EDA技术

第6章 IP应用及相关语法



《EDA技术实用教程——Verilog HDL版》
（第七版）
配套课件

EDA Lab

IP核



- Intellectual Property Core
- 知识产权核
- 在集成电路设计中，指的是一种可重用的逻辑单元或芯片设计模块。这些模块通常已经经过验证，设计人员可以基于这些模块进行设计，从而缩短设计周期
- Altera
  - Quartus主界面中IP Catalog
  - LPM
  - MegaCore

EDA Lab

LPM参数可设置模块库



- LPM: 参数可设置模块库。用户可调用LPM库中的模块，并为其设定适当的参数，就能满足设计需要。
- Quartus II的宏功能模块与LPM函数包括：
  - 算数组件
  - 组合电路
  - I/O组件
  - 存储器编译器
  - 存储组件
  - 各类专业核

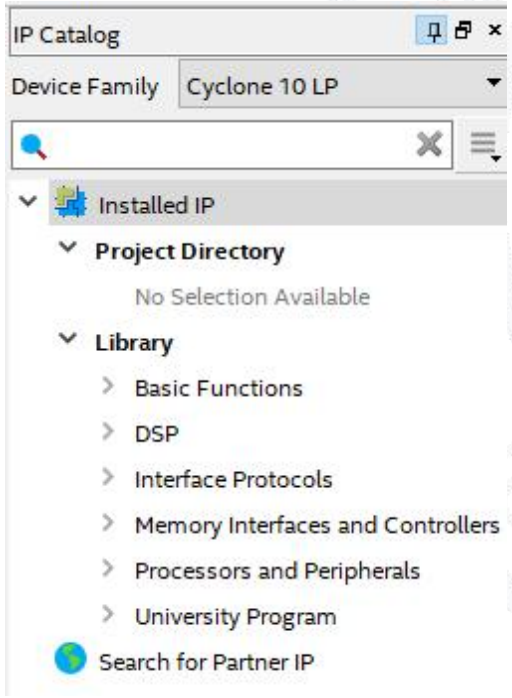


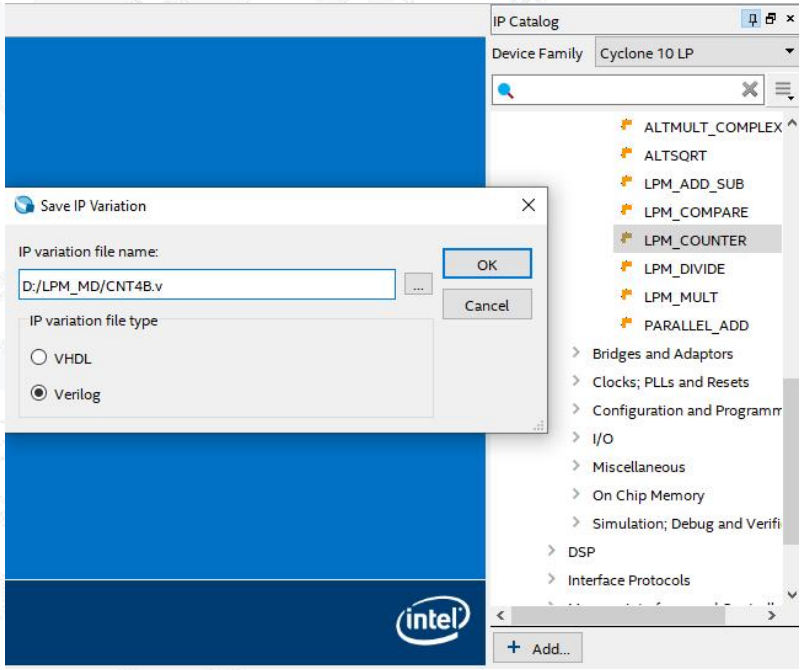
HANGZHOU DIANZI UNIVERSITY 1956

6.1 计数器LPM宏模块调用

6.1.1 计数器LPM模块文本代码的调用

（1）打开宏功能块调用管理器





6.1 计数器LPM宏模块调用

6.1.1 计数器LPM模块文本代码的调用

（2）单击Next按钮后打开如图所示对话框



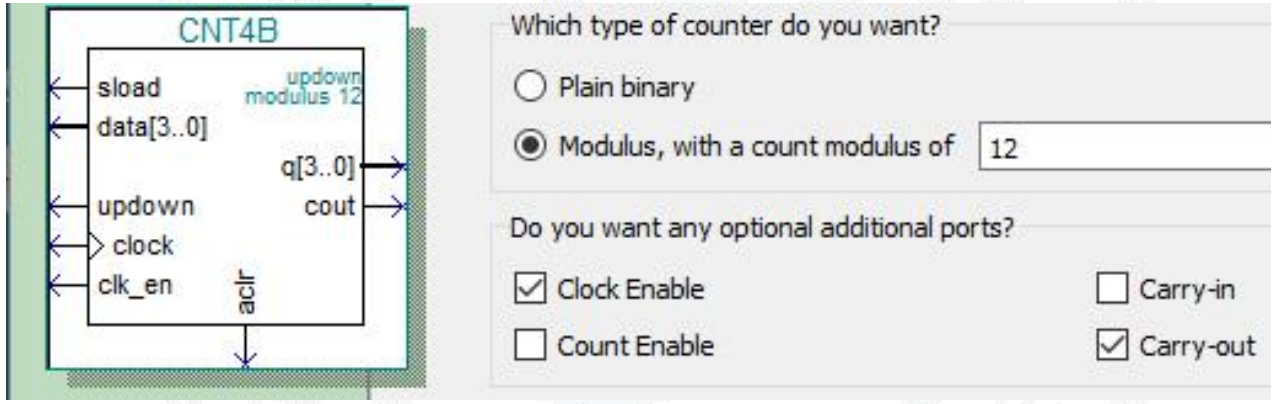
选择4位计数器；

选择“Create an updown input...”，使计数器有加减控制功能。

6.1 计数器LPM宏模块调用

6.1.1 计数器LPM模块文本代码的调用

（3）再单击Next按钮，打开如图所示的对话框



Plain binary: 普通二进制计数器

Modulus…:模（）计数器

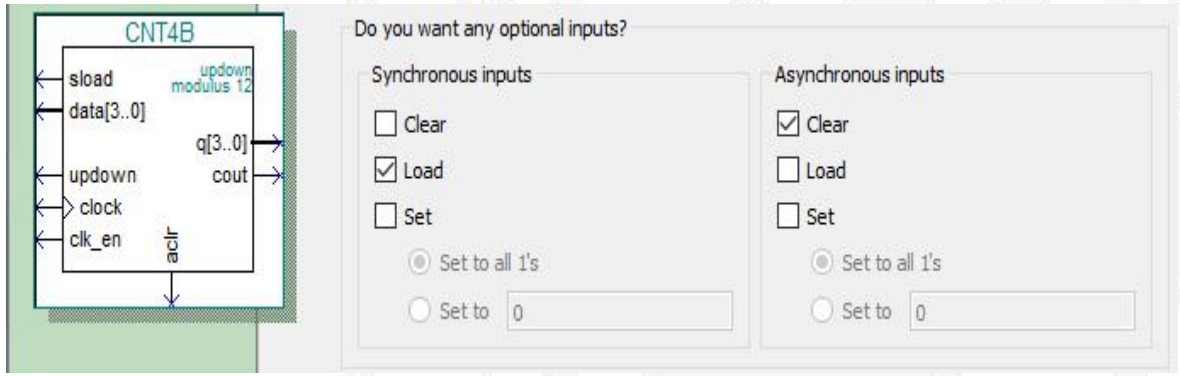
Clock Enable: 时钟使能控制

Carry-out:进位输出

6.1 计数器LPM宏模块调用

6.1.1 计数器LPM模块文本代码的调用

（4）再单击Next按钮，打开如图所示的对话框



选择同步加载和异步清0

以上流程设置生成了LPM计数器的Verilog文件CNT4B.v，可被高一层次的Verilog程序作为计数器元件调用。

6.1 计数器LPM宏模块调用



6.1.2 LPM计数器代码与参数传递语句

【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));
```

EDA Lab

6.1 计数器LPM宏模块调用



6.1.2 LPM计数器代码与参数传递语句

【例6-1】

```
      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
```

EDA Lab

6.1 计数器LPM宏模块调用



6.1.2 LPM计数器代码与参数传递语句

【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6-2是一个defparam语句应用示例。模块REG24B是一个24位寄存器，其中调用了可参数设置的LPM寄存器模块LPM_FF。

EDA Lab

6.1 计数器LPM宏模块调用

6.1.2 LPM计数器代码与参数传递语句

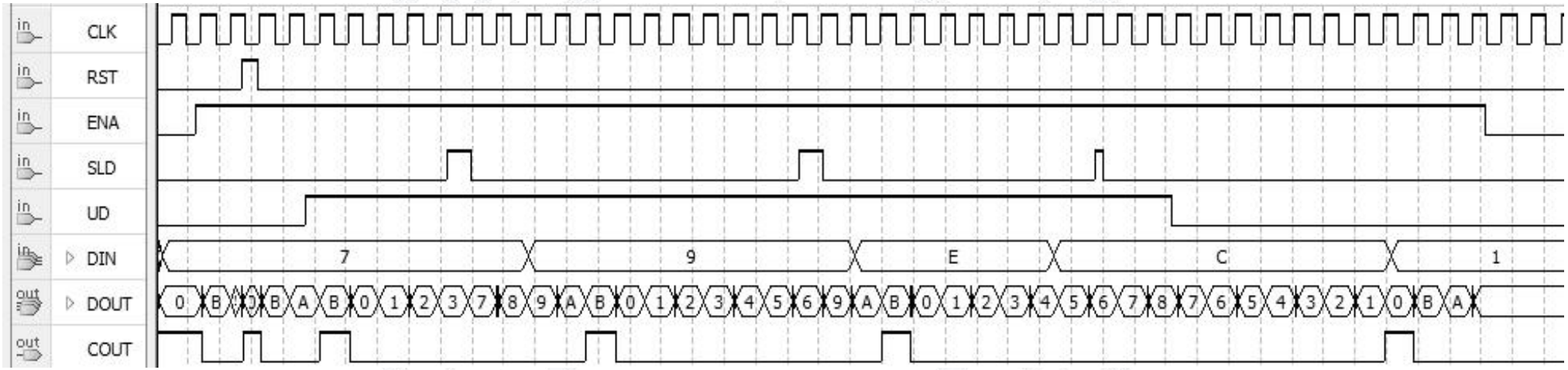
【例6-3】

```
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
```

设计一个程序（例6-3）调用例6-1计数器文件。

6.1 计数器LPM宏模块调用

6.1.3 创建工程与仿真测试



CNT4BIT.v的仿真波形

6.2 利用属性控制乘法器的构建



为实现乘法器，如果按照通常表述方式，综合出的乘法器会占用大量逻辑资源，而且速度不一定快。最常用的方法是调用FPGA内部已嵌入的硬件乘法器，此类乘法器常用于DSP技术中，故称DSP模块。

【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
```

以R2为输出口的乘法器以纯组合逻辑方式的宏单元构建。

以R1为输出口的乘法器调用FPGA的嵌入式乘法器来构建。

EDA Lab

6.2 利用属性控制乘法器的构建



```
wire  [15:0]  R2, R1 /*
synthesis multstyle = "
logic " */
```

Flow Status	Successful - Sat Mar 27 23:38:53 2021
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Standard Edition
Revision Name	MULT8
Top-level Entity Name	MULT8
Family	Cyclone 10 LP
Device	10CL006YU256C8G
Timing Models	Final
Total logic elements	95 / 6,272 (2 %)
Total registers	0
Total pins	64 / 177 (36 %)
Total virtual pins	0
Total memory bits	0 / 276,480 (0 %)
Embedded Multiplier 9-bit elements	1 / 30 (3 %)
Total PLLs	0 / 2 (0 %)

完全用逻辑宏单元构建乘法器的编译报告

- 资源使用: 95个LE, 0个DSP乘法器模块

```
wire  [15:0]  R2, R1 /*
synthesis multstyle =
"dsp" */;
```

Flow Status	Successful - Sat Mar 27 23:42:52 2021
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Standard Edition
Revision Name	MULT8
Top-level Entity Name	MULT8
Family	Cyclone 10 LP
Device	10CL006YU256C8G
Timing Models	Final
Total logic elements	0 / 6,272 (0 %)
Total registers	0
Total pins	64 / 177 (36 %)
Total virtual pins	0
Total memory bits	0 / 276,480 (0 %)
Embedded Multiplier 9-bit elements	2 / 30 (7 %)
Total PLLs	0 / 2 (0 %)

