.8 DDS实现原理与应用

6.8.2 DDS信号发生器设计示例

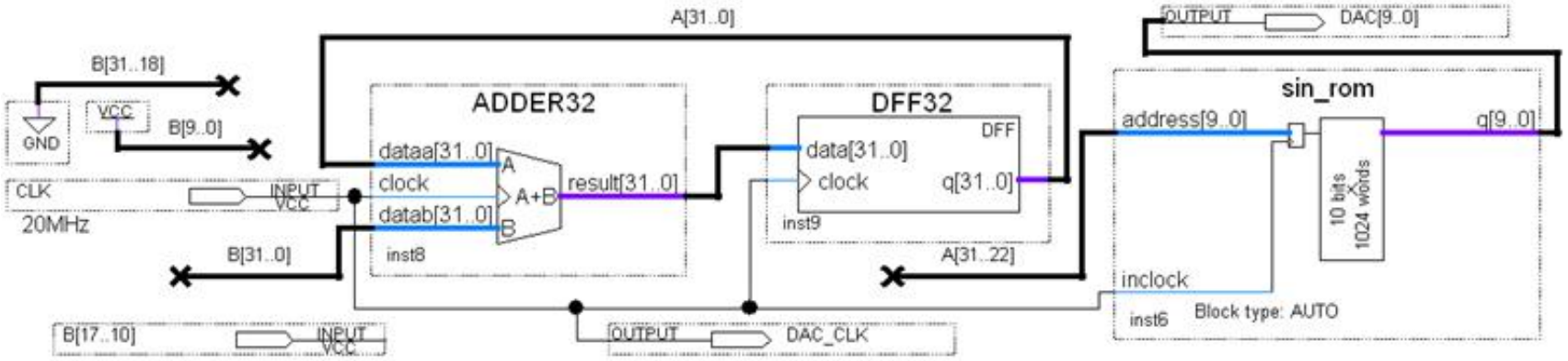

图6-41 DDS信号发生器电路顶层原理图

6.8 DDS实现原理与应用

6.8.2 DDS信号发生器设计示例

$$f_{out} = \frac{B[31..0]}{2^{32}} \cdot f_{clk} \qquad (6\text{-}9)$$

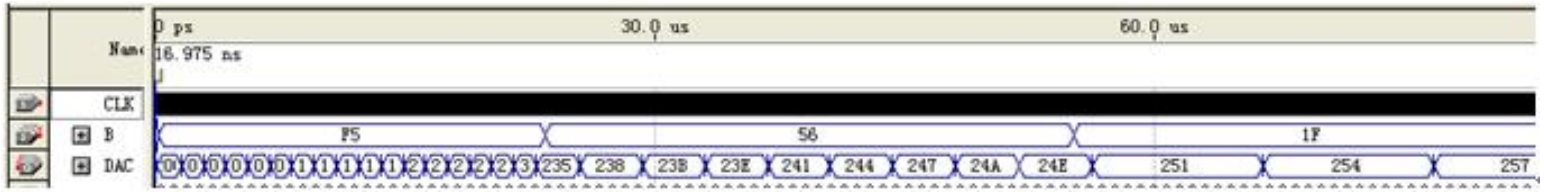

图6-42 图6-41的仿真波形

实验与设计

实验6-1 查表式硬件运算器设计

实验6-2 正弦信号发生器设计

实验6-3 简易逻辑分析仪设计

实验与设计

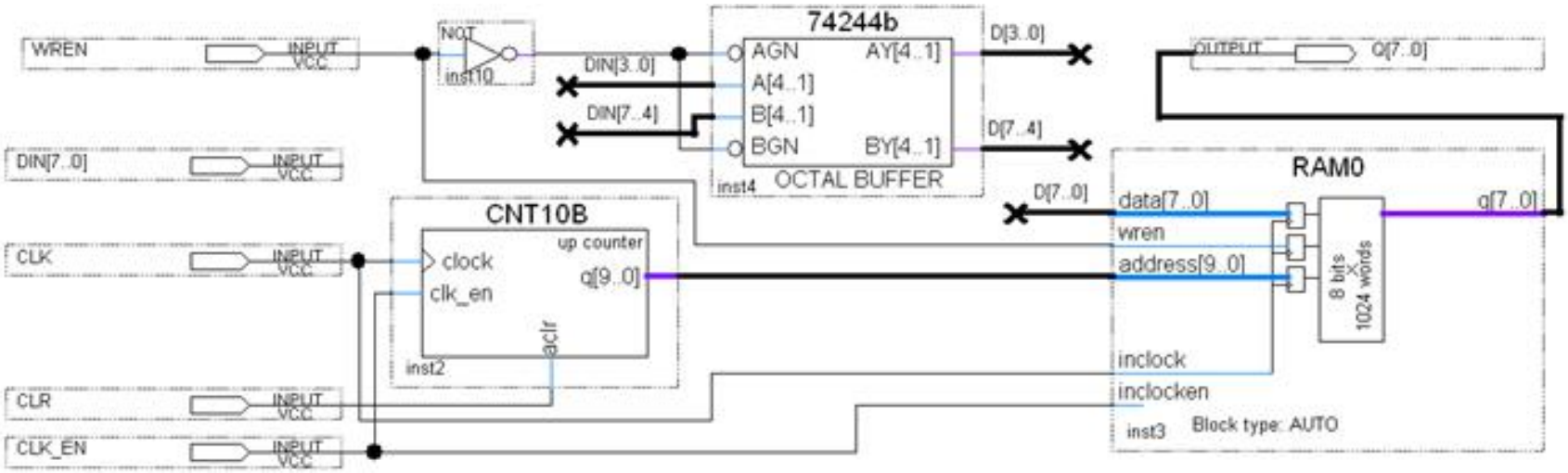

图6-43 逻辑数据采样电路顶层设计

实验与设计

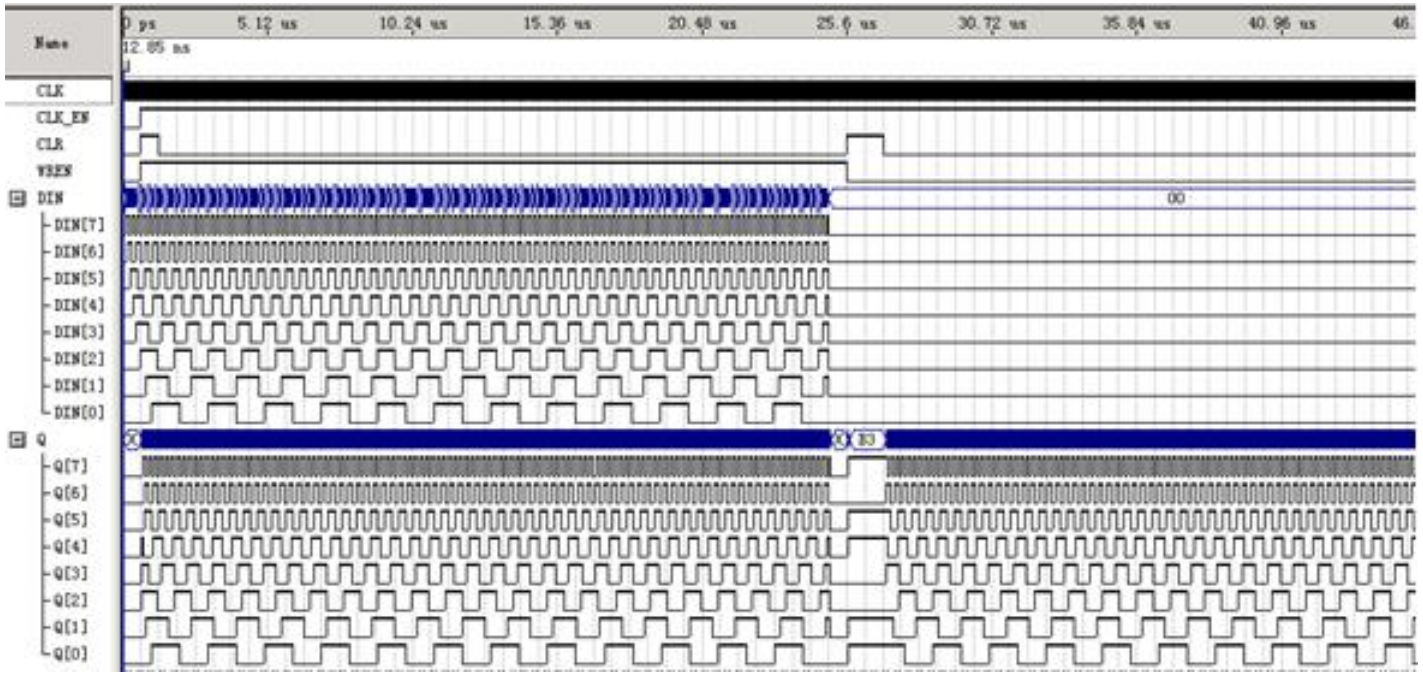

图6-44 逻辑数据采样电路时序仿真波形

实验与设计

