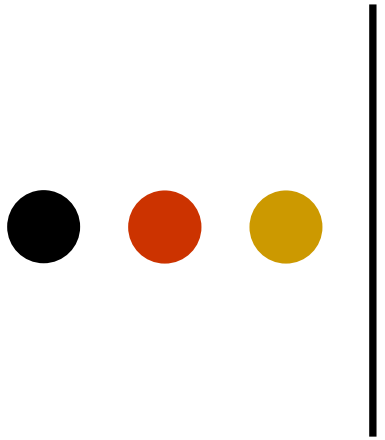


第5章

宏功能模块使用方法

5.1 基于LPM模块的计数器设计

5.1.1 计数器LPM模块文本文件的调用

（1）打开宏功能块调用管理器。

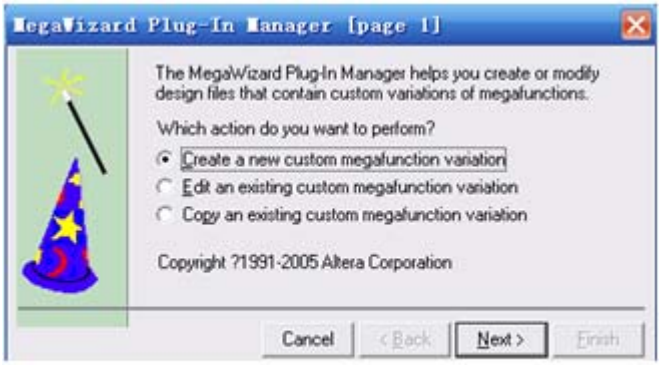


图 5-1　定制新的宏功能块

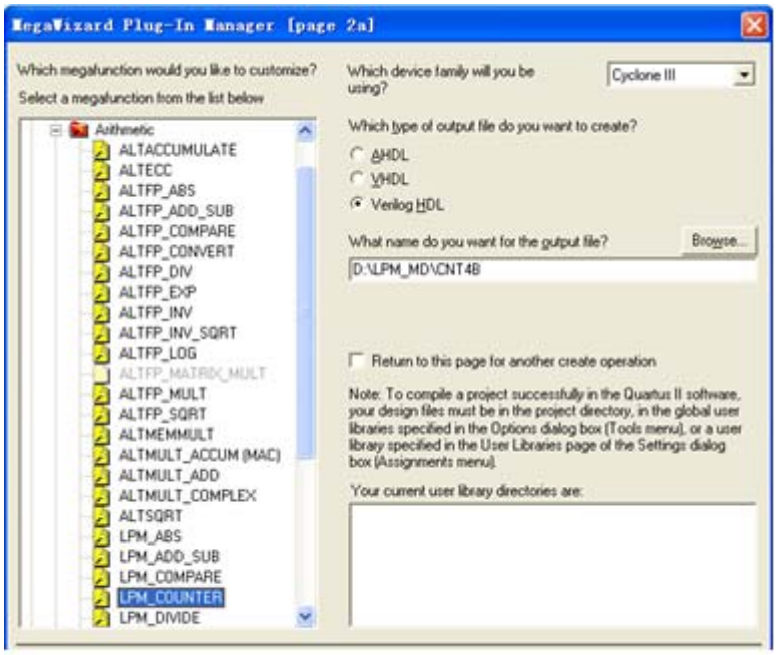


图 5-2　LPM 宏功能块设定

5.1 基于LPM模块的计数器设计

5.1.1 计数器LPM模块文本文件的调用

（2）单击Next按钮后打开如图5-3所示的对话框。

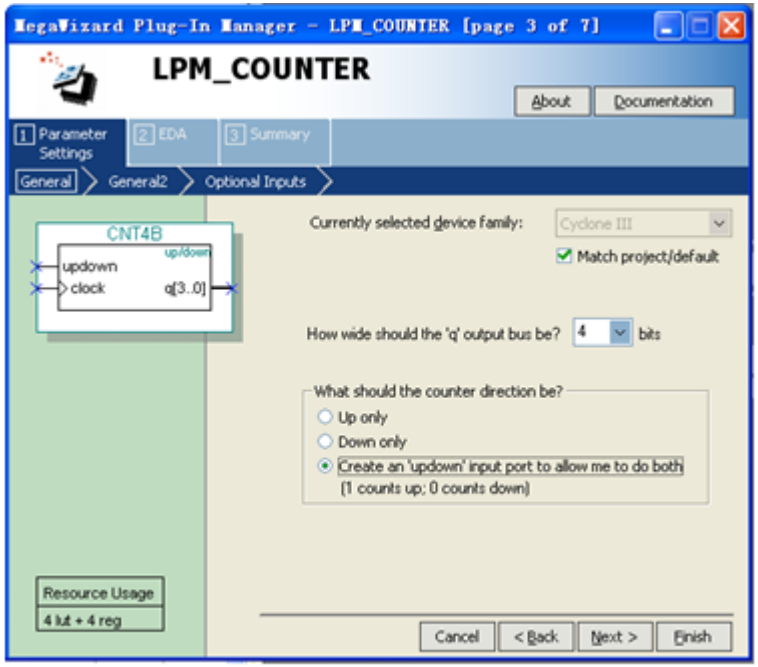


图 5-3 设 4 位可加减计数器

5.1 基于LPM模块的计数器设计

5.1.1 计数器LPM模块文本文件的调用

（3）再单击Next按钮，打开如图5-4所示的对话框。

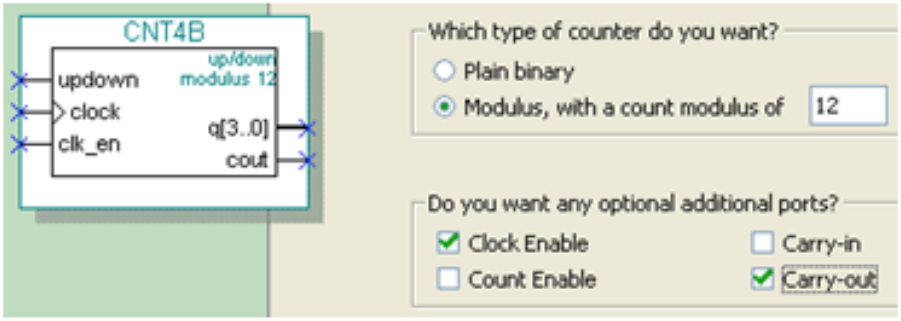


图 5-4 设定计数器，含时钟使能和进位输出

5.1 基于LPM模块的计数器设计

5.1.1 计数器LPM模块文本文件的调用

（4）再单击**Next**按钮，打开如图**5-5**所示的对话框。

图 5-5 加入 4 位并行数据预置功能

5.1.2 LPM计数器程序与参数传递语句

【例 5-1】 Quartus II 生成的计数器文件 CNT4B.v

```
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                                                  //参数传递说明语句
   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
```