调用了DSP模块的编译报告

- 资源使用: 2个DSP乘法器模块, 0个LE

此设计高速且节省逻辑资源

EDA Lab

6.3 LPM_RAM宏模块的设置与使用

6.3.1 初始化文件及其生成

存储器的初始化文件就是指可配置于**LPM_RAM**或**LPM_ROM**中的数据或程序文件代码。

初始化文件的格式有3种：**mif　hex　dat**

1．mif格式文件

（1）直接编辑法

File -> New -> Memory Initialization

File ->产生mif数据文件大小选择窗口->

在mif数据表格中键入数据

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

Lab

6.3 LPM_RAM宏模块的设置与使用

6.3.1 初始化文件及其生成

（2）文件编辑法

【例6-5】

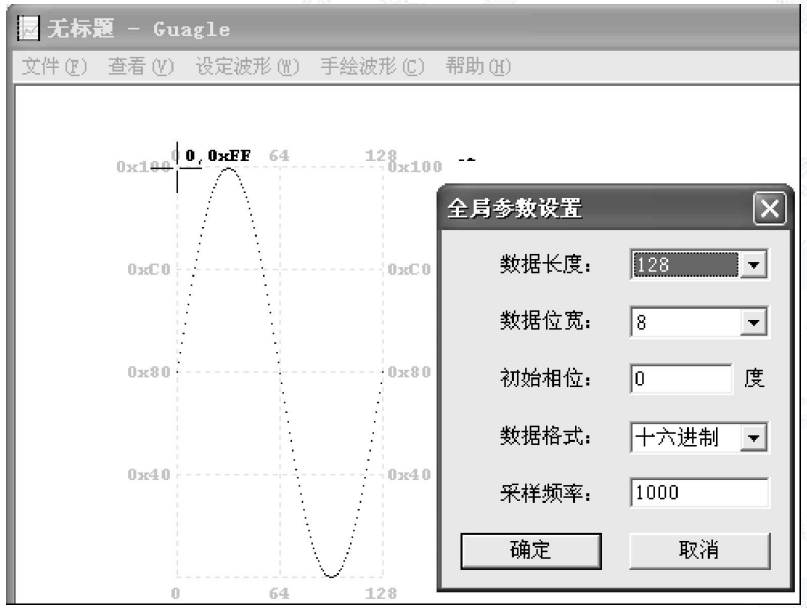
```
DEPTH=128;                    : 数据深度，即存储的数据个数
WIDTH=8;                      : 输出数据宽度
ADDRESS_RADIX = HEX;          : 地址数据类型，HEX表示选择十六进制数据类型
DATA_RADIX = HEX;             : 存储数据类型，HEX表示选择十六进制数据类型
CONTENT                       : 此为关键词
BEGIN                         : 此为关键词
0000        :      0080;
0001        :      0086;
0002        :      008C;
… (数据略去)
007E        :      0073;
007F        :      0079;
END;
```