实验6-4 DDS正弦信号发生器设计

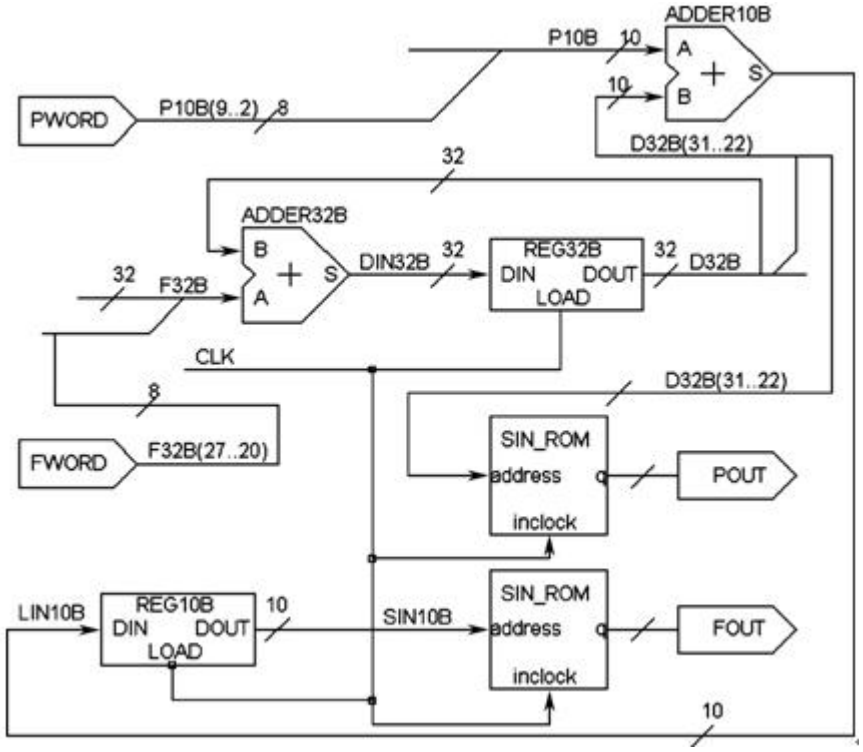

图6-45 DDS正弦信号发生器顶层原理图

实验与设计

实验6-5 移相信号发生器设计

实验6-6 AM幅度调制信号发生器设计

实验6-7 硬件消抖动电路设计

实验与设计

【例 6-8】

```
module ERZP  (CLK, KIN,KOUT);
   input   CLK, KIN;                          //工作时钟和输入信号
   output  KOUT;     reg KOUT;
   reg [3:0] KH,KL;                           //定义对高电平和低电平脉宽计数之寄存器。
   always @(posedge CLK)  begin
      if (!KIN)  KL<=KL+1 ;                   //对键输入的低电平脉宽计数
        else  KL<=4'b0000;    end             //若出现高电平，则计数器清 0
   always @(posedge CLK)  begin
      if (KIN)  KH<= KH+1;                    //同时对键输入的高电平脉宽计数
        else KH<=4'b0000;   end               //若出现高电平，则计数器清 0
   always @(posedge CLK)  begin
      if (KH > 4'b1100) KOUT<=1'B1;//对高电平脉宽计数一旦大于 12，则输出 1
        else if (KL > 4'b0111)
    KOUT<=1'B0;                               //对低电平脉宽计数若大于 7，则输出 0
    end
endmodule
```

实验与设计

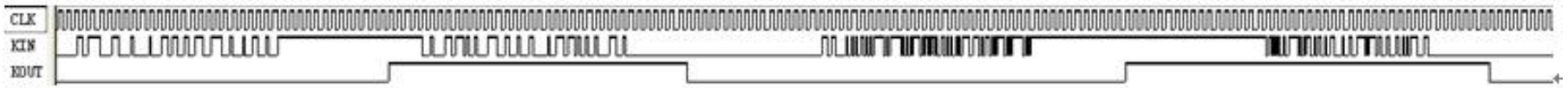

图6-46 例6-8消抖动电路仿真波形