5.1 基于LPM模块的计数器设计

5.1.2 LPM计数器程序与参数传递语句

defparam <宏模块元件例化名>.<宏模块参数名> = <参数值>

【例 5-2】

```
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 5-3】

```
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
```

5.1 基于LPM模块的计数器设计

5.1.3 创建工程与仿真测试

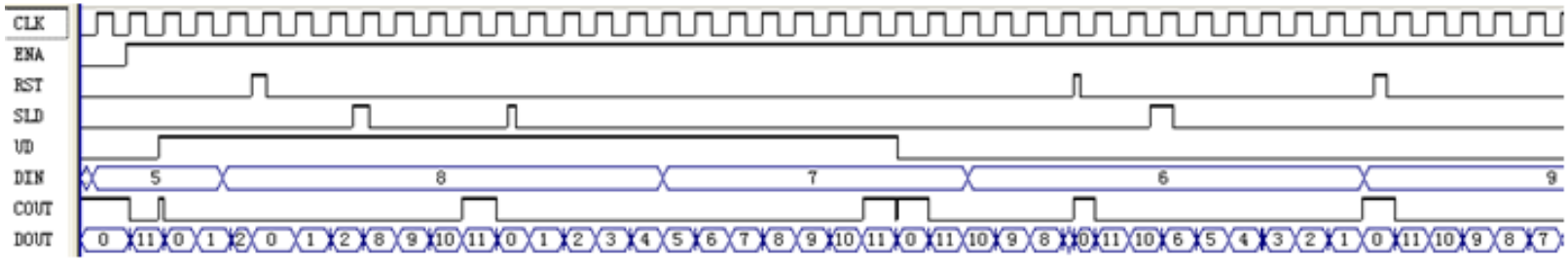


图 5-6 CNT4BIT.v 的仿真波形

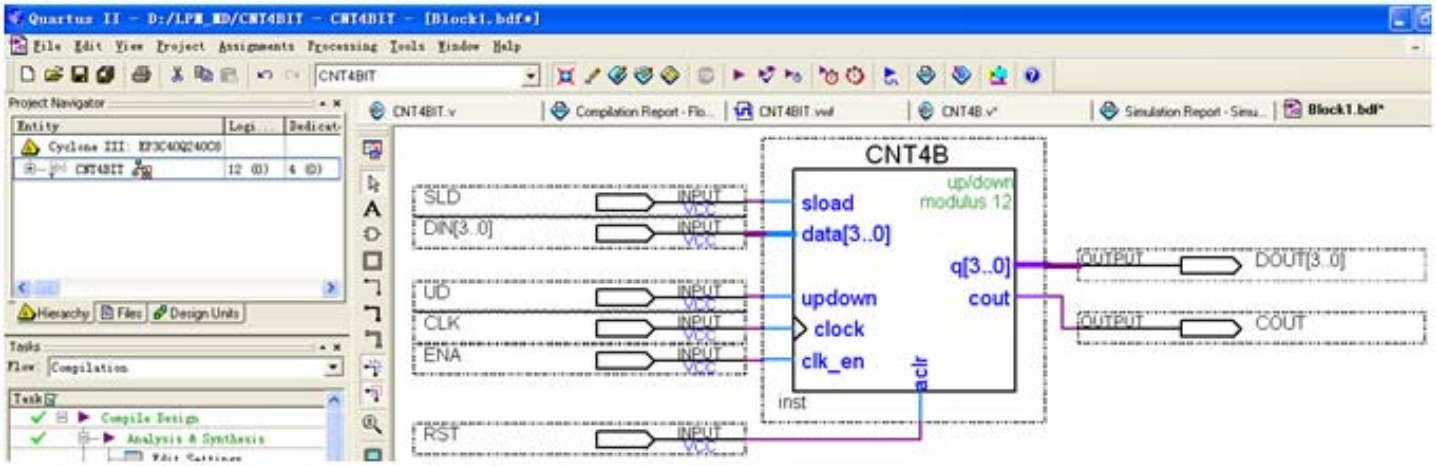


图 5-7　原理图输入设计

5.2 流水线乘法累加器设计

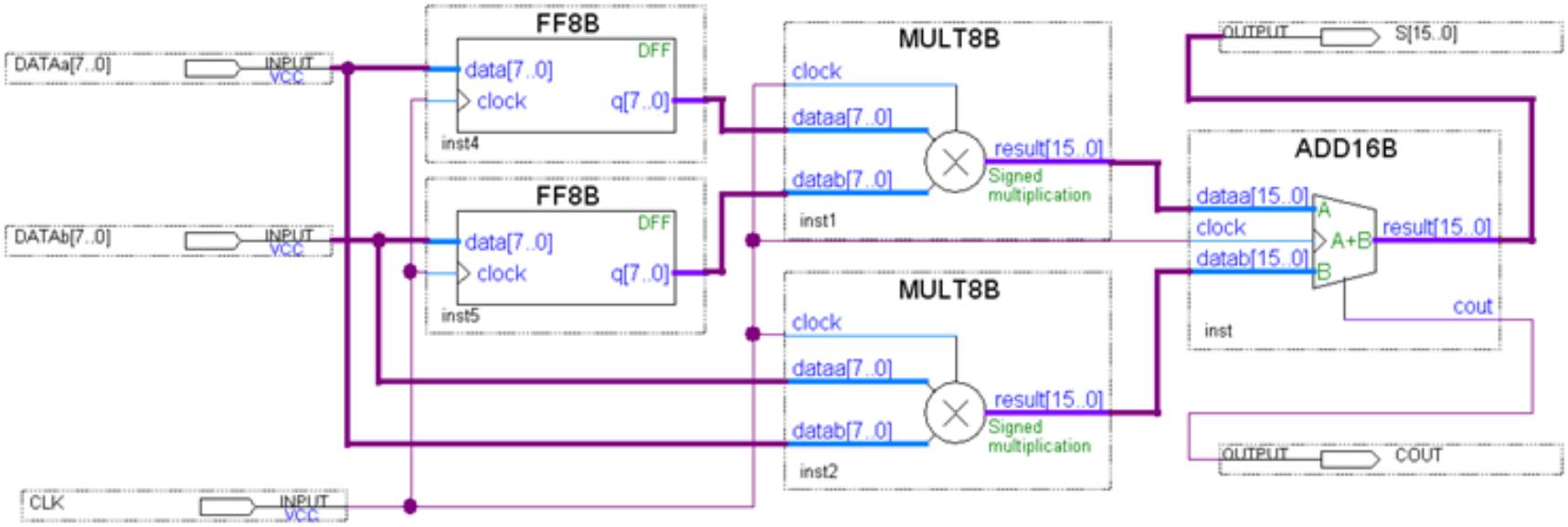


图 5-8　8 位乘法累加器顶层设计

5.2 流水线乘法累加器设计

5.2.1 LPM加法器模块设置

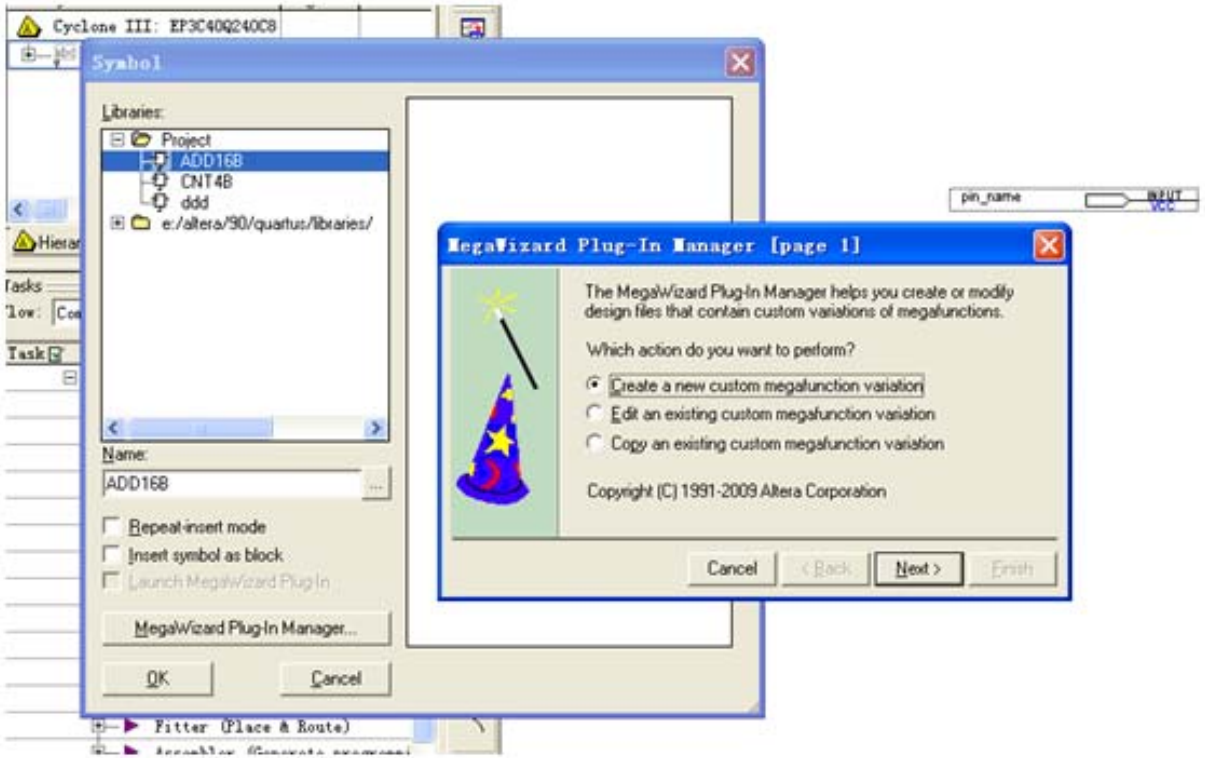


图 5-9　从原理图编辑窗进入 MegaWizard Plug-In Manager 管理器

5.2 流水线乘法累加器设计

5.2.1 LPM加法器模块设置

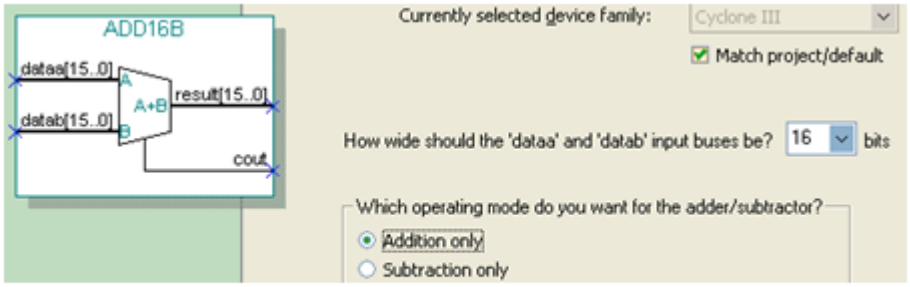


图 5-10 选择 16 位加法工作方式

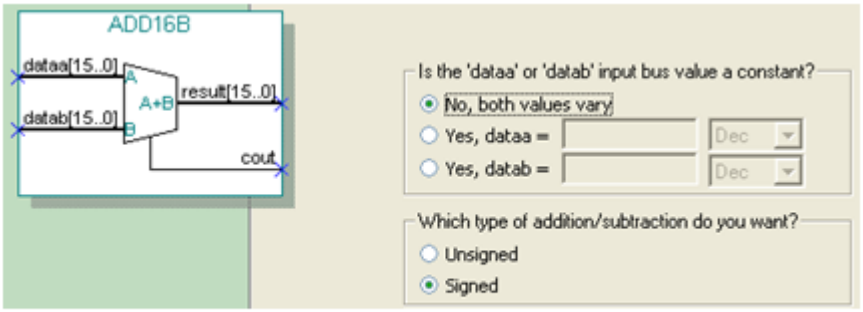


图 5-11 选择有符号加法操作类型输入

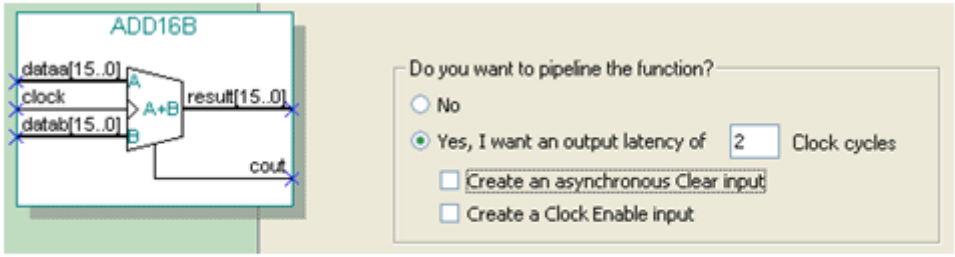


图 5-12 选择流水线方式

5.2 流水线乘法累加器设计

5.2.2 LPM乘法器模块设置

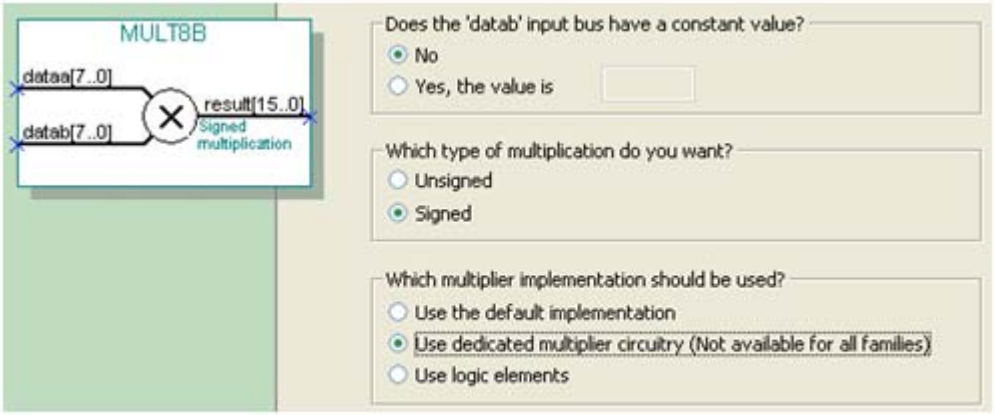


图 5-13　选择有符号乘法模式，并用专用乘法器模块构建乘法器

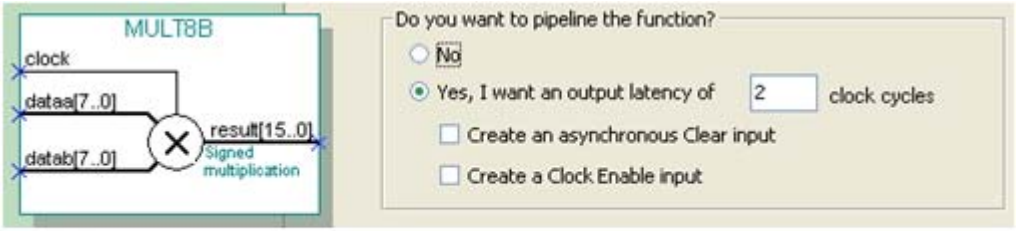


图 5-14　选择 2 级流水线乘法模式

5.2 流水线乘法累加器设计

5.2.3 仿真乘法累加器

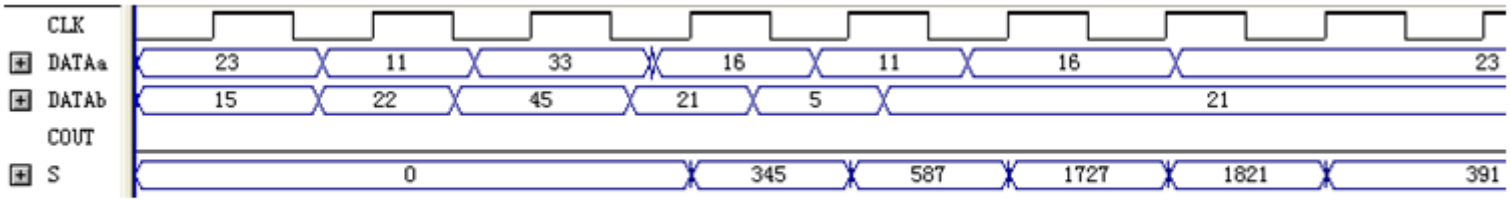


图 5-15　电路图图 5-8 的 MULTADD 工程仿真波形

Device	EP3C40Q240C8
Timing Models	Final
Met timing requirements	N/A
Total logic elements	50 / 39,600 (< 1 %)
Total combinational functions	17 / 39,600 (< 1 %)
Dedicated logic registers	50 / 39,600 (< 1 %)
Total registers	50
Total pins	34 / 129 (26 %)
Total virtual pins	0
Total memory bits	0 / 1,161,216 (0 %)
Embedded Multiplier 9-bit elements	2 / 252 (< 1 %)
Total PLLs	0 / 4 (0 %)

Device	EP3C40Q240C8
Timing Models	Final
Met timing requirements	N/A
Total logic elements	238 / 39,600 (< 1 %)
Total combinational functions	212 / 39,600 (< 1 %)
Dedicated logic registers	188 / 39,600 (< 1 %)
Total registers	188
Total pins	34 / 129 (26 %)
Total virtual pins	0
Total memory bits	0 / 1,161,216 (0 %)
Embedded Multiplier 9-bit elements	0 / 252 (0 %)
Total PLLs	0 / 4 (0 %)

图 5-16　对乘法器的构建模式选择不同设置后的编译报告

5.2 流水线乘法累加器设计

5.2.3 仿真乘法累加器

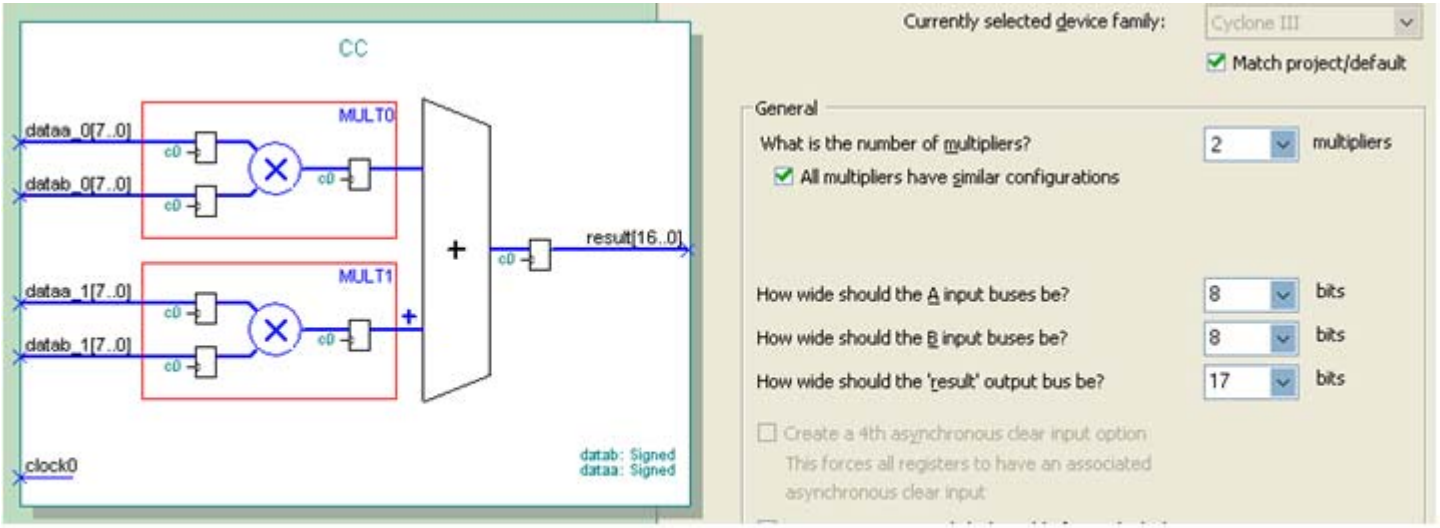


图 5-17　ALTMULT_ADD 模块设置对话框

5.2 流水线乘法累加器设计

5.2.4 乘法器的Verilog文本表述和相关属性设置

【例 5-4】