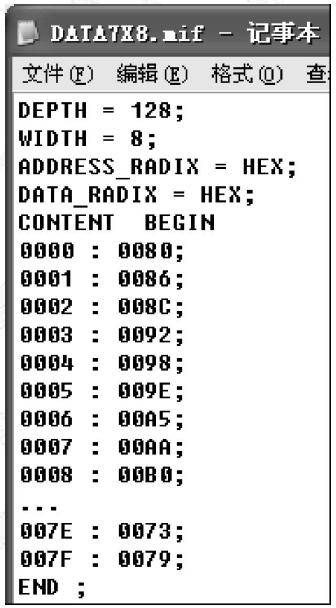
6.3 LPM_RAM宏模块的设置与使用

6.3.1 初始化文件及其生成

（3）高级语言生成：用C、MATLAB等高级语言工具生成

（4）专用mif文件生成器：mif maker





6.3 LPM_RAM宏模块的设置与使用



6.3.1 初始化文件及其生成

2．hex格式文件

（1）方法1：New -> Hexadecimal (Intel-Format) File ->存盘为hex格式文件。

（2）方法2：利用汇编程序编辑器将数据编辑于hex数据编辑窗中，然后用汇编编译器产生hex格式文件。

3．dat格式文件

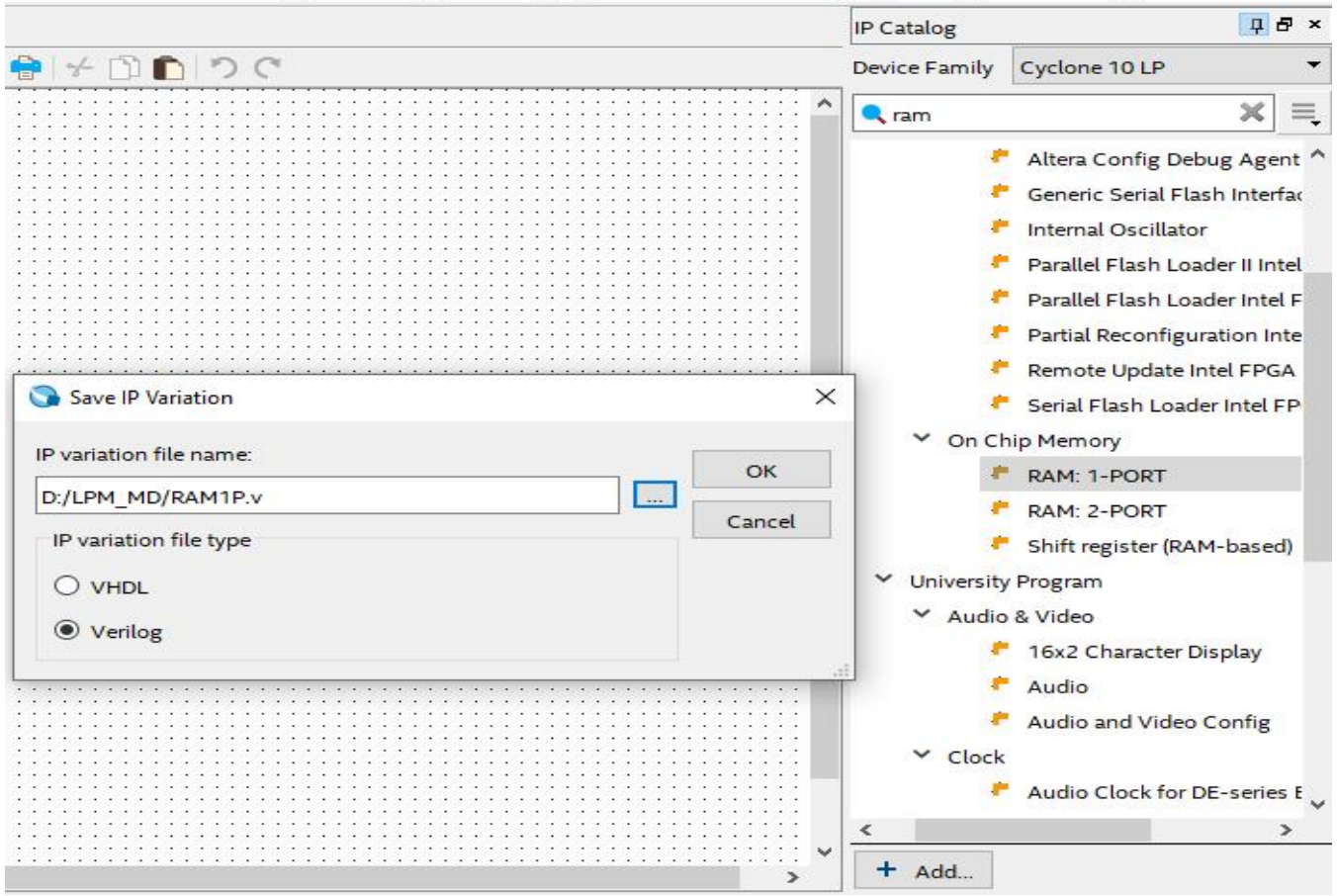
dat格式的数据文件的调用可用标准的Verilog语句直接实现。

EDA Lab

6.3 LPM_RAM宏模块的设置与使用

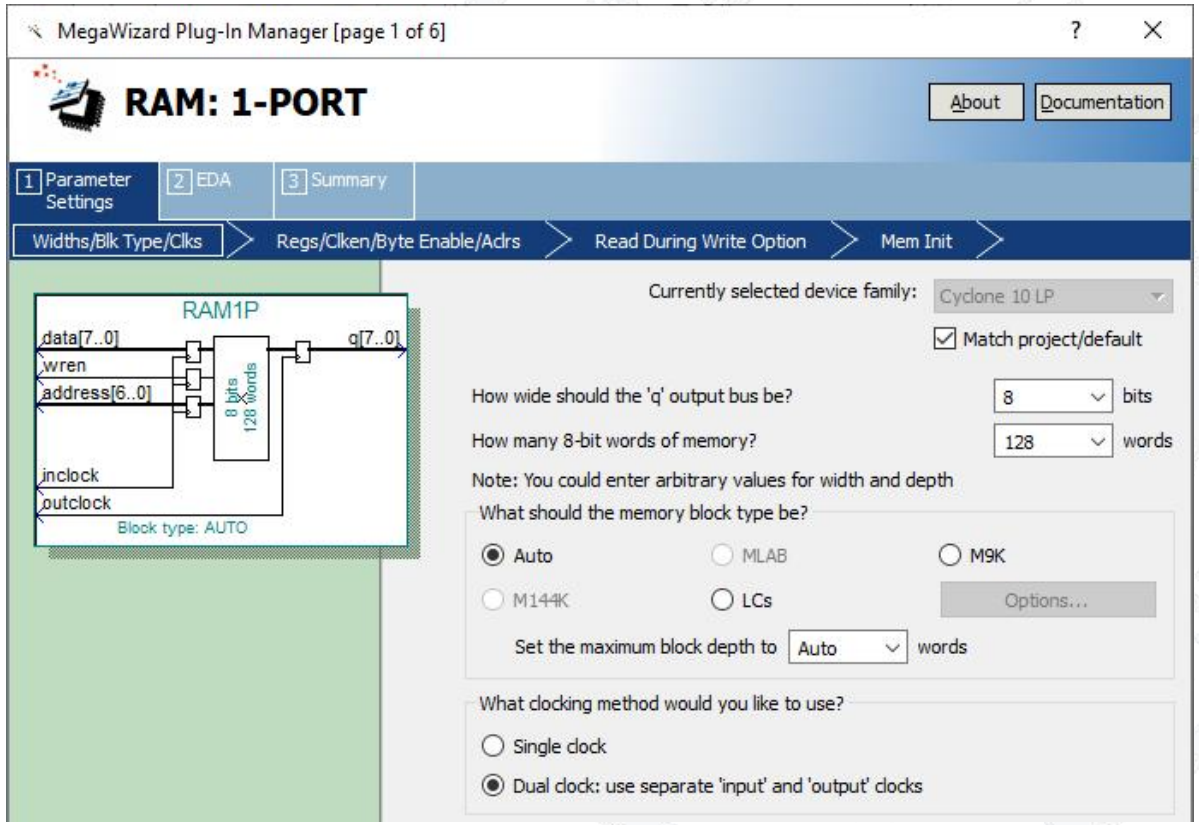
6.3.2 LPM_RAM的设置和调用

在IP Catalog中，选择On-chip Memory项下的单口RAM模块：RAM: 1-PORT



6.3 LPM_RAM宏模块的设置与使用

6.3.2 LPM_RAM的设置和调用



数据位8位

数据深度128

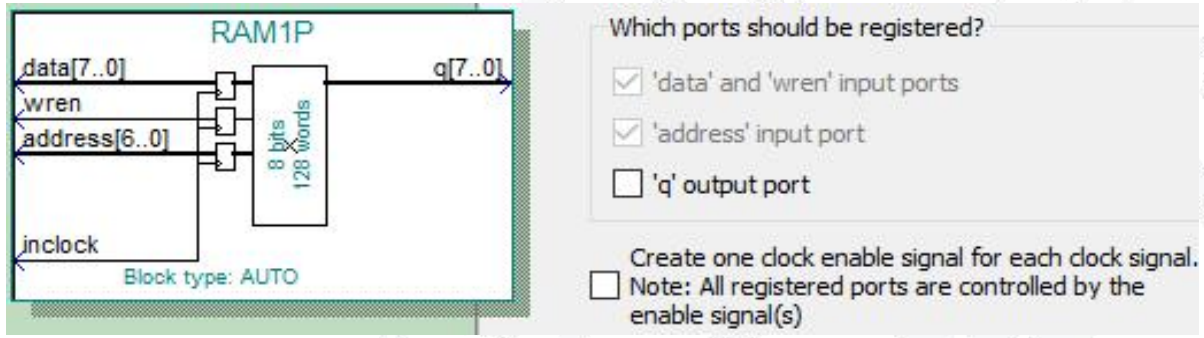
对应CycloneIII

存储器构建方式M9K

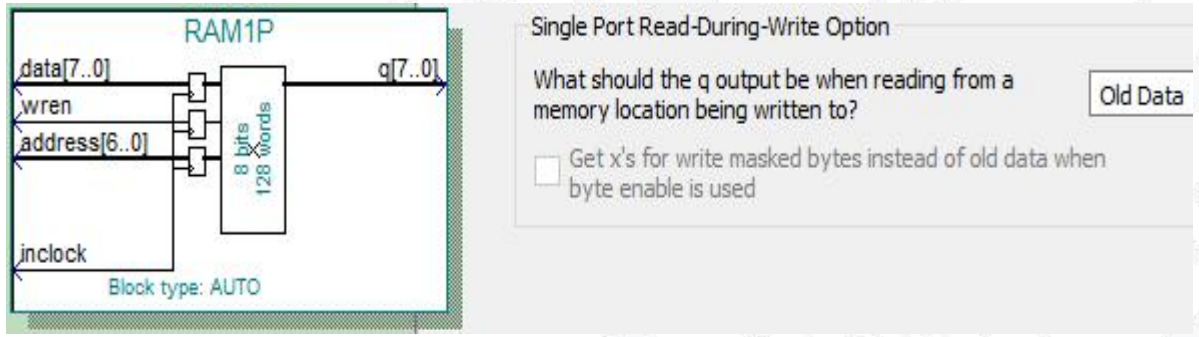
双时钟方式

6.3 LPM_RAM宏模块的设置与使用

6.3.2 LPM_RAM的设置和调用



设定RAM仅输入时钟控制

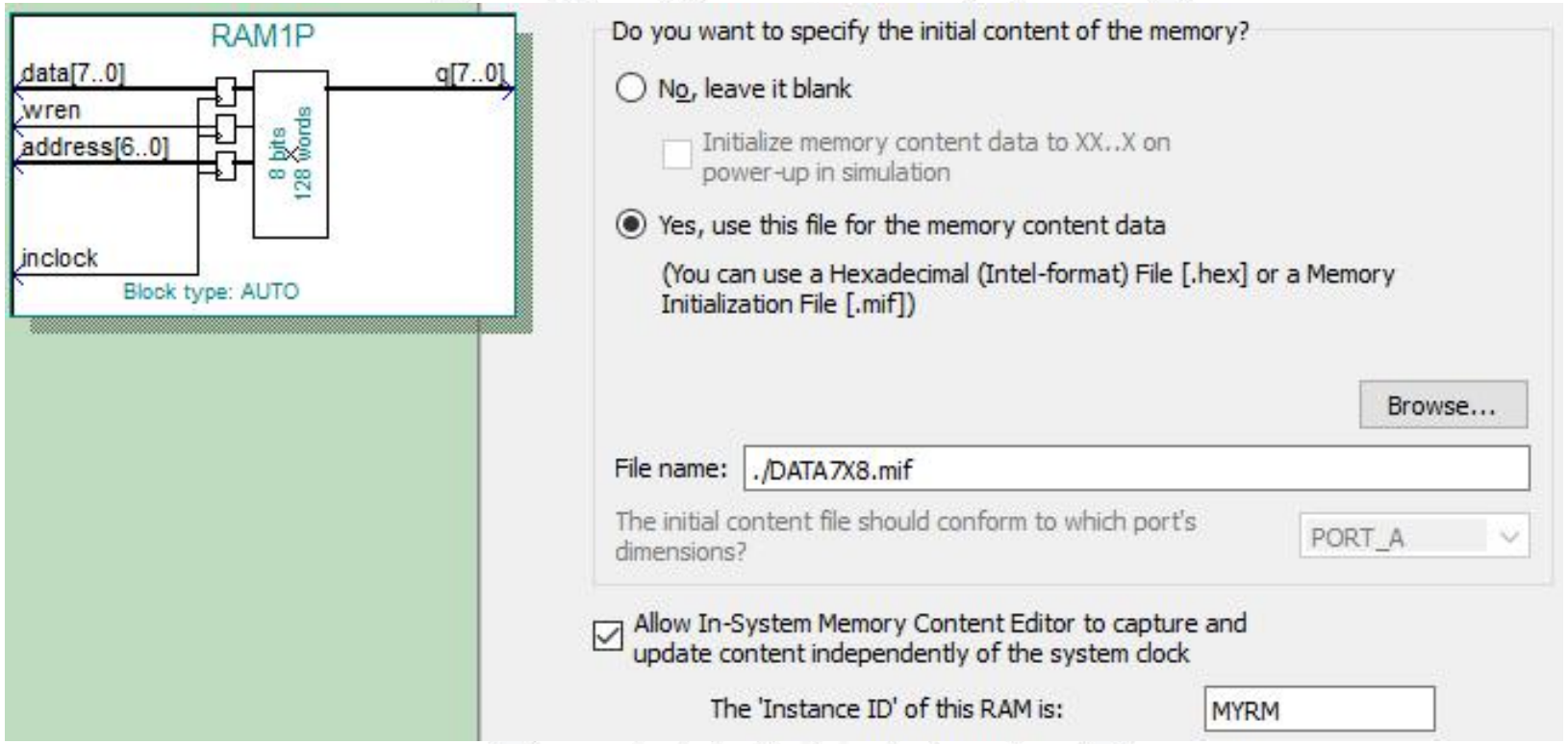


设定在写入同时读出

原数据：Old Data

6.3 LPM_RAM宏模块的设置与使用

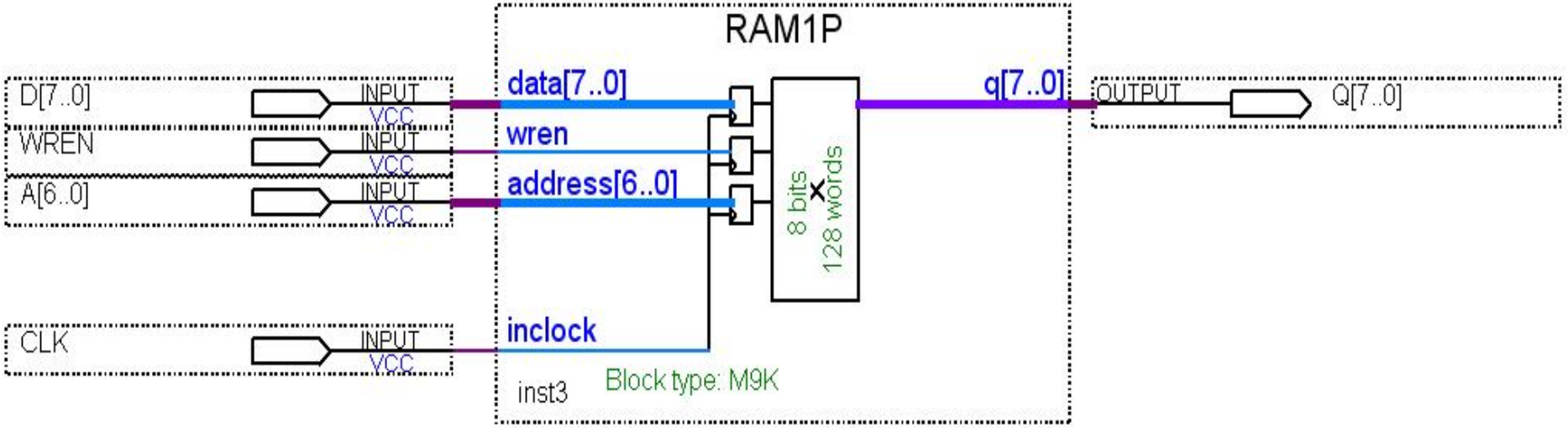
6.3.2 LPM_RAM的设置和调用



设定初始化文件和允许在系统编辑

6.3 LPM_RAM宏模块的设置与使用

6.3.2 LPM_RAM的设置和调用

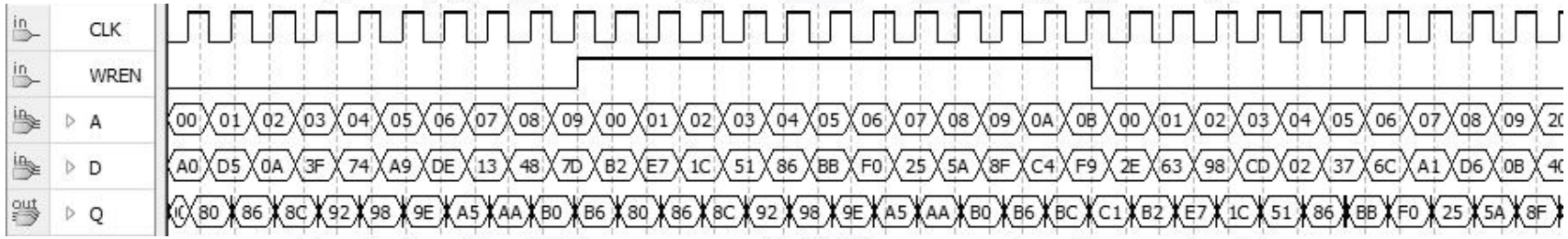


在原理图上连接好的RAM模块

6.3 LPM_RAM宏模块的设置与使用

6.3.3 仿真测试LPM_RAM

通过测试了解其各信号线的功能和加载于其中的初始化文件数据是否成功



RAM仿真波形

WREN=0，在时钟上升沿，随着地址递增，读出数据，OldData；

WREN=1，输入数据D在时钟上升沿写入；

WREN再次为0，读出写入的数据。

6.3 LPM_RAM宏模块的设置与使用



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
```

6.3 LPM_RAM宏模块的设置与使用



6.3.4 存储器的Verilog代码描述

【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
```

【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
```

EDA Lab

6.3 LPM_RAM宏模块的设置与使用



6.3.4 存储器的Verilog代码描述

这定义了一个宽度为8位、容量为1024个存储单元的存储器，取名为MEM87

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

对存储器赋值要求只能对存储器的某一单元赋值：

```
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
```

寄存器和存储器定义上的区别

```
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
```

EDA Lab

6.3 LPM_RAM宏模块的设置与使用

6.3.5 存储器设计的结构控制

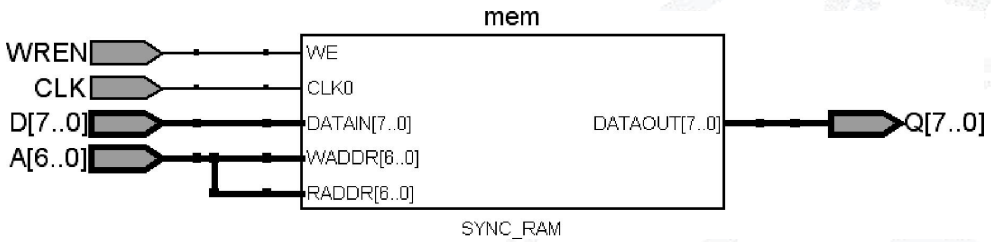


图6-19 例6-6的RTL电路模块图

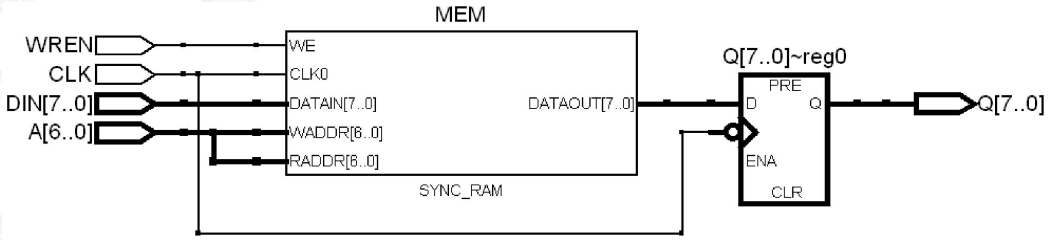


图6-20 例6-7的RTL电路模块图

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

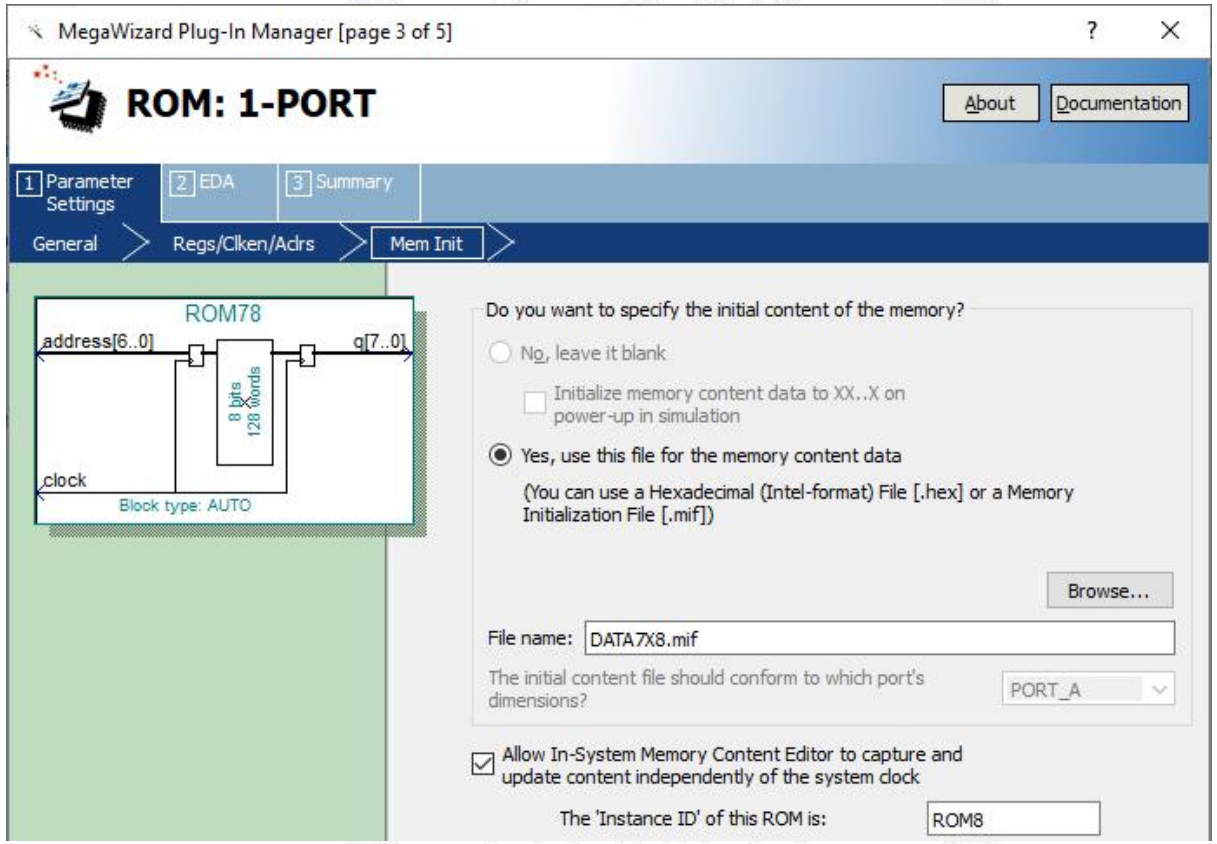
Top-level Entity Name	RAM78
Family	Cyclone III
Device	EP3C55F484C8
Timing Models	Final
Met timing requirements	N/A
Total logic elements	0 / 55,856 (0 %)
Total combinational functions	0 / 55,856 (0 %)
Dedicated logic registers	0 / 55,856 (0 %)
Total registers	0
Total pins	25 / 328 (8 %)
Total virtual 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

EDA Lab

6.4 LPM_ROM的定制和使用示例

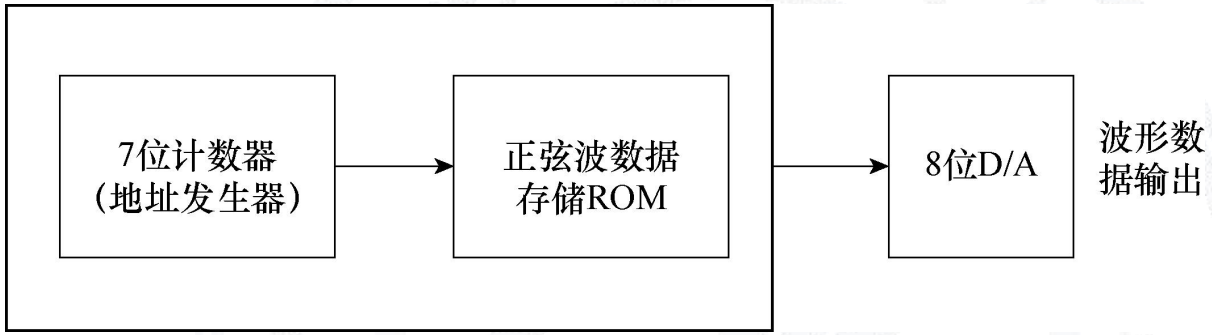
6.4.1 简易正弦信号发生器设计



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



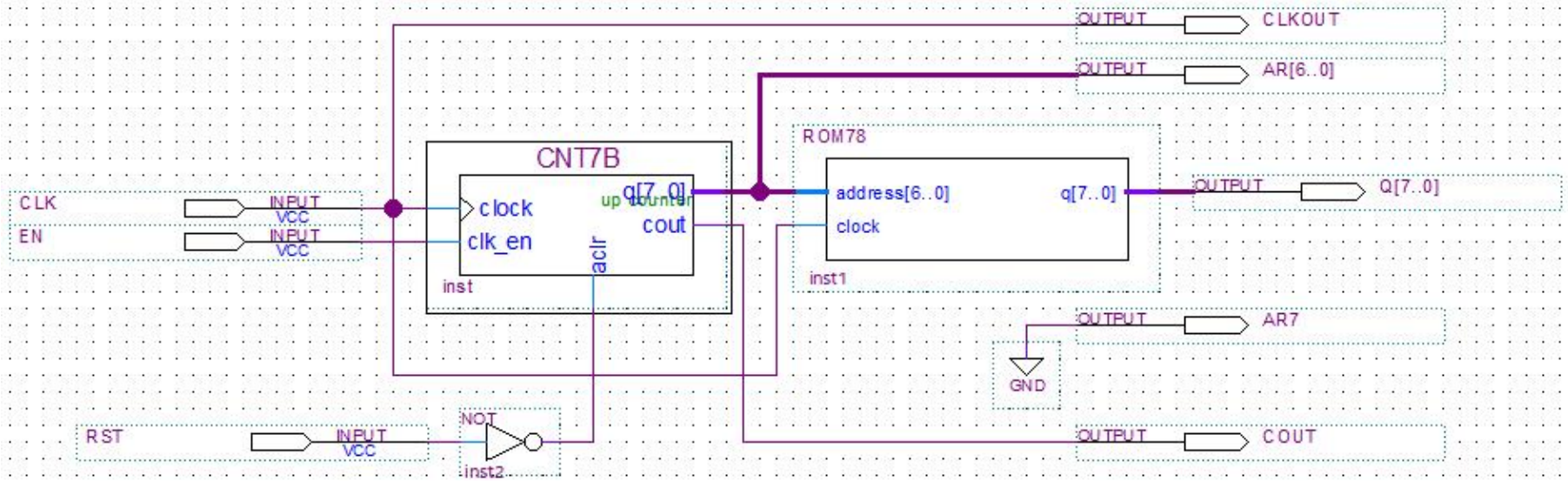
正弦信号发生器结构框图

地址信号发生器：由计数器构成。

正弦信号数据存储器ROM：含有128个8位波形数据（一个正弦波周期）。

6.4 LPM_ROM的定制和使用示例

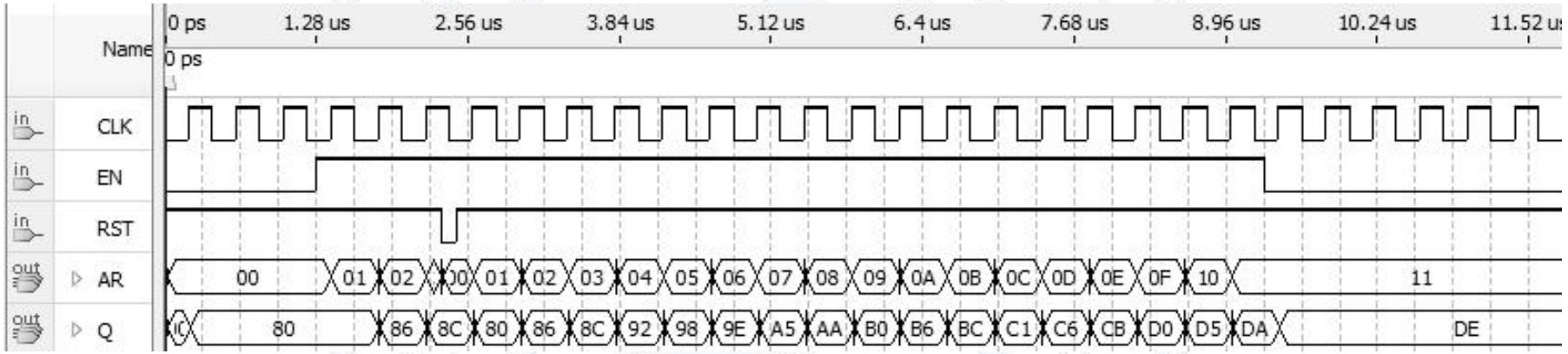
6.4.1 简易正弦信号发生器设计



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

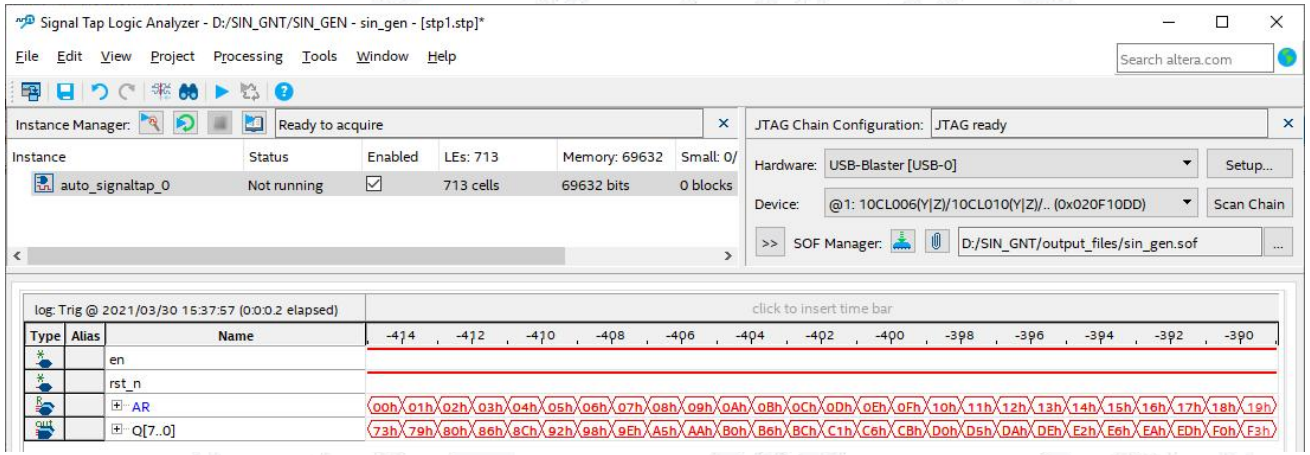


电路仿真波形

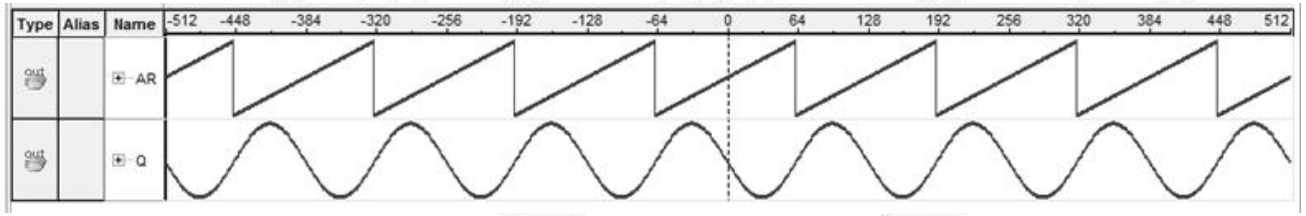
由波形可见，随着每一个时钟上升沿的到来，输出端口将正弦波数据依次输出。

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

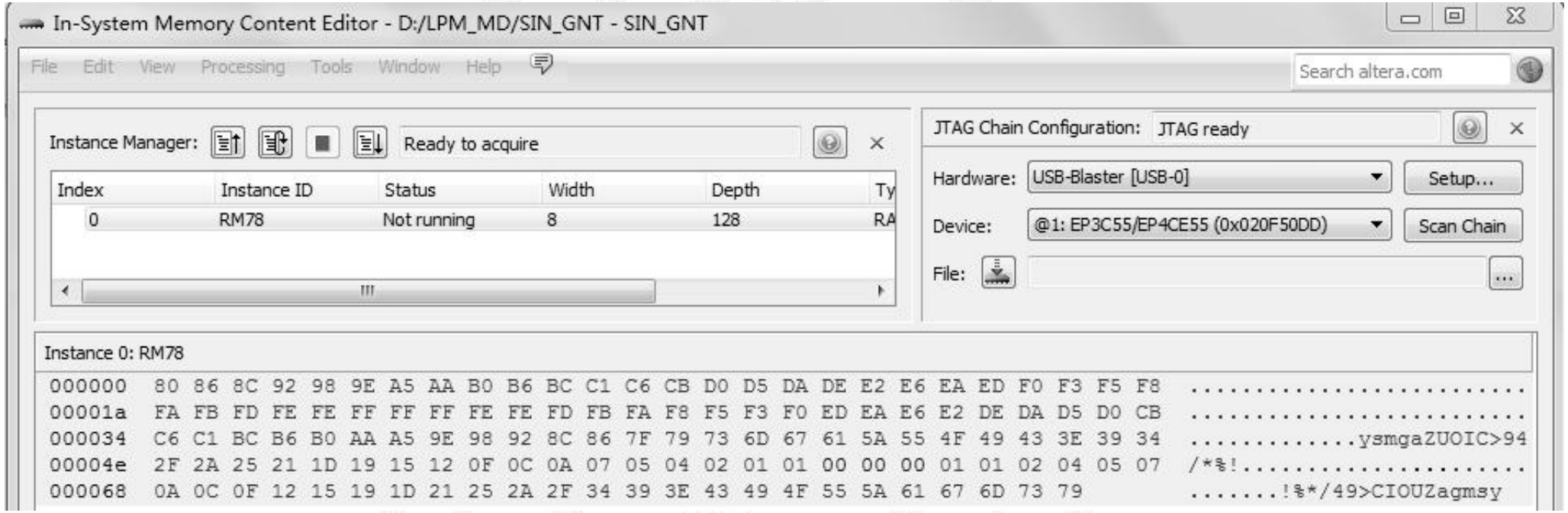


正弦信号发生器数据输出的SignalTap II实时测试界面



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



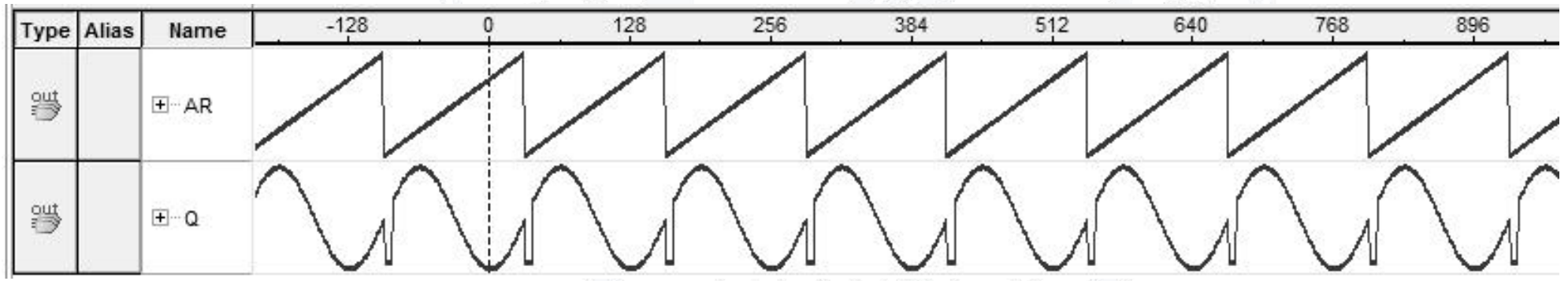
In-System Memory Content Editor编辑窗，从FPGA中的ROM读取波形数据

6.5 在系统存储器数据读写编辑器应用

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
```