```
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" */;
```

5.2 流水线乘法累加器设计

5.2.4 乘法器的Verilog文本表述和相关属性设置

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	64 / 95 (67 %)
Total virtual pins	0
Total memory bits	0 / 423,936 (0 %)
Embedded Multiplier 9-bit elements	2 / 46 (4 %)
Total PLLs	0 / 2 (0 %)

图 5-18 例 5-4 的编译报告

5.2 流水线乘法累加器设计

5.2.4 乘法器的Verilog文本表述和相关属性设置

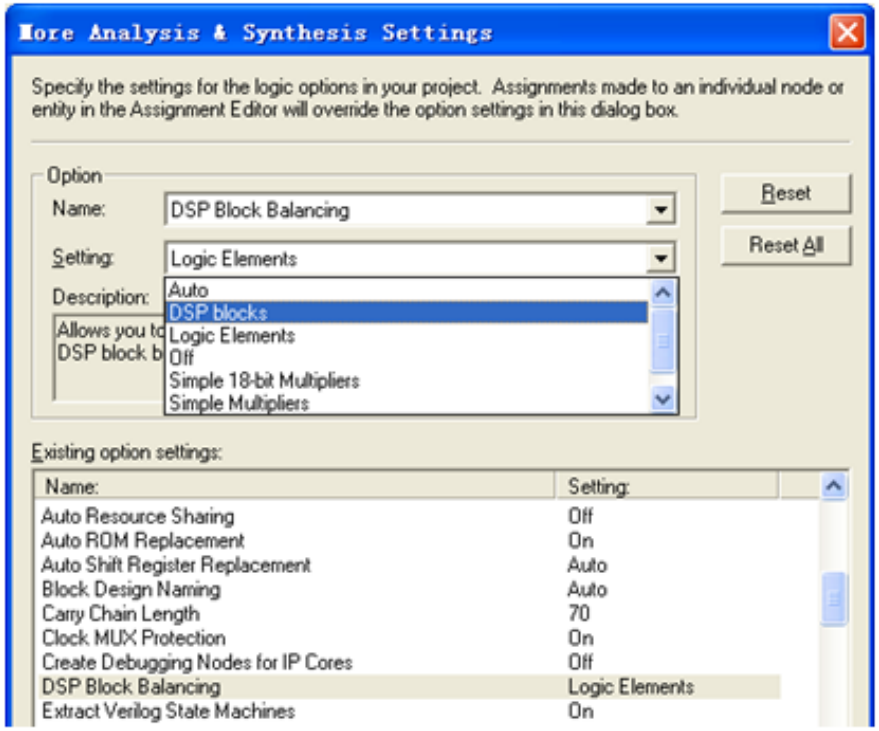


图 5-19　设置乘法器用 DSP 模块构建

5.3 LPM_RAM模块的设置

5.3.1 初始化文件生成

1. 建立.mif格式文件

（1）直接编辑法。

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 5-20 mif 文件编辑窗

5.3 LPM_RAM模块的设置

（2）文件编辑法。

【例 5-5】

```
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;
```

5.3 LPM_RAM模块的设置

（3）用C等软件生成。

（4）专用mif文件生成器。

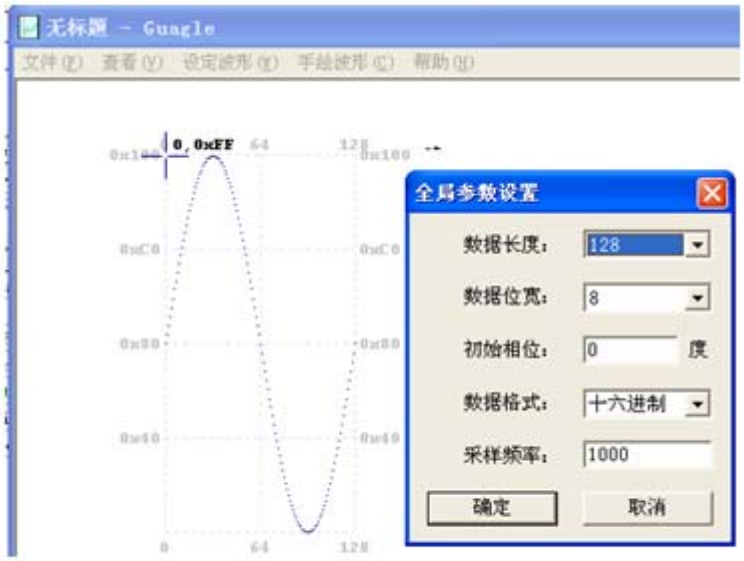


图 5-21 利用 mif 生成器生成正弦波数据文件

图 5-22 打开 mif 文件

5.3 LPM_RAM模块的设置

2．建立.hex格式文件

```
ORG 0000H
DB  255 , 254, 252 , 249
DB  245 , 239 , 233 , 225
DB  217 , 207 , 197 , 186
DB  174 , 162 , 150 , 137
DB  124 , 112 ,  99 ,  87
DB   75 ,  64 ,  53 ,  43
DB   34 ,  26 ,  19 ,  13
DB    8 ,   4 ,   1 ,   0
DB    0 ,   1 ,   4 ,   8
DB   13 ,  19 ,  26 ,  34
DB   43 ,  53 ,  64 ,  75
DB   87 ,  99 , 112 , 124
DB  137 , 150 , 162 , 174
DB  186 , 197 , 207 , 217
DB  225 , 233 , 239 , 245
DB  249 , 252 , 254 , 255
END
```

图 5-23 用汇编器生成

5.3 LPM_RAM模块的设置

5.3.2 LPM_RAM设置和调用

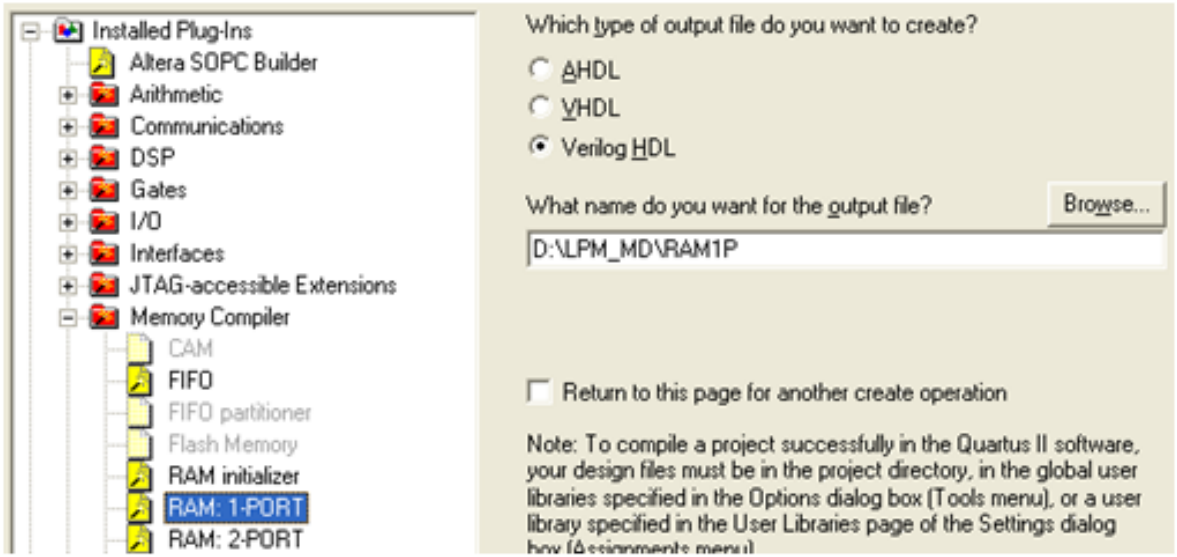


图 5-24　调用单口 LPM RAM

5.3 LPM_RAM模块的设置

5.3.2 LPM_RAM设置和调用

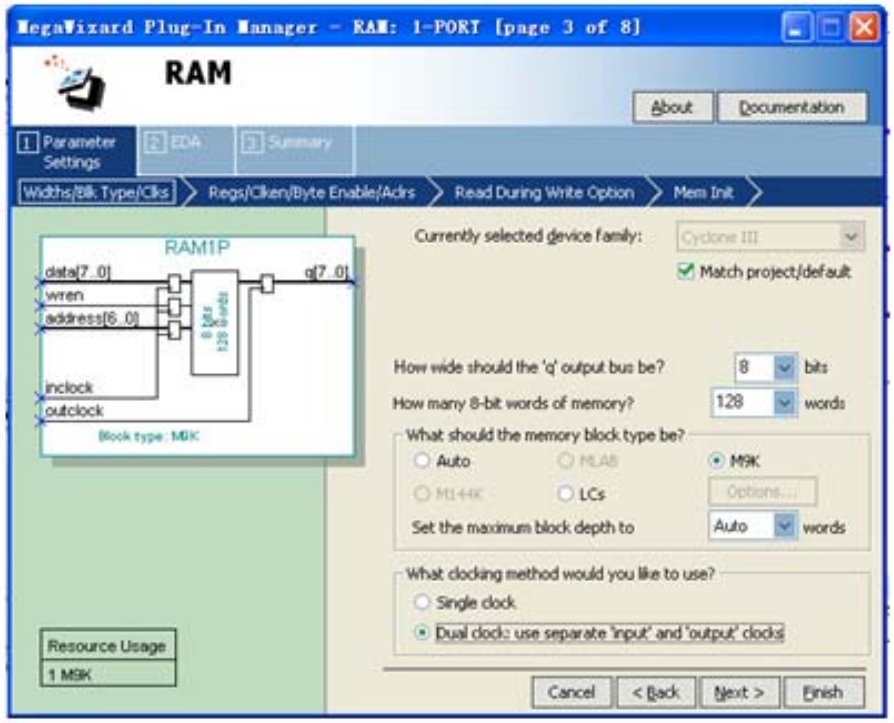


图 5-25　设定RAM参数

5.3 LPM_RAM模块的设置

5.3.2 LPM_RAM设置和调用

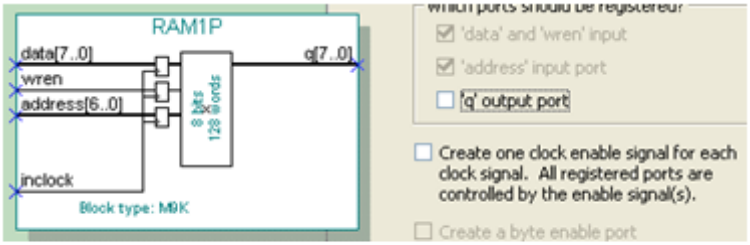


图 5-26　设定 RAM 仅输入时钟控制

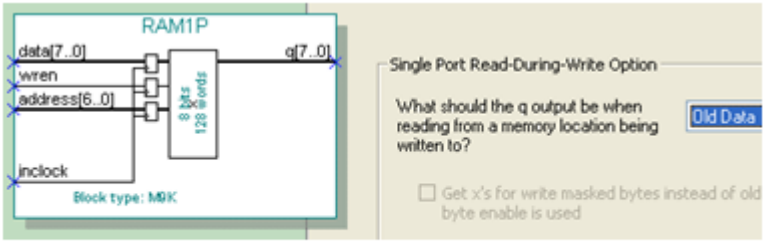


图 5-27　设定在写入同时读出原数据：Old Data

5.3 LPM_RAM模块的设置

5.3.2 LPM_RAM设置和调用

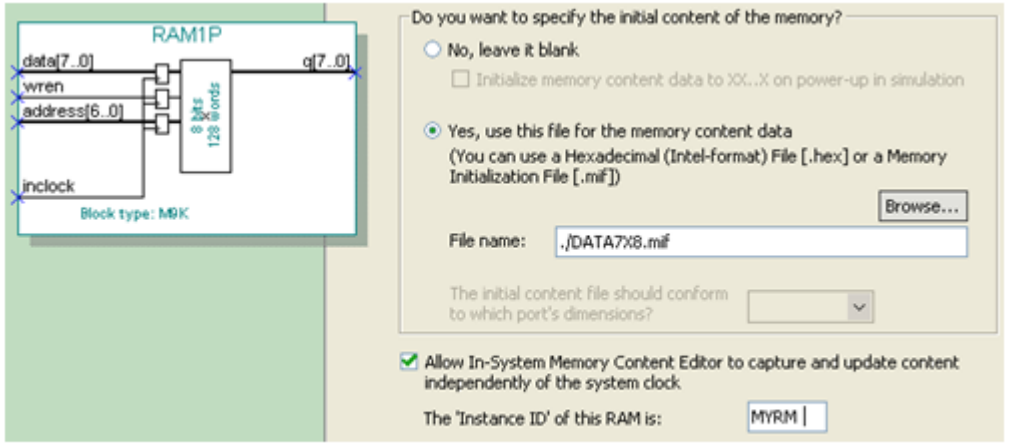


图 5-28 设定初始化文件和允许在系统编辑

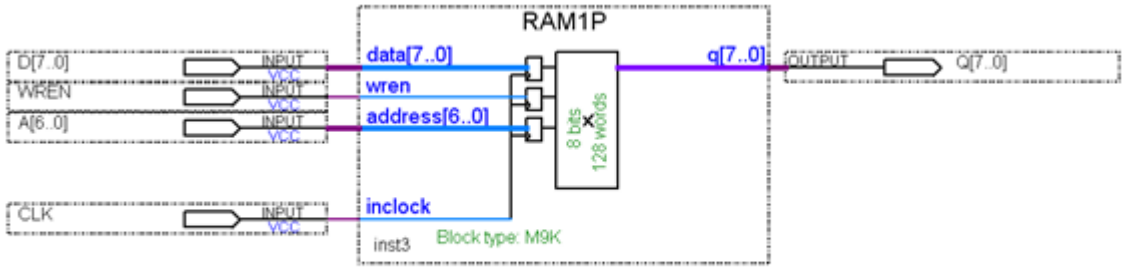


图 5-29 在原理图编辑器上连接好的 RAM 模块

5.3 LPM_RAM模块的设置

5.3.3 测试LPM_RAM

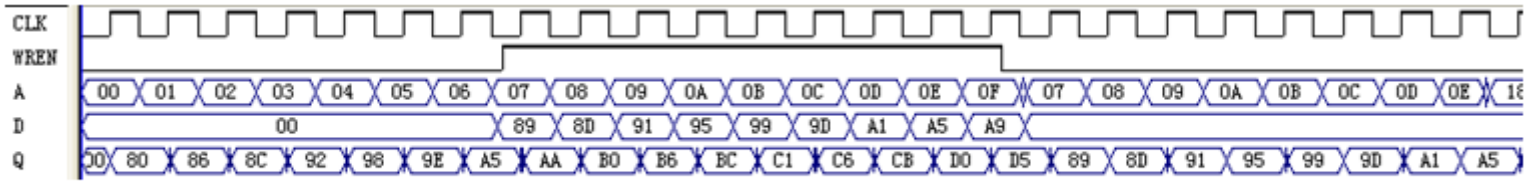


图 5-30 的 RAM 的仿真波形

5.3 LPM_RAM模块的设置

5.3.4 存储器的Verilog文本描述及相关属性应用

【例 5-7】