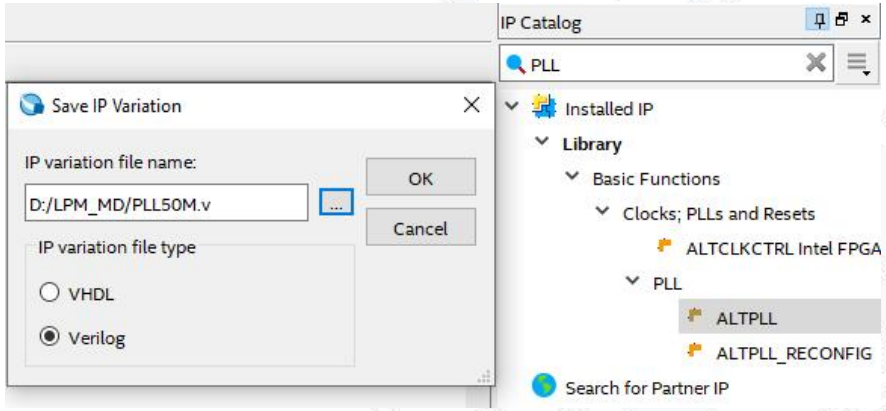
在此将编辑好的数据载入FPGA中的ROM内



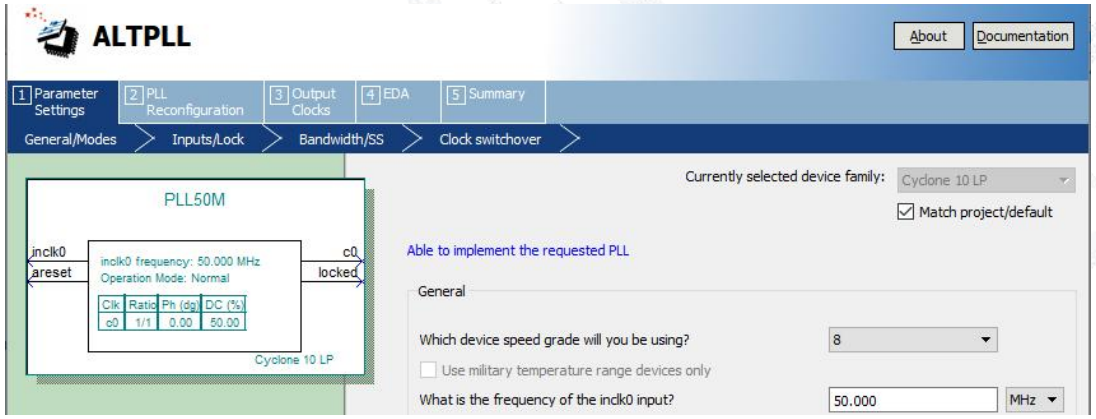
SignalTap II测得的数据波形

6.6 LPM嵌入式锁相环调用

原理图编辑窗口，Insert -> Symbol -> MegaWizard Plug-In Manager -> Create a new custom... -> Basic Functions -> ALTPLL