```
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端口
endmodule
```

5.3 LPM_RAM模块的设置

5.3.4 存储器的Verilog文本描述及相关属性应用

1. 存储器端口描述

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

5.3 LPM_RAM模块的设置

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 ;        //允许寄存器 A 整体赋值
MEM = 16'hABCD;       //不允许对存储器多个或者所有单元同时赋值
```

5.3 LPM_RAM模块的设置

3．存储器初始化文件属性应用

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

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

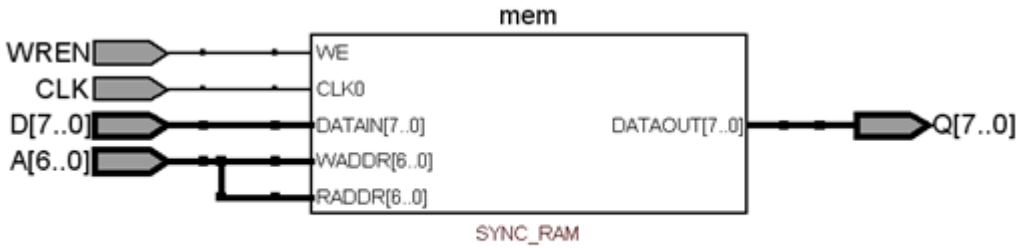


图 5-31 例 5-7 的 RAM78 的 RTL 图

5.3 LPM_RAM模块的设置

3．存储器初始化文件属性应用

Family	Cyclone III
Device	EP3C5E144C8
Timing Models	Final
Met timing requirements	N/A
Total logic elements	1,497 / 5,136 (29 %)
Total combinational functions	1,340 / 5,136 (26 %)
Dedicated logic registers	1,024 / 5,136 (20 %)
Total registers	1024
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 5-32 例 5-7 的编译报告

Family	Cyclone III
Device	EP3C5E144C8
Timing Models	Final
Met timing requirements	N/A
Total logic elements	168 / 5,136 (3 %)
Total combinational functions	156 / 5,136 (3 %)
Dedicated logic registers	98 / 5,136 (2 %)
Total registers	98
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 5-33 例 5-8 或图 5-29 的编译报告

5.3 LPM_RAM模块的设置

4．文本方式调用存储器LPM模块

【例 5-8】

```
module RAM78(Q1, D1, A1, CLK1, WREN1);
  output[7:0] Q1;     input[7:0] D1;    input[6:0] A1;  input  CLK1,WREN1;
   RAMP  IC1(.A(A1), .D(D1), .CLK(CLK1), .Q(Q1),  .WREN(WREN1) );
endmodule
```

【例 5-9】

```
module RAMP (A,D,CLK,WREN,Q);  //RAM1P.v
   input[6:0] A;   input[7:0] D;   input CLK;   input WREN ;
   output[7:0]  Q;
   altsyncram   U1(                              //例化 ram 模块
          .wren_a(WREN), .clock0(CLK), .address_a(A),
          .data_a(D), .q_a(Q),   .aclr0(1'b0),
          .aclr1(1'b0),  .address_b(1'b1), .addressstall_a(1'b0),
          .addressstall_b(1'b0), .byteena_a(1'b1), .byteena_b(1'b1),
          .clock1 (1'b1),   .clocken0 (1'b1), .clocken1 (1'b1),
          .clocken2(1'b1),.clocken3(1'b1),.data_b(1'b1),.wren_b(1'b0),
          .eccstatus (),.q_b(), .rden_a(1'b1), .rden_b(1'b1)
              );
  defparam        // 参数传递
    U1.clock_enable_input_a = "BYPASS",
    U1.clock_enable_output_a = "BYPASS",
    U1.init_file = "DATA7X8.mif",          //初始化数据文件,后缀最好小写
    U1.intended_device_family = "Cyclone III",
    U1.lpm_hint = "ENABLE_RUNTIME_MOD=YES,INSTANCE_NAME=MYRM",
    U1.lpm_type = "altsyncram",            //LPM 类型
    U1.numwords_a = 128,                   //数据数量 128
    U1.operation_mode = "SINGLE_PORT",
    U1.outdata_aclr_a = "NONE",            //无异步地址清 0
    U1.outdata_reg_a = "UNREGISTERED",     //输出无锁存
    U1.power_up_uninitialized = "FALSE",
    U1.ram_block_type = "M9K",
    U1.read_during_write_mode_port_a = "OLD_DATA",
    U1.widthad_a = 7,                      //地址线宽度 7
    U1.width_a = 8,                        //数据线宽度 8
    U1.width_byteena_a = 1;                //byteena_a 输入口宽度 1
  Endmodule
```

5.4 LPM_ROM的定制和使用

5.4.1 LPM_ROM定制和测试

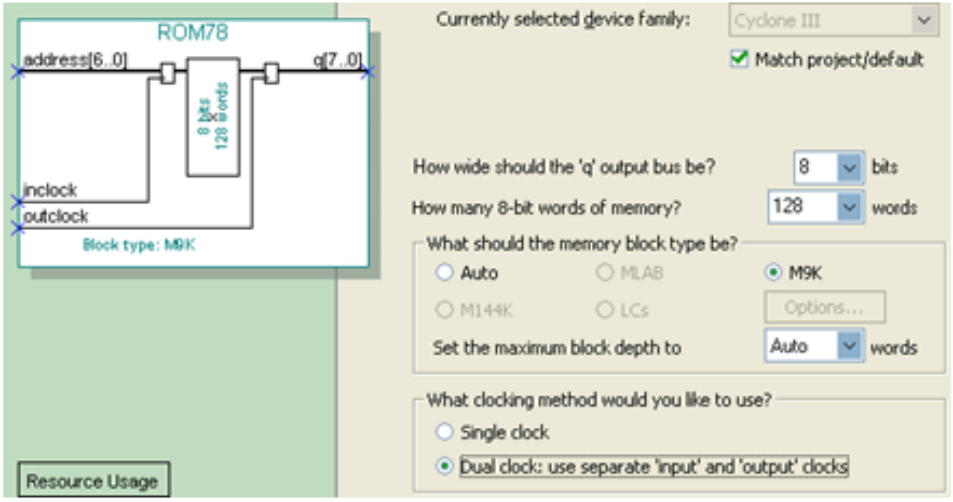


图 5-34 调用 LPM ROM 之参数设置

5.4 LPM_ROM的定制和使用

5.4.1 LPM_ROM定制和测试

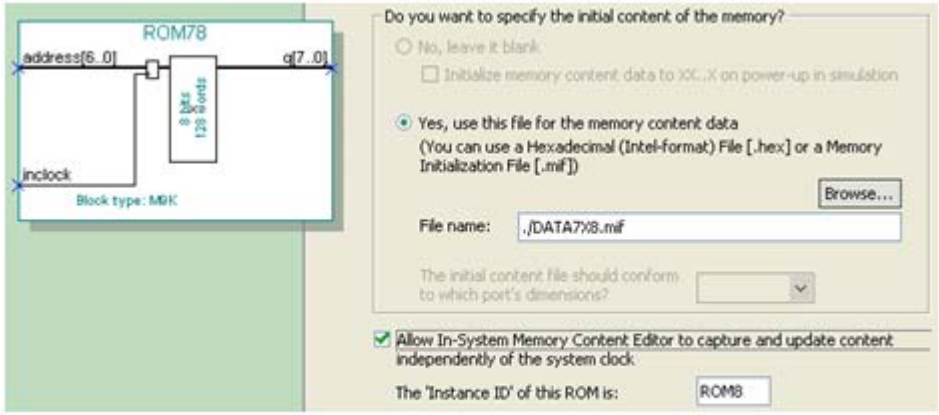


图 5-35 加入初始化配置文件

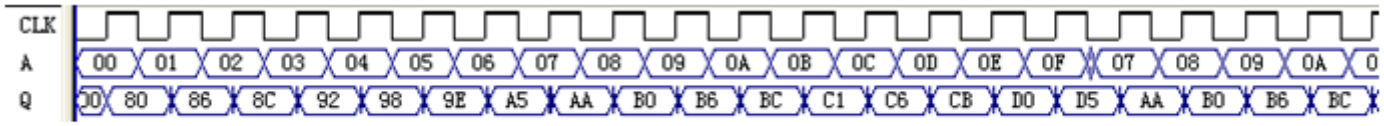
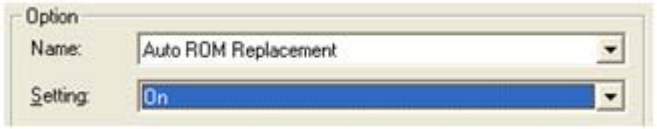


图 5-36 LPM ROM 仿真测试

5.4 LPM_ROM的定制和使用

5.4.2 LPM存储器模块替代设置



5.4 LPM_ROM的定制和使用

5.4.3 正弦信号发生器设计

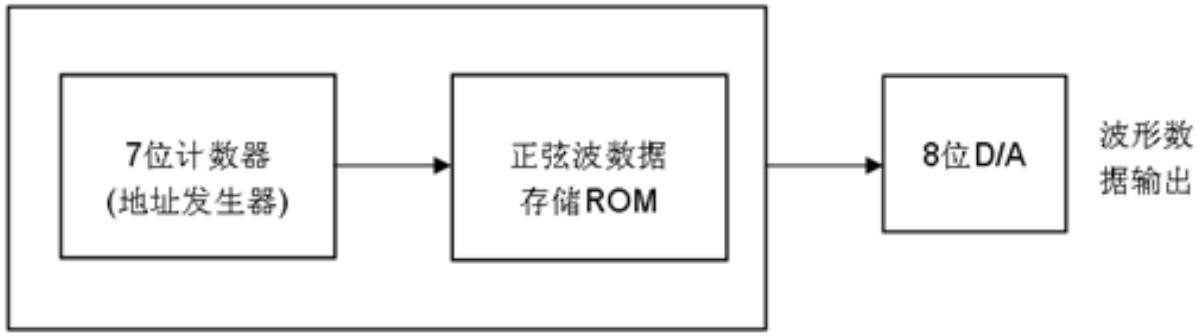


图 5-37　正弦信号发生器结构框图

5.4 LPM_ROM的定制和使用

5.4.3 正弦信号发生器设计

【例 5-10】

```
module SIN_GNT(RST,CLK,EN, Q,AR);
   output [7:0] Q ;  output [6:0] AR ; //AR是7位地址发生器输出测试口
   input  EN,CLK,RST ;   wire [6:0] TMP;    reg [6:0] Q1 ;
    always @(posedge CLK or negedge RST )
       if (!RST)   Q1 <=7'B0000000;
         else  if (EN)   Q1 <= Q1+1 ;
         else  Q1 <= Q1;
    assign TMP=Q1;        assign  AR=TMP;
    ROM78  IC1(.address(TMP), .inclock(CLK), .q (Q) );//例化ROM78.v
endmodule
```

【例 5-11】 ROM78.v

```
module ROM78 (address, inclock,    q);
   input   [6:0]  address; input  inclock; output[7:0]  q;
    ...
```

5.4 LPM_ROM的定制和使用

5.4.3 正弦信号发生器设计

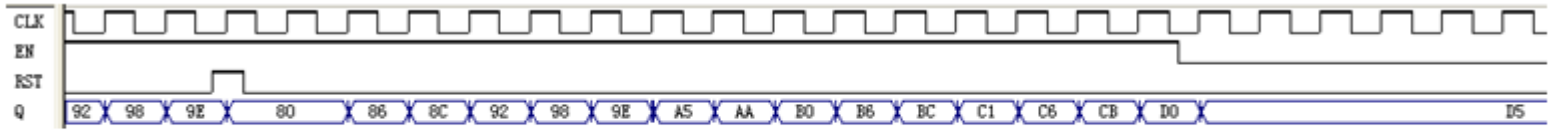


图 5-38　例 5-10 的仿真波形输出

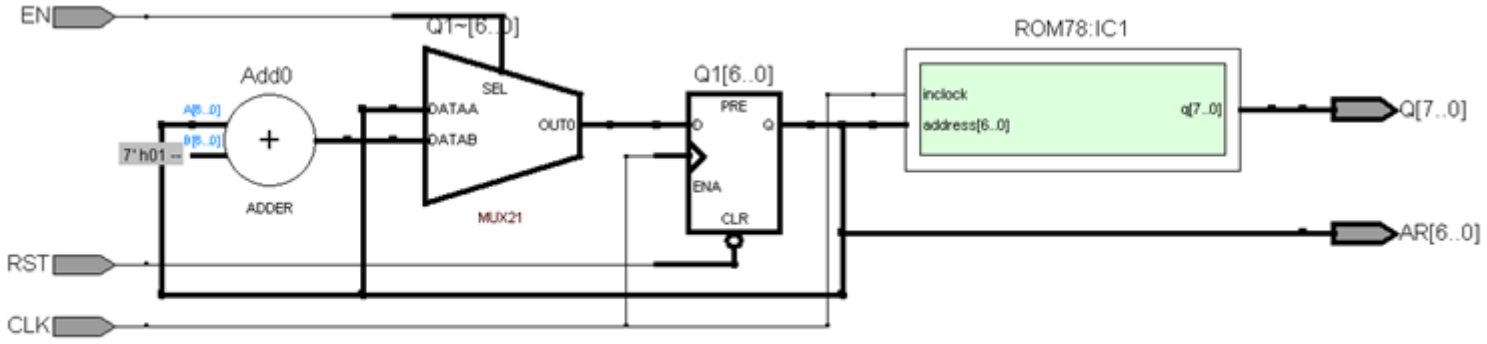


图 5-39　例 5-10 的 RTL 电路图

5.4 LPM_ROM的定制和使用

5.4.4 硬件实现和测试

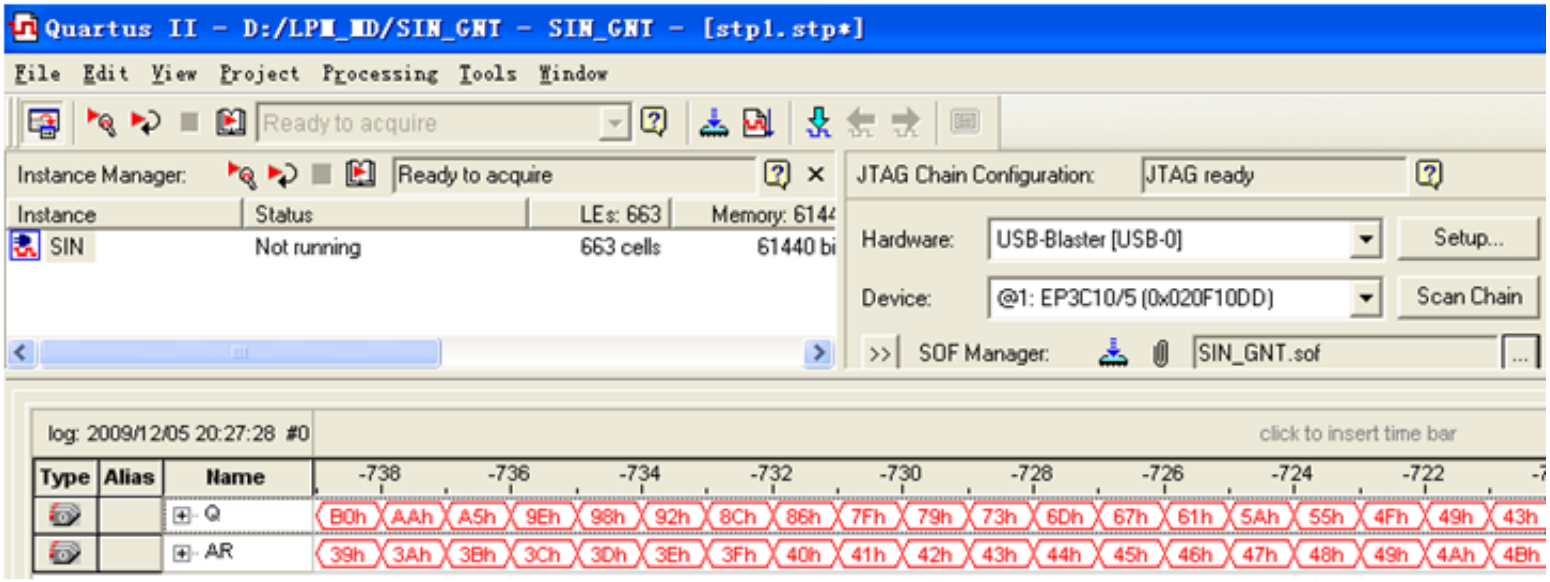


图 5-40 正弦信号发生器数据输出的 SignalTapII 测试图

5.4 LPM_ROM的定制和使用

5.4.4 硬件实现和测试

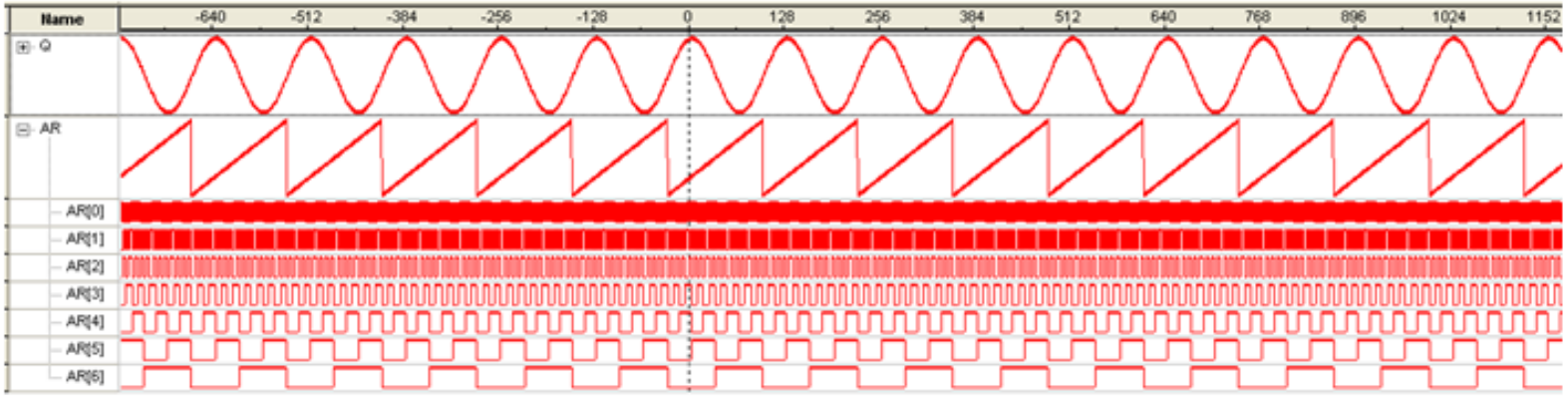


图 5-41 正弦信号发生器的 SignalTapII 的波形显示图

5.5 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。

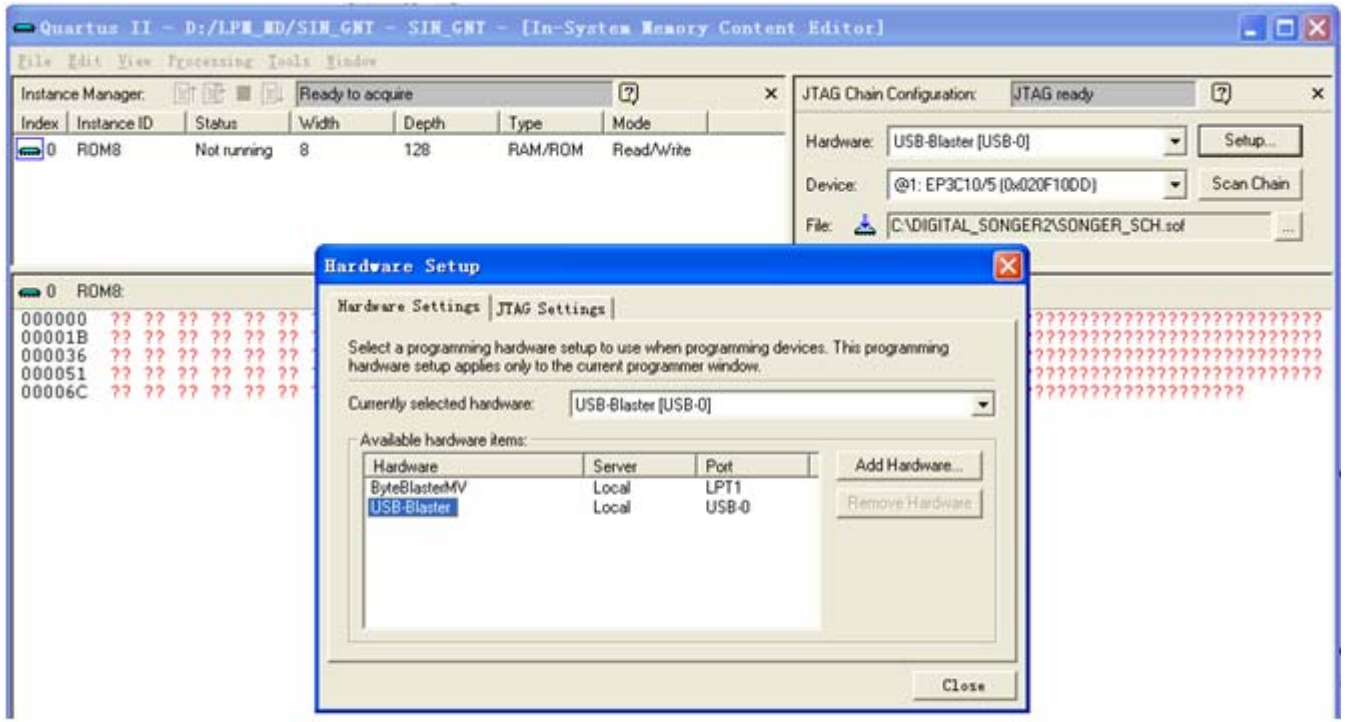


图 5-42　In-System Memory Content Editor 编辑窗

5.5 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。

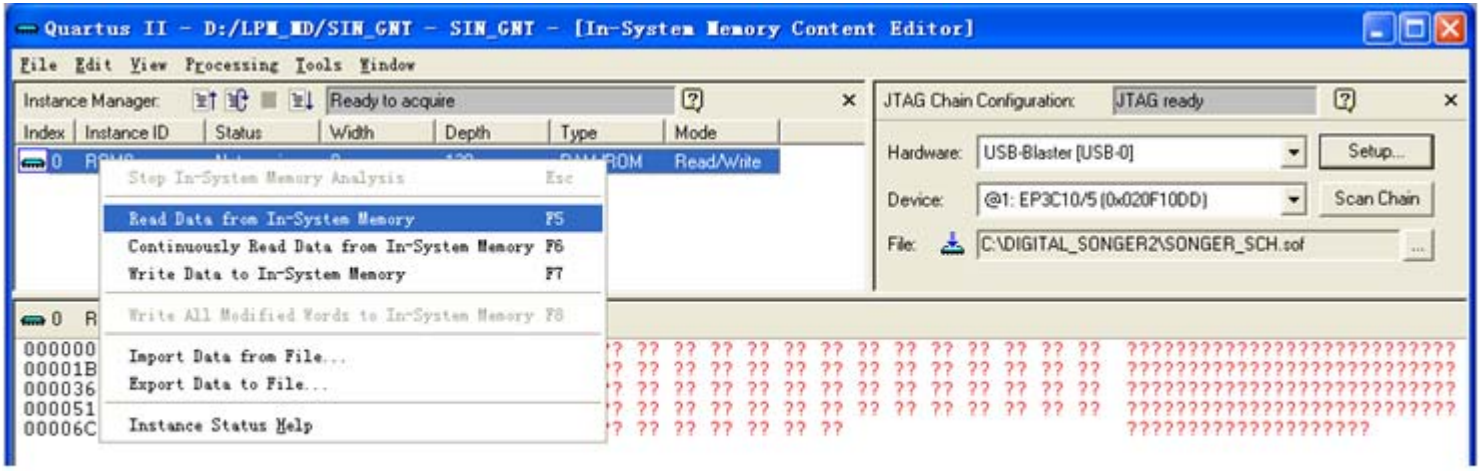


图 5-43　与实验系统上的 FPGA 通信正常情况下的编辑窗口界面

5.5 在系统存储器数据读写编辑器应用

（2）读取ROM中的波形数据。

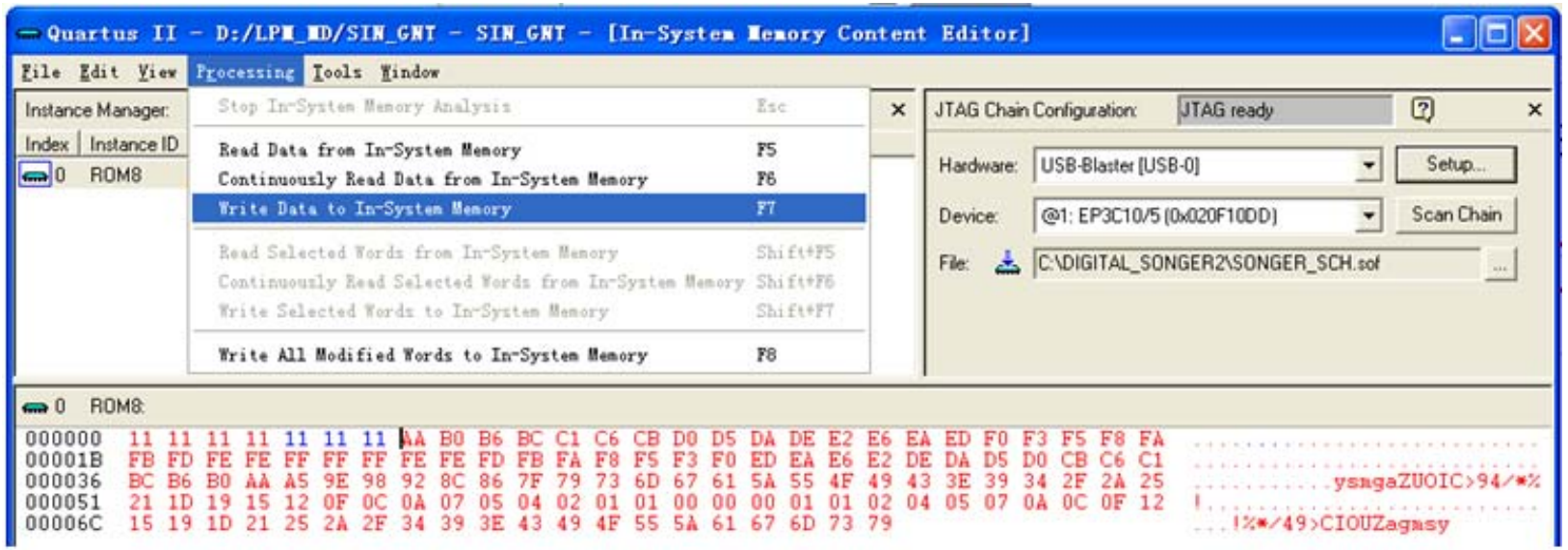


图 5-44　从 FPGA 中的 ROM 读取波形数据并编辑数据

5.5 在系统存储器数据读写编辑器应用

（3）写数据。

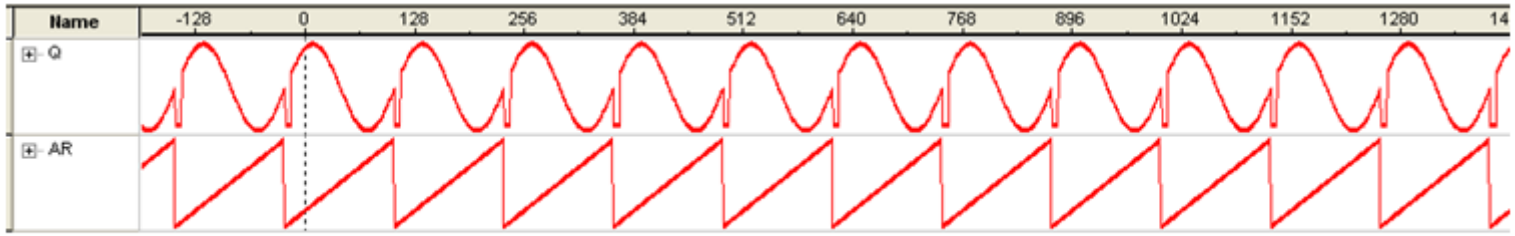


图 5-45　下载编辑数据后的 SignalTap II 采样波形

（4）输入输出数据文件。

5.6 嵌入式锁相环ALTPLL调用

5.6.1 嵌入式锁相环参数设置

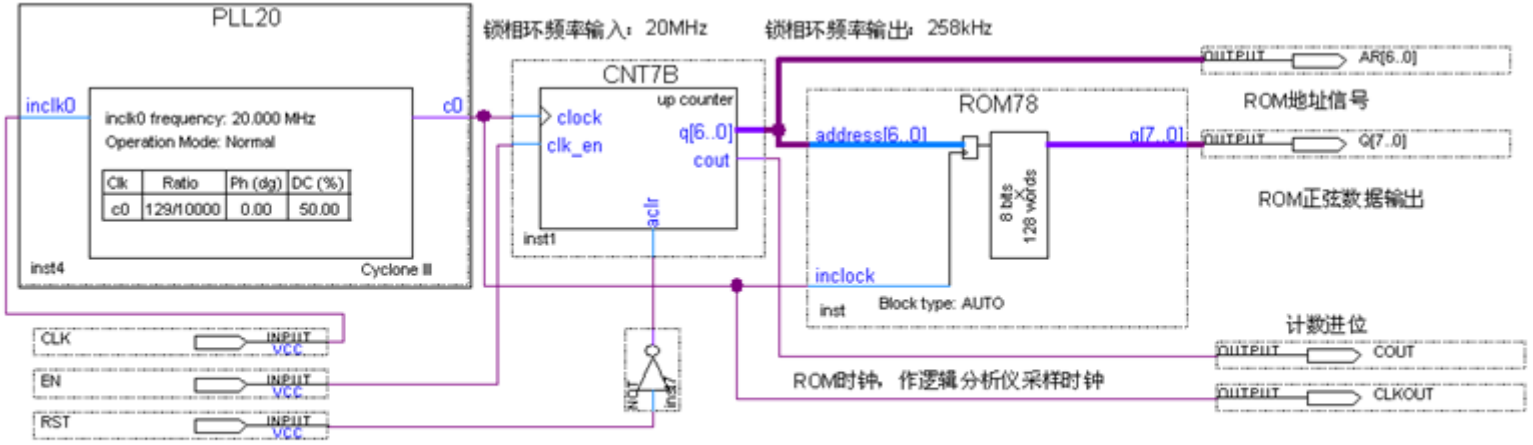


图 5-46　采用嵌入式锁相环作时钟的正弦信号发生器电路图

5.6 嵌入式锁相环ALTPLL调用

5.6.1 嵌入式锁相环参数设置

（1）为了在此原理图顶层设计中加入一个锁相环，在原理图编辑窗右键点击，选择**Insert->Symbol**。

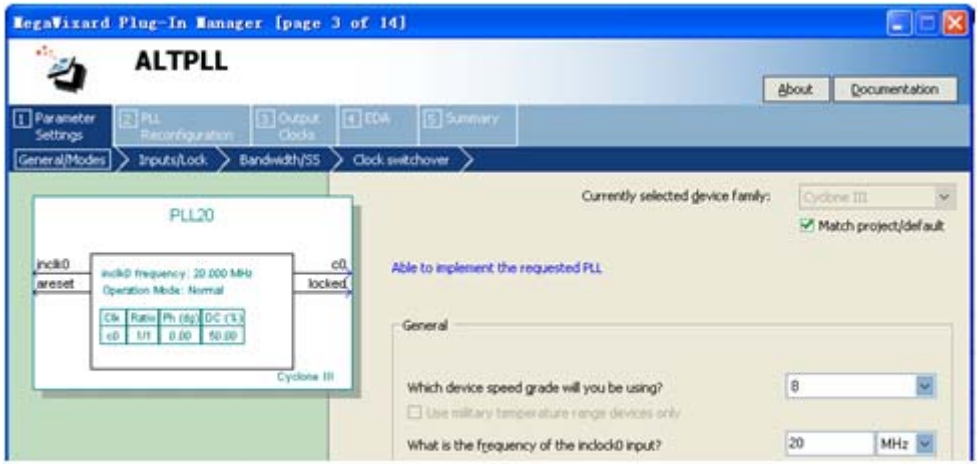