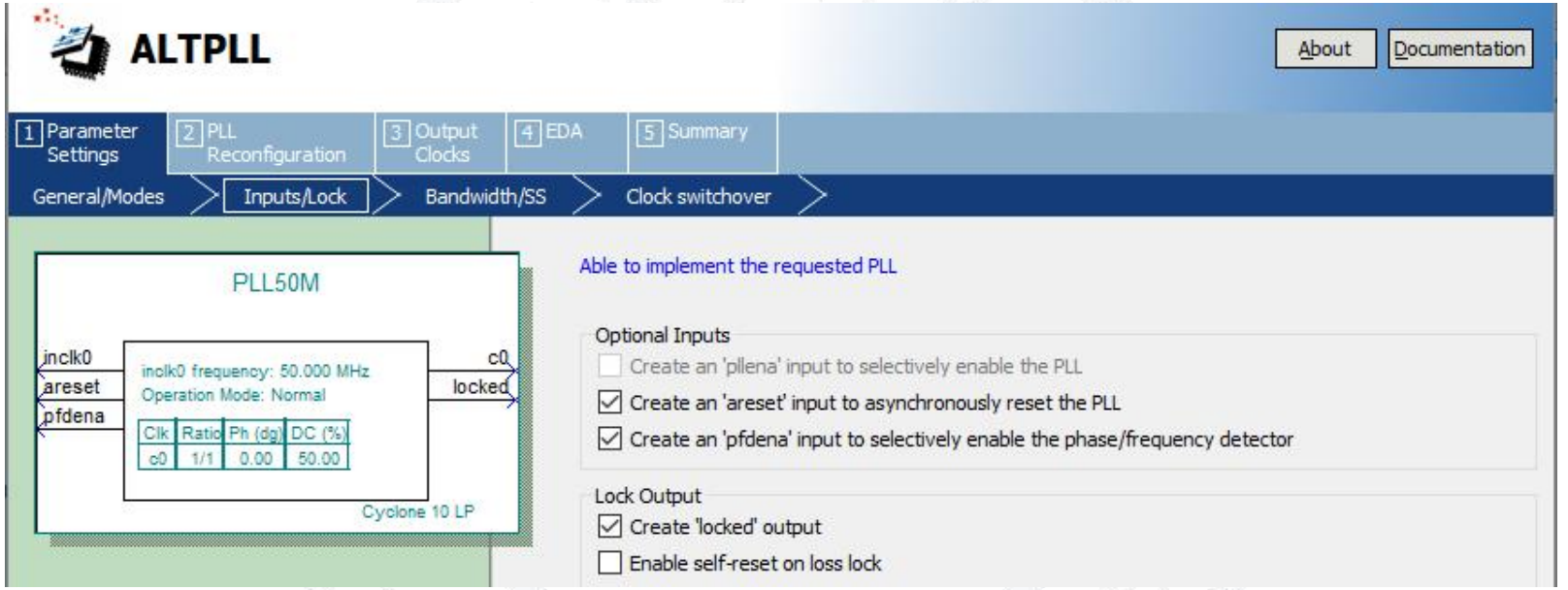
选择锁相环ALTPLL



选择输入参考时钟inclk0为50MHz

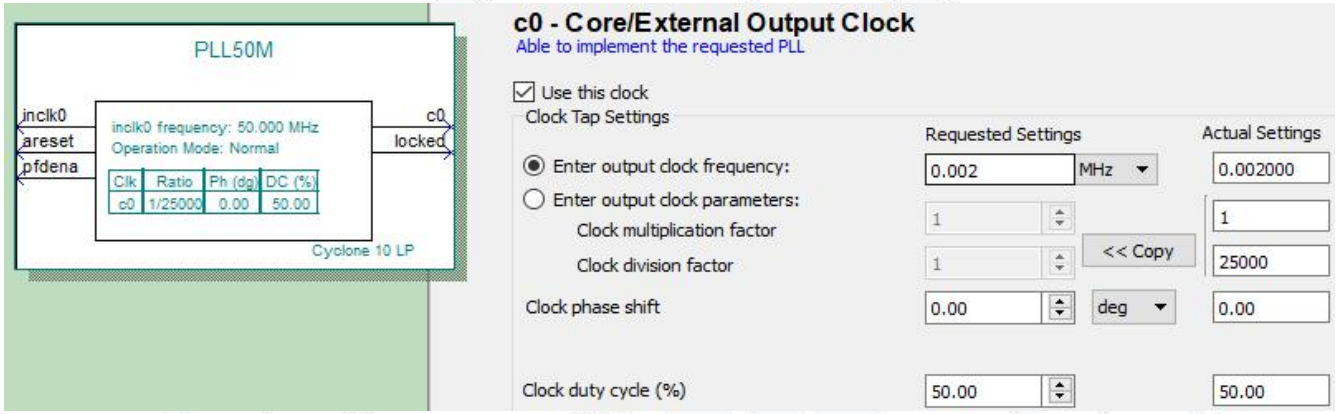
EDA Lab

6.6 LPM嵌入式锁相环调用

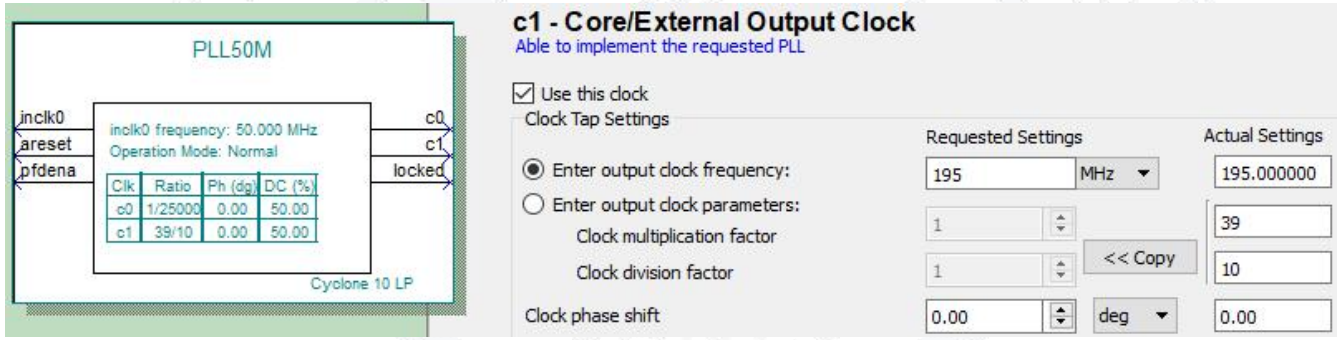


选择PLL的控制信号：PLL使能控制pllena，异步复位areset，锁相标志输出locked。

6.6 LPM嵌入式锁相环调用

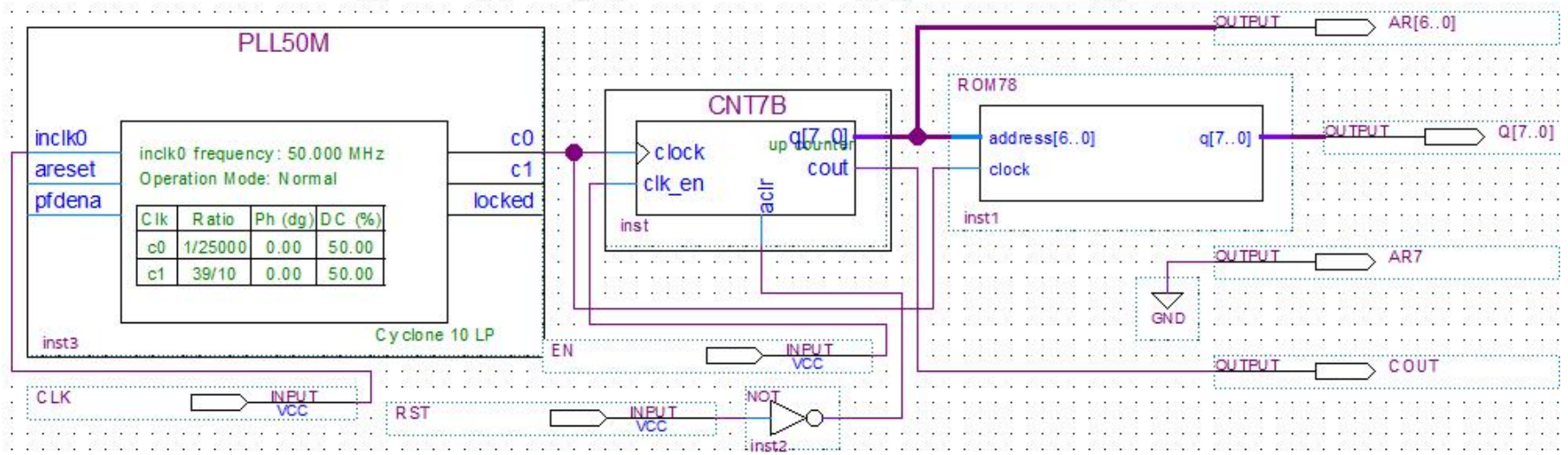


选择c0的输出频率为0.002MHz



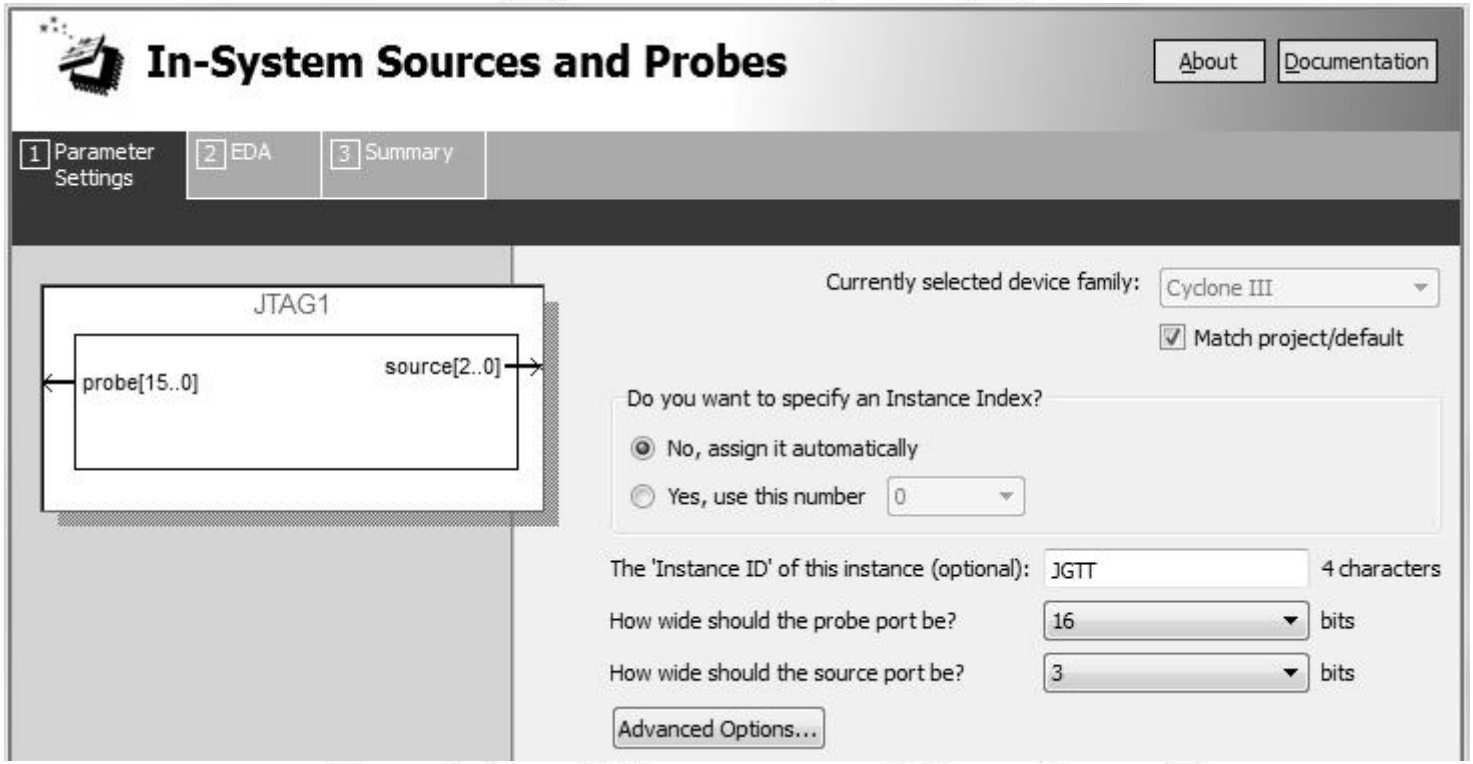
输出第二个时钟信号c1

6.6 LPM嵌入式锁相环调用



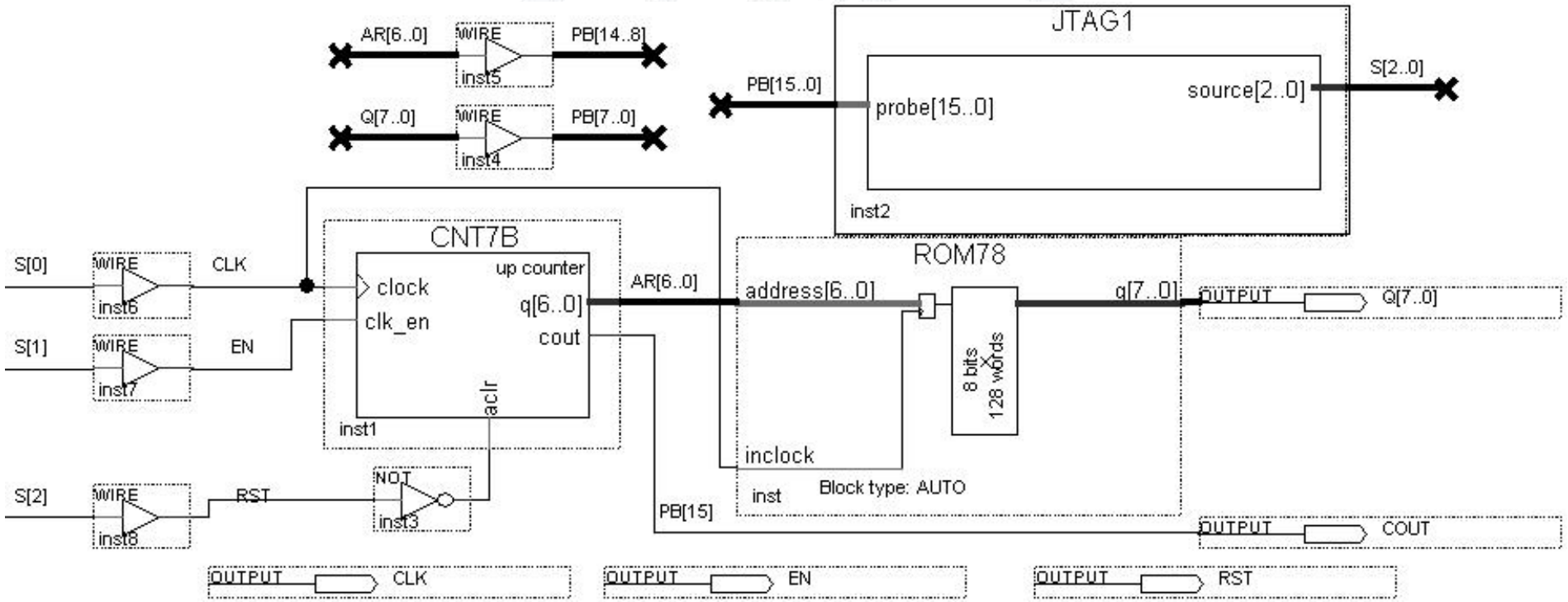
采用了嵌入式锁相环作时钟的正弦信号发生器电路

6.7 In-System Sources and Probes Editor使用方法



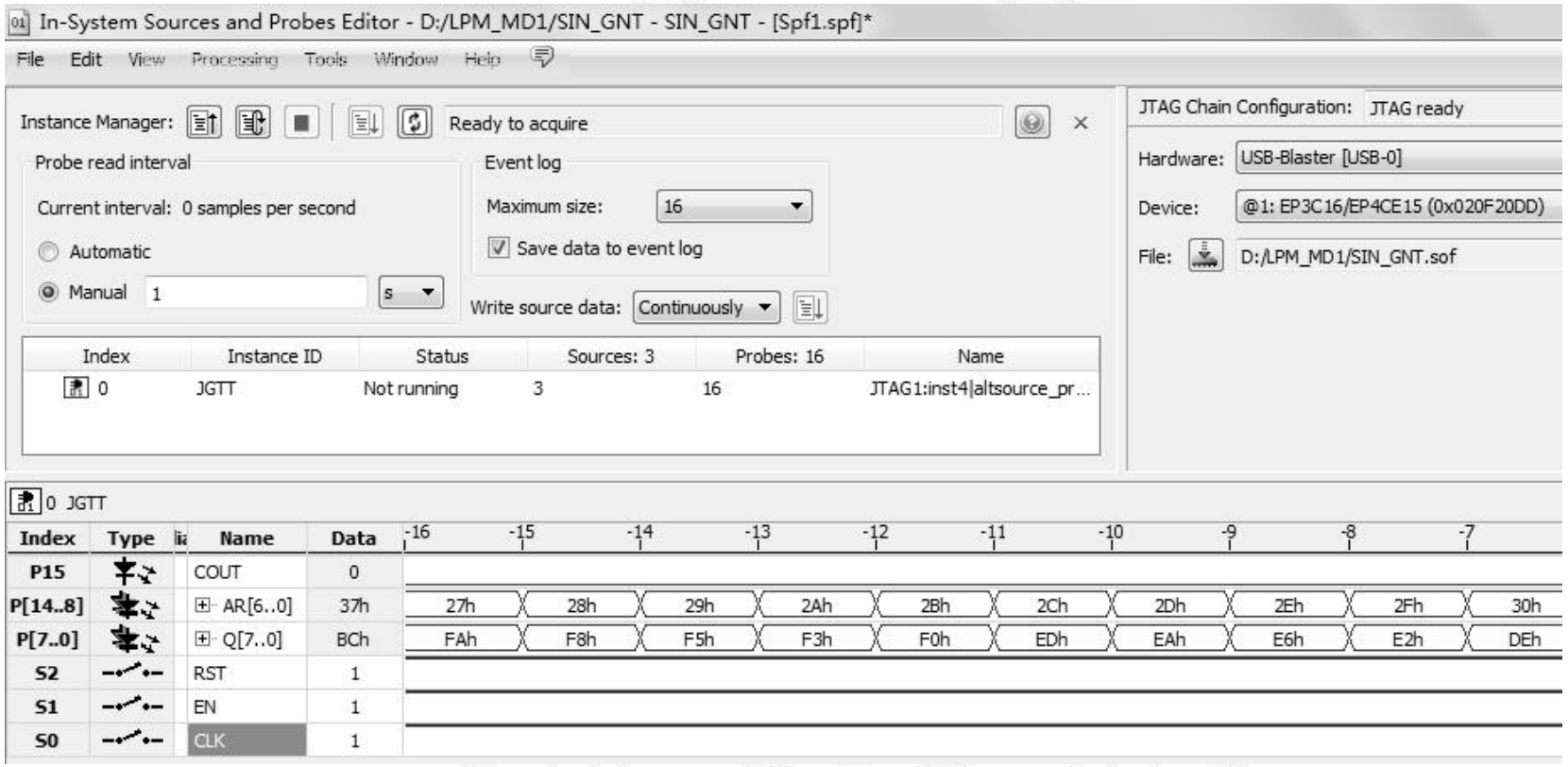
为In-System Sources and Probes模块设置参数

6.7 In-System Sources and Probes Editor使用方法



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



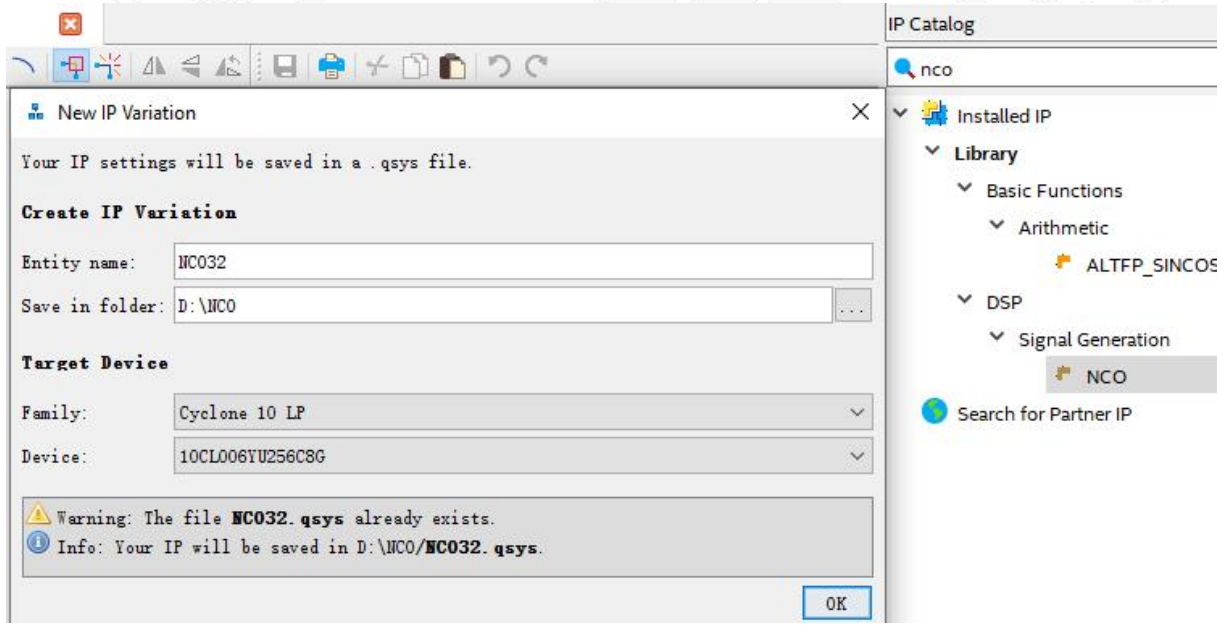
In-System Sources and Probes Editor的测试情况

6.8 NCO数控振荡器核使用方法

（1）定制**NCO**。IP Catalog ->DSP NCO

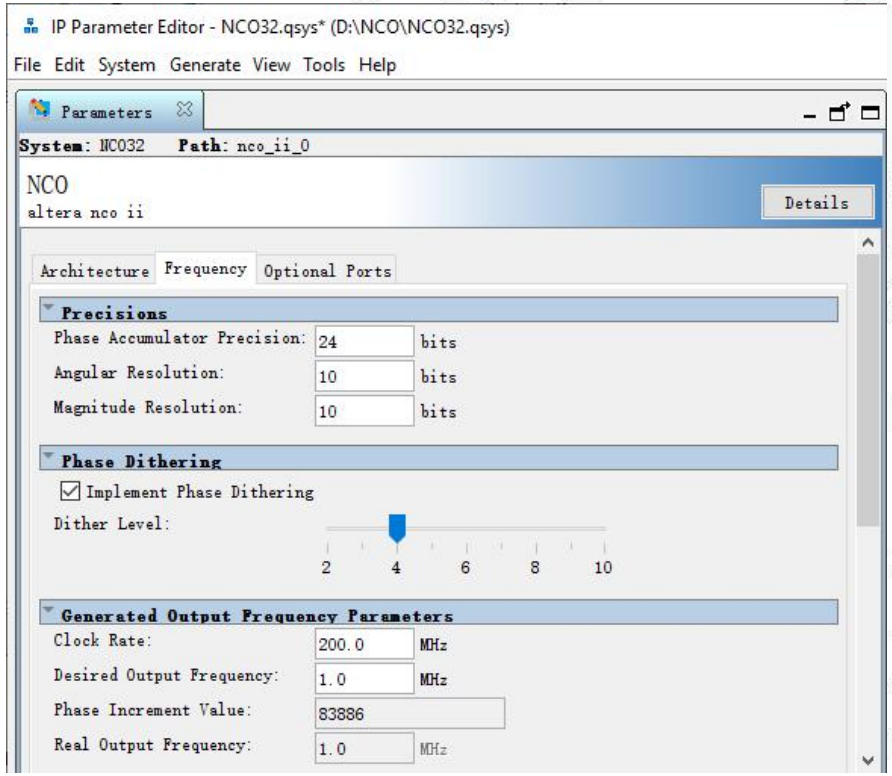
（2）进入**Core**文件生成选择窗。

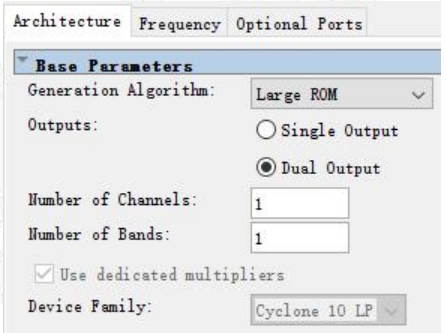
首先选择Documentation，了解此IP核的功能、设置方法、使用方法以及相关信息。

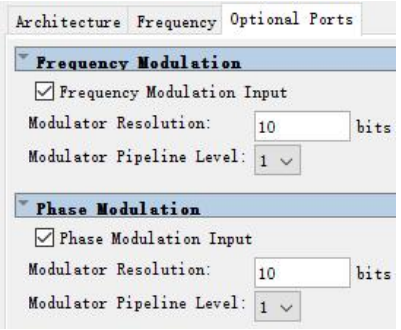


6.8 NCO数控振荡器核使用方法

（3）设置参数。





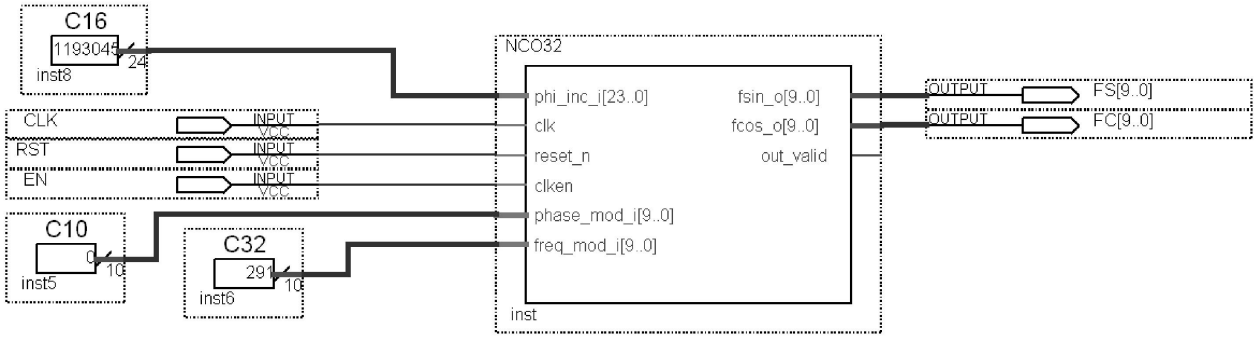


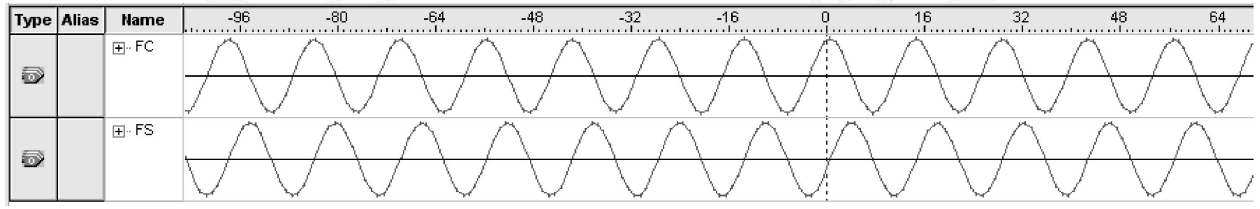
6.8 NCO数控振荡器核使用方法

（4）生成仿真文件。Step 2：Set Up Simulation

（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。



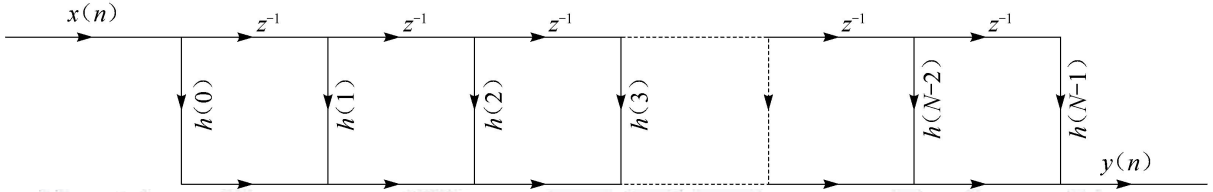


6.9 FIR核使用方法

*N*阶FIR滤波器系统的传递函数

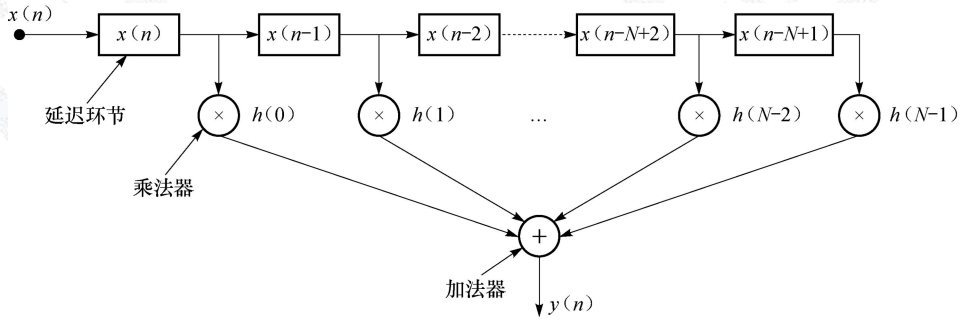
$$H(z)=\sum_{n=0}^{N-1}h(n)z^{-n}$$

对应于下图所示的FIR滤波器结构



直接型FIR滤波器结构

*N*阶的FIR系统差分方程可表示为 $Y(n)=\sum_{m=0}^{N-1}h(m)x(n-m)$

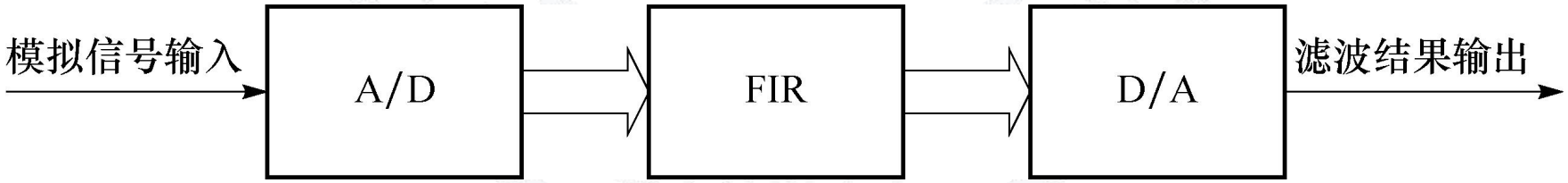


直接型FIR实现结构

EDA Lab

6.9 FIR核使用方法

- 采用Altera的MegaCore来设计一个37阶8位高速FIR滤波器。图6-49是实际应用的FIR滤波器设计示意图。
- 定制FIR核的流程与NCO核基本相同，也附带产生了VHDL和Verilog模块调用示例文件、MATLAB仿真文件（后缀为.m），以及用于Quartus仿真的向量文件