图 5-47 选择输入参考时钟 inclk0 为 20MHz

5.6 嵌入式锁相环ALTPLL调用

5.6.1 嵌入式锁相环参数设置

（2）在图5-47所示窗中首先设置输入时钟频率inclk0为20MHz。

（3）然后单击Next按钮。

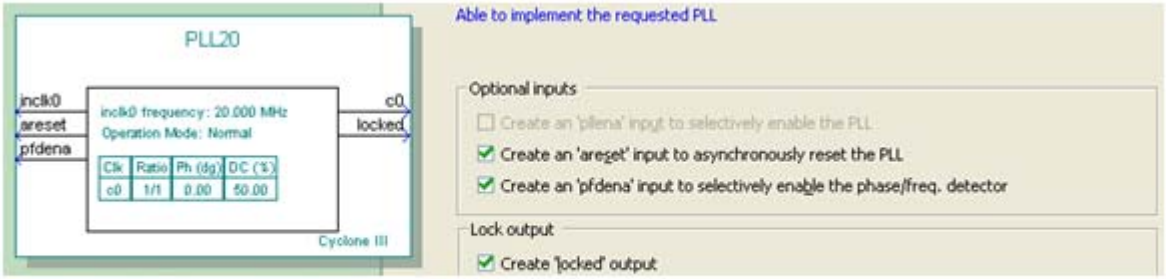


图 5-48 选择控制信号

5.6 嵌入式锁相环ALTPLL调用

5.6.1 嵌入式锁相环参数设置

（4）然后单击Next按钮，在不同的窗中进行设置。

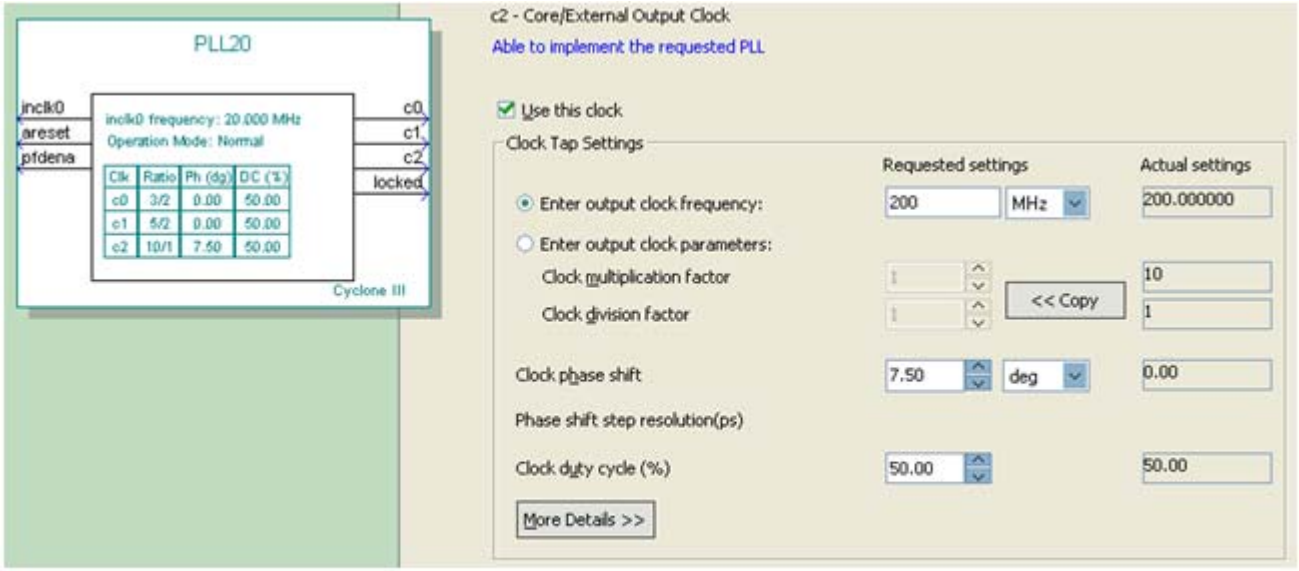


图 5-49 选择 e0 的输出频率为 200MHz

5.6 嵌入式锁相环ALTPLL调用

5.6.2 锁相环调用注意事项

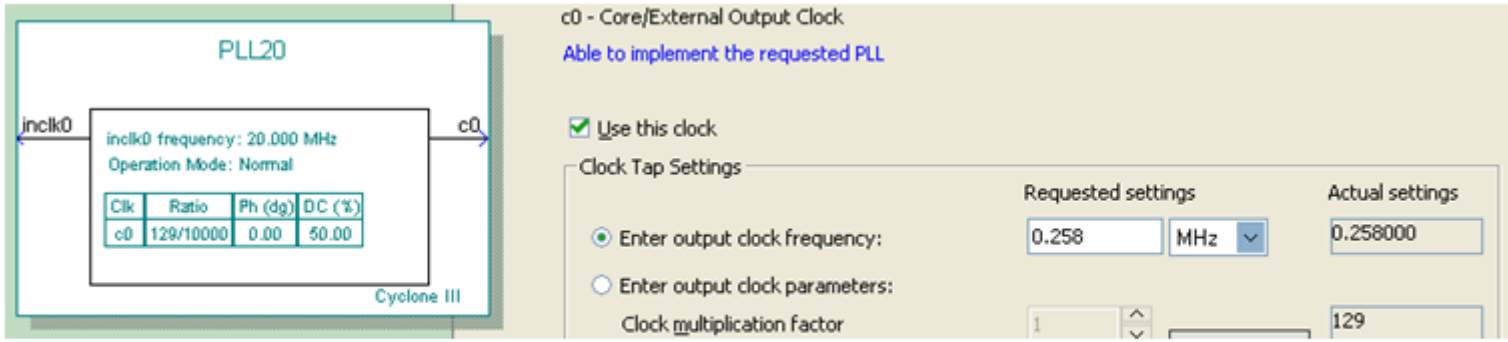


图 5-50 选择输出频率为 0.258MHz 作正弦信号发生器工作时钟

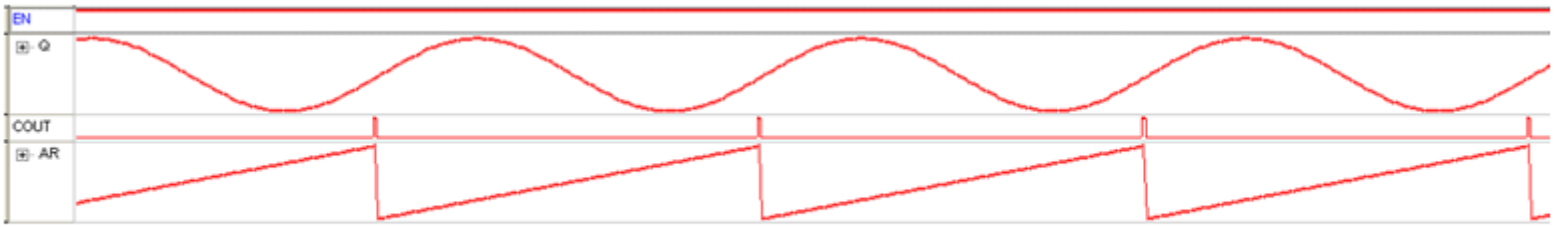


图 5-51 图 5-46 设计的逻辑分析仪实时采样输出

5.7 DDS实现原理与应用

5.7.1 DDS原理

$$f_{SIN} = M（f_{clk}/2^n）$$

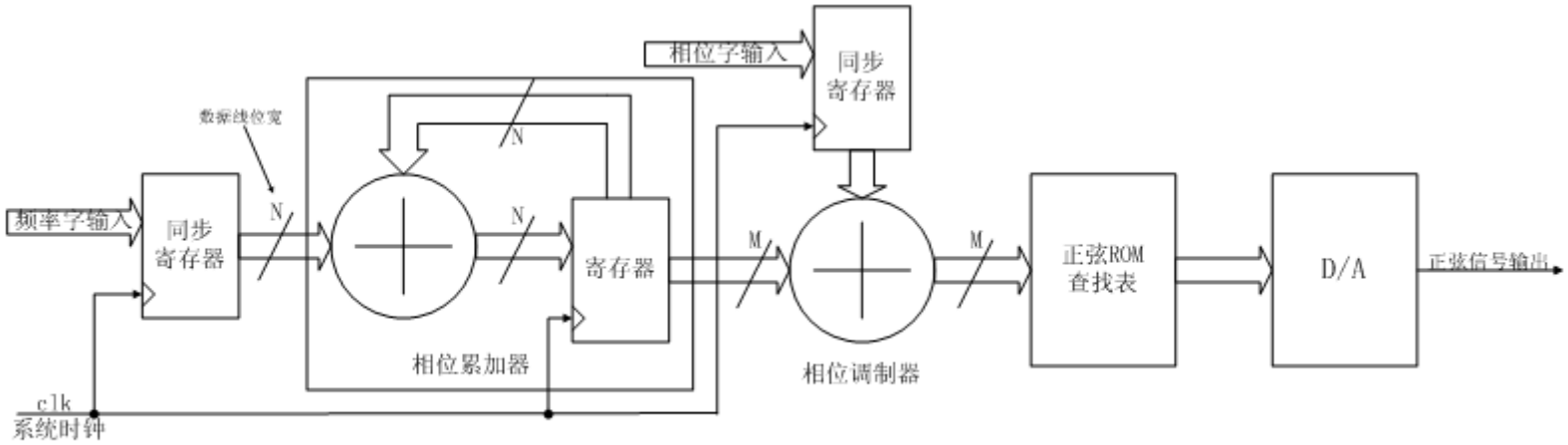


图 5-52　DDS 基本结构

5.7 DDS实现原理与应用

5.7.1 DDS原理

相位累加器宽	对应采样点数
8	256
12	4096
16	65536
20	1048576
24	16777216
28	268435456
32	4294967296

图 5-53 相位累加器位宽和采样点关系

5.7 DDS实现原理与应用

5.7.2 DDS信号发生器设计

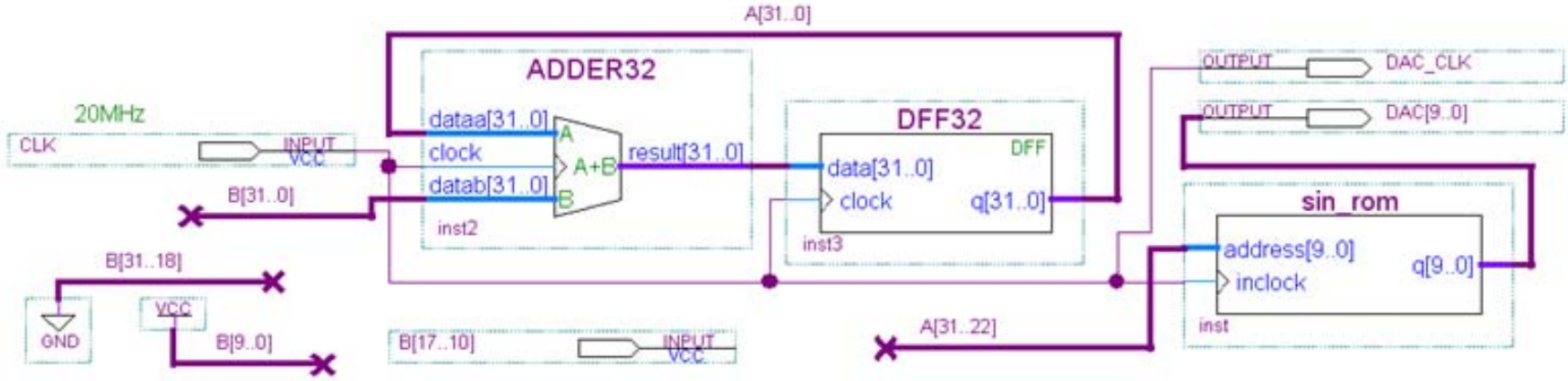


图 5-54　DDS 信号发生器电路顶层原理图

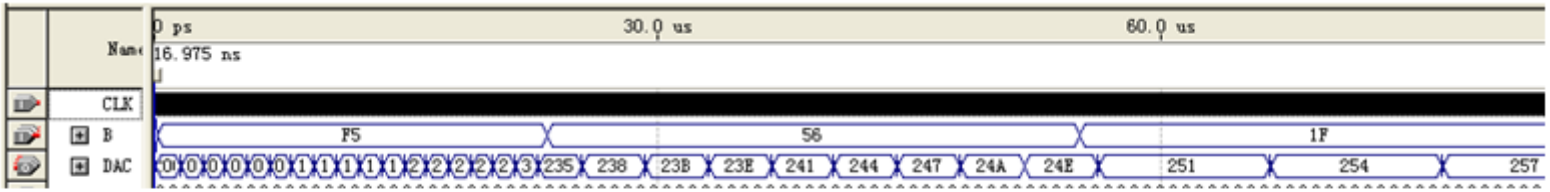


图 5-55　图 5-54 所示电路的仿真波形

实训项目

5-1．查表式硬件运算器设计

【例 5-12】

```
WIDTH = 8 ;
DEPTH = 256 ;
ADDRESS_RADIX = HEX ;
DATA_RADIX = HEX ;
CONTENT  BEGIN
00:00; 01:00; 02:00; 03:00; 04:00; 05:00; 06:00; 07:00; 08:00; 09:00;