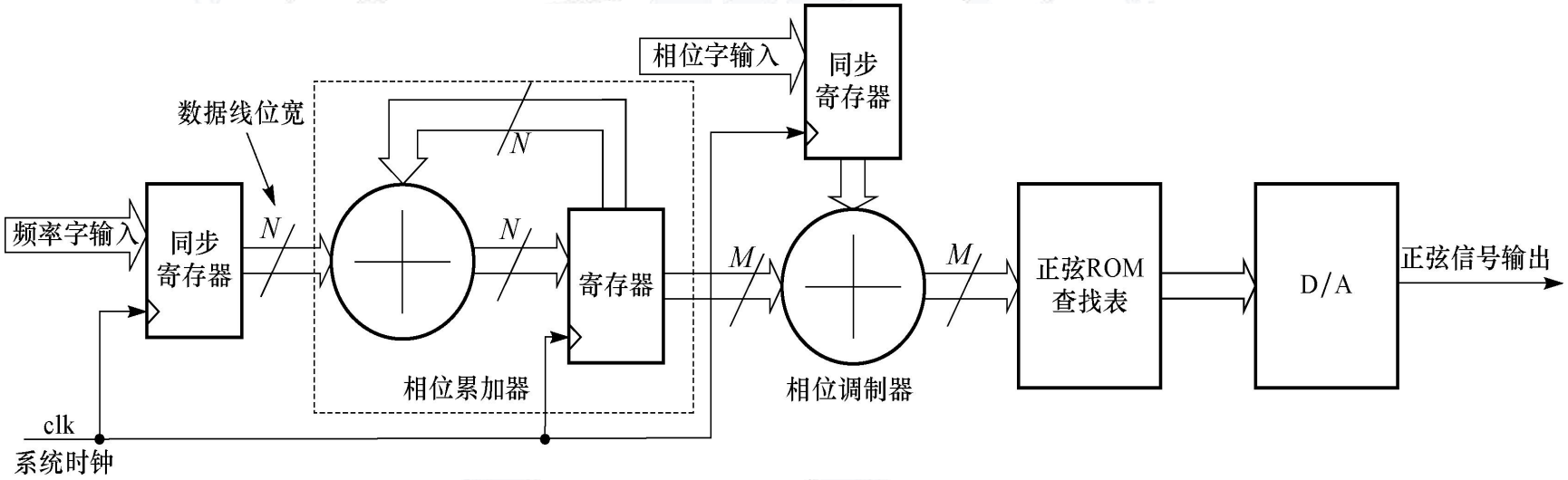


FIR滤波器设计示意

6.10 DDS实现原理与应用

6.10.1 DDS原理

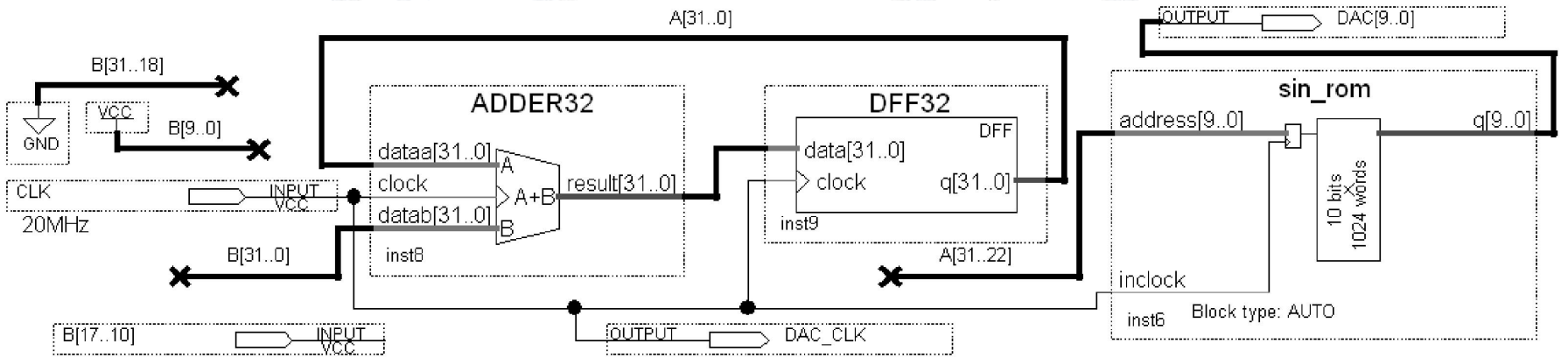
- DDS（direct digital synthesizer）即直接数字合成器，是一种新型的频率合成技术。具有较高的频率分辨率，可以实现快速的频率切换，并且在改变时能够保持相位的连续，很容易实现频率、相位和幅度的数控调制。



基本DDS结构

6.10 DDS实现原理与应用

6.10.2 DDS信号发生器设计示例



DDS信号发生器电路顶层原理图

其中相位累加器的位宽是32.

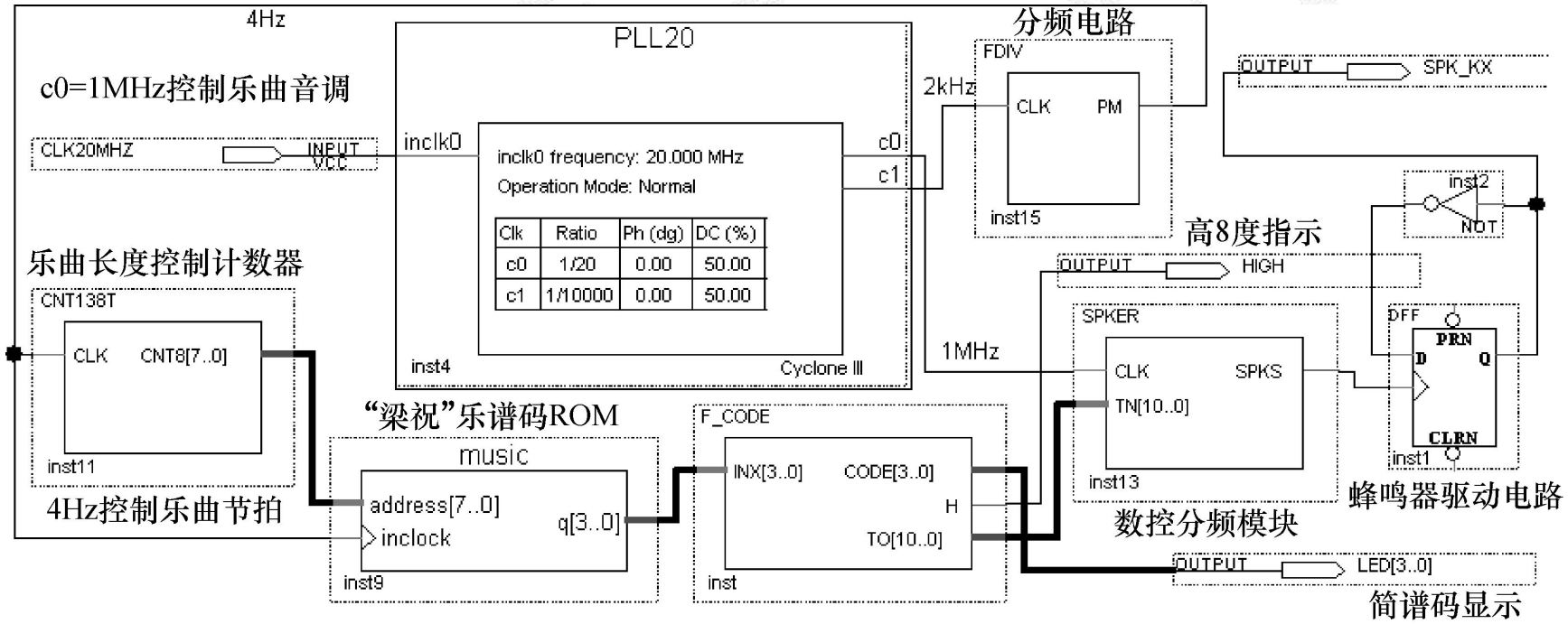
实验与设计



- 6-1 查表式硬件运算器设计
- 6-2 正弦信号发生器设计
- 6-3 DDS正弦信号发生器设计
- 6-4 简易数据采集系统设计
- 6-5 移相信号发生器设计
- 6-6 16位×16位高速硬件乘法器设计

EDA Lab

6-7 乐曲硬件演奏电路设计



乐曲演奏电路顶层设计

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
```

EDA Lab

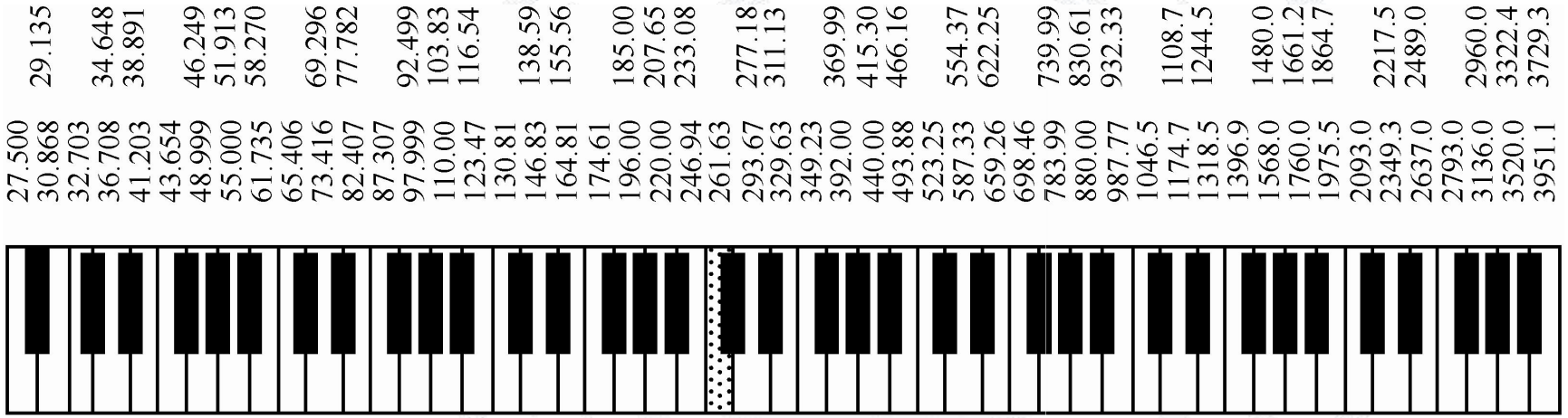
实验与设计



【例**6-10**】

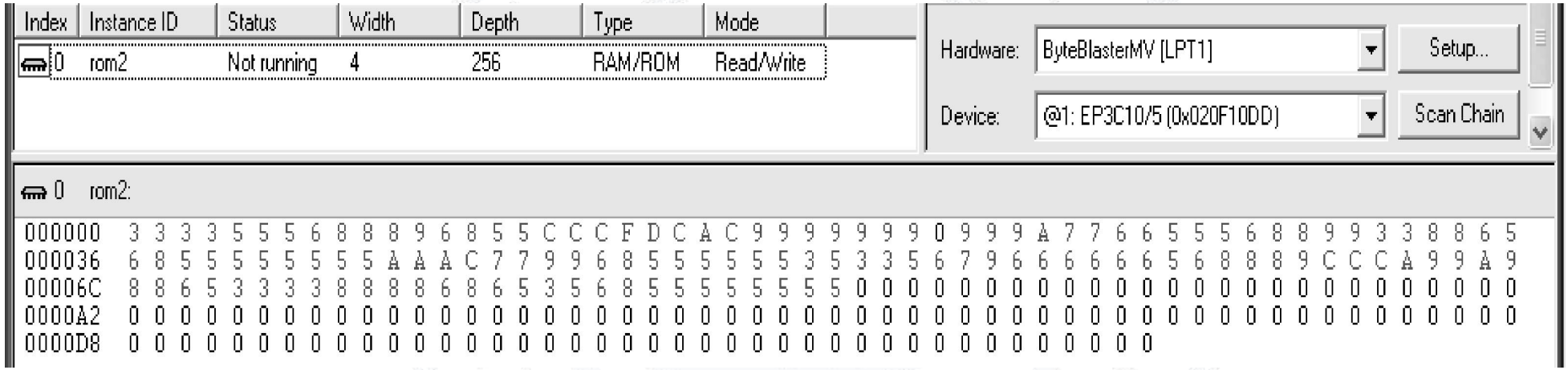
```
module F_CODE (INX, CODE, H, TO);
  input[3:0] INX; output[3:0] CODE; output H; output[10:0] TO;
  reg[10:0] TO;   reg[3:0] CODE;   reg H;
  always @(INX)   begin
  case (INX)        // 译码电路，查表方式，控制音调的预置
   0 : begin TO <= 11'H7FF; CODE<=0; H<=0; end
   1 : begin TO <= 11'H305; CODE<=1; H<=0; end
   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
```

6-7 乐曲硬件演奏电路设计



电子琴音阶基频对照图（单位Hz）

6-7 乐曲硬件演奏电路设计



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

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



【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;
```

EDA Lab