10:00; 11:01; 12:02; 13:03; 14:04; 15:05; 16:06;  17:07; 18:08; 19:09;
20:00; 21:02; 22:04; 23:06; 24:08; 25:10; 26:12; 27:14; 28:16; 29:18;
30:00; 31:03; 32:06; 33:09; 34:12; 35:15; 36:18; 37:21; 38:24; 39:27;
40:00; 41:04; 42:08; 43:12; 44:16; 45:20; 46:24; 47:28; 48:32; 49:36;
50:00; 51:05; 52:10; 53:15; 54:20; 55:25; 56:30; 57:35; 58:40; 59:45;
60:00; 61:06; 62:12; 63:18; 64:24; 65:30; 66:36; 67:42; 68:48; 69:54;
70:00; 71:07; 72:14; 73:21; 74:28; 75:35; 76:42; 77:49; 78:56; 79:63;
80:00; 81:08; 82:16; 83:24; 84:32; 85:40; 86:48; 87:56; 88:64; 89:72;
90:00; 91:09; 92:18; 93:27; 94:36; 95:45; 96:54; 97:63; 98:72; 99:81;
END ;
```

实训项目

5-2 正弦信号发生器设计

5-3 基于**Verilog**表述的频率计设计

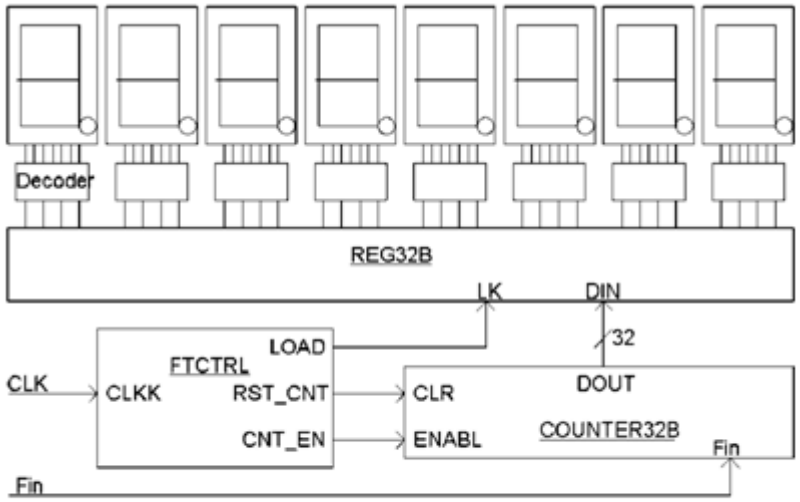


图 5-56 频率计电路框图

实训项目

【例 5-13】

```
module FTCTRL (CLKK, CNT_EN, RST_CNT, LOAD);
   input CLKK;        output CNT_EN, RST_CNT,LOAD;
   wire CNT_EN, LOAD;        reg RST_CNT,Div2CLK;
   always @(posedge CLKK)
      Div2CLK <= ~Div2CLK ;
   always @(CLKK or Div2CLK)  begin
     if (CLKK==1'b0 & Div2CLK==1'b0)  RST_CNT <= 1'b1 ;
       else  RST_CNT <= 1'b0 ;       end
   assign LOAD = ~Div2CLK ;     assign CNT_EN = Div2CLK ;
endmodule
```

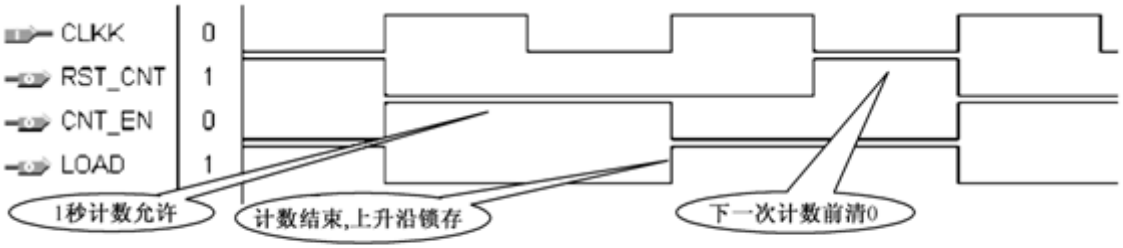


图 5-57 频率计测频控制器 FTCTRL 测控时序图

实训项目

5-4 DDS正弦信号发生器设计

5-5 移相信号发生器设计

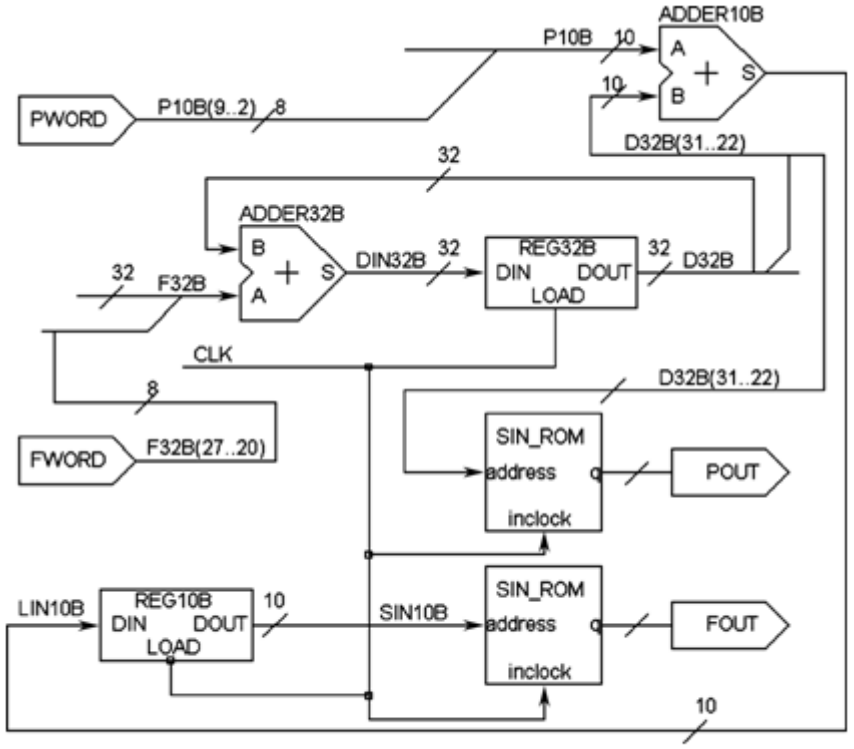


图 5-58 数字移相信号发生器电路模型图

实训项目

5-6 VGA简单图像显示控制模块设计

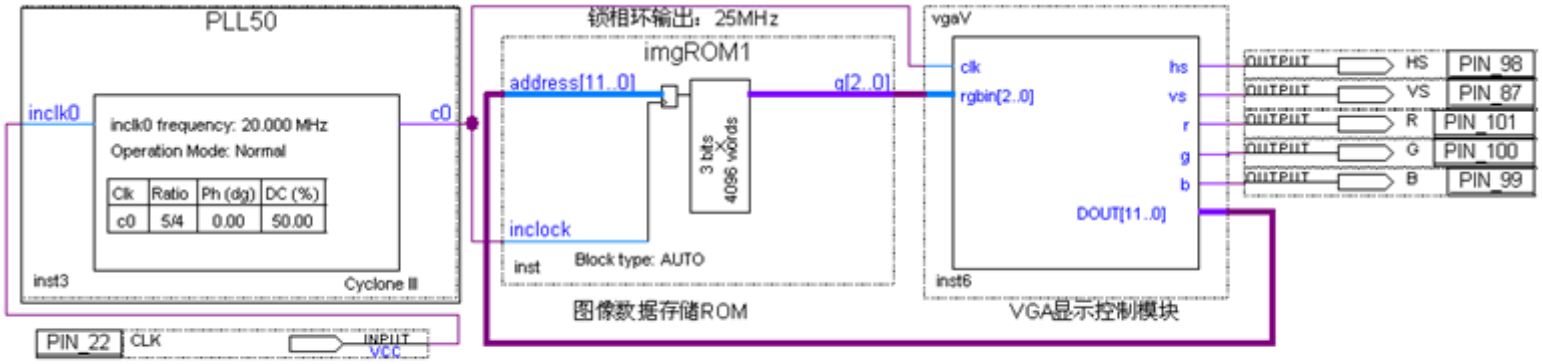


图5-59　VGA图像显示控制模块原理图

【例 5-14】

```
module vgaV (clk, hs, vs, r, g, b, rgbin, DOUT);//VGA
   input clk ;     //工作时钟25MHz
    output hs,vs;   //场同步，行同步信号
   output r,g,b ;  // 红，绿，蓝信号，
   input[2:0] rgbin; //像素数据
   output[11:0] DOUT; //图像数据ROM的地址信号
   reg[9:0] hcnt, vcnt;      reg r,g,b;      reg hs,vs;
   assign DOUT = {vcnt[5:0], hcnt[5:0]} ;
   always @(posedge clk)  begin  //水平扫描计数器
      if (hcnt<800)   hcnt<=hcnt+1 ;
      else            hcnt<={10{1'b0}} ;
      end
   always @(posedge clk)   begin  //垂直扫描计数器
      if (hcnt==640+8)    begin
         if (vcnt<525)  vcnt<=vcnt+1 ;
         else    vcnt<={10{1'b0}} ;  end   end
   always @(posedge clk)   begin //场同步信号发生
      if ((hcnt>=640+8+8) & (hcnt<640+8+8+96))
        hs<=1'b0 ; else  hs<=1'b1 ;    end
   always @(vcnt)   begin   //行同步信号发生
      if ((vcnt>=480+8+2) & (vcnt<480+8+2+2))
         vs<=1'b0 ;  else  vs<=1'b1 ;  end
   always @(posedge clk)   begin
      if (hcnt<640 & vcnt<480)   //扫描终止
      begin  r<=rgbin[2] ;  g<=rgbin[1] ;  b<=rgbin[0];   end
      else begin  r<=1'b0; g<=1'b0;  b<=1'b0;  end
   end    endmodule
```

实训项目

5-7 乐曲硬件演奏电路设计

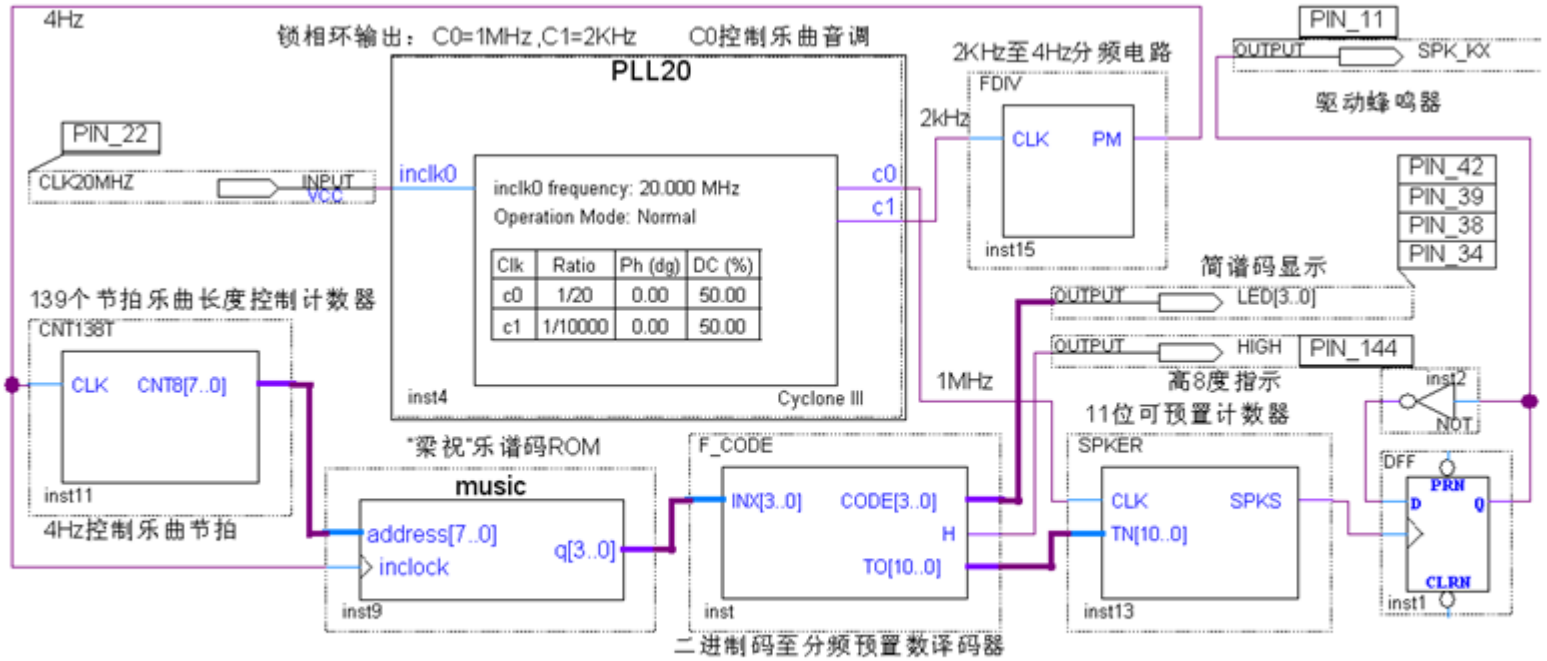


图 5-60 乐曲演奏电路顶层设计

实训项目

5-7 乐曲硬件演奏电路设计

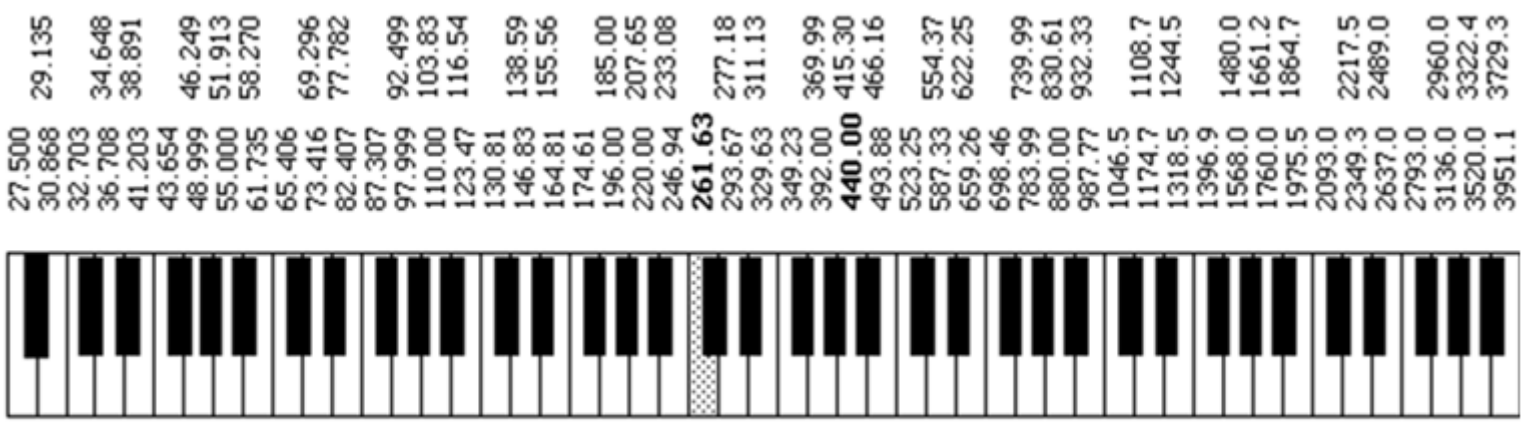


图5-61 电子琴音阶基频对照图（单位Hz）

实训项目

5-7 乐曲硬件演奏电路设计

【例 5-15】

```
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
```

【例 5-16】

```
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
```

实训项目

5-7 乐曲硬件演奏电路设计

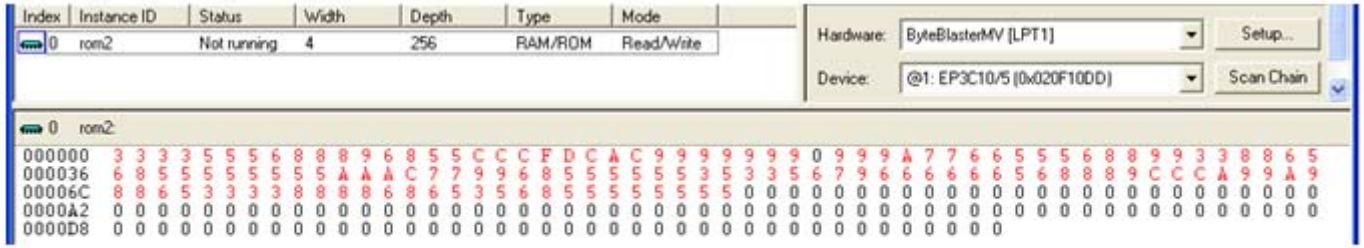


图5-62 In-System Memory Content Editor对MUSIC模块的数据读取

实训项目

5-7 乐曲硬件演奏电路设计

【例 5-17】

```
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
```

【例 5-18】

```
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
```

实训项目

【例 5-19】

```
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 ;
```

实训项目

5-8 数码扫描显示电路设计

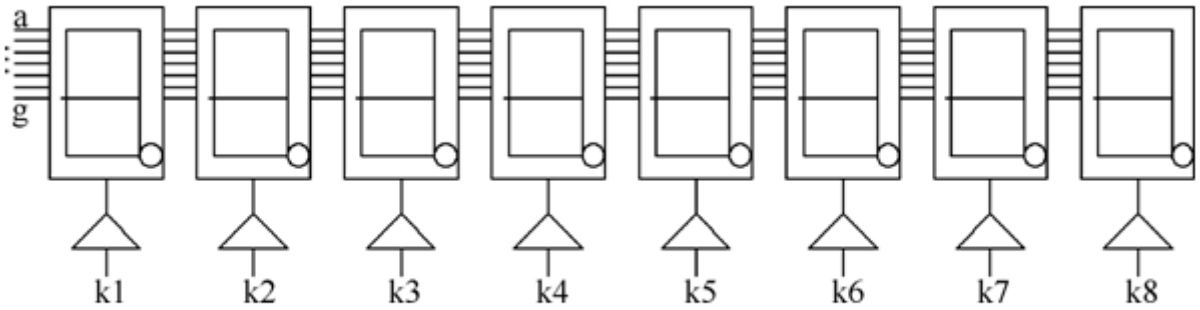


图 5-63　8 位数码扫描显示